Term: M1-EE

Semester: 1

AY: 2025-2026

Abdelbacet Mhamdi

Dr.-Ing. in Electrical Engineering

Senior Lecturer at ISET Bizerte

abdelbacet.mhamdi@bizerte.r-iset.tn

Artificial Intelligence - Part 1

LAB MANUAL

Institute of Technological Studies of Bizerte

Available @ https://github.com/a-mhamdi/jlai/

Honor code

The University of North Carolina at Chapel Hill
Department of Physics and Astronomy
http://physics.unc.edu/undergraduate-program/labs/general-info/

"During this course, you will be working with one or more partners with whom you may discuss any points concerning laboratory work. However, you must write your lab report, in your own words.

Lab reports that contain identical language are not acceptable, so do not copy your lab partner's writing.

If there is a problem with your data, include an explanation in your report. Recognition of a mistake and a well-reasoned explanation is more important than having high-quality data, and will be rewarded accordingly by your instructor. A lab report containing data that is inconsistent with the original data sheet will be considered a violation of the Honor Code.

Falsification of data or plagiarism of a report will result in prosecution of the offender(s) under the University Honor Code.

On your first lab report you must write out the entire honor pledge:

The work presented in this report is my own, and the data was obtained by my lab partner and me during the lab period.

On future reports, you may simply write "*Laboratory Honor Pledge*" and sign your name."

Contents

In order to activate the virtual environment and launch **Jupyter Notebook**, you need to proceed as follow

① Press simultaneously the keys CTRL ALT and T on the keyboard;

② Type `jlai1` in the console prompt line;

Terminal

```
student@isetbz:~$ jlai1
```

③ Finally hit the ↵ key.

KEEP THE SYSTEM CONSOLE OPEN.

▼ Remark 1

You should be able to utilize Julia from within the notebook through:

Jupyter Lab *at http://localhost:2468*

Pluto *at http://localhost:1234*

Submit your lab reports in **PDF** format. Be sure to review your files with your instructor before the lab session ends.

https://forms.gle/sHQgBigF6hFFYqD16

Please use one of the provided templates when preparing your lab assessments:

LaTeX `https://www.overleaf.com/read/pwgpyvcxcvym#9e34eb`

Typst `https://typst.app/universe/package/ailab-isetbz`

A. Mhamdi

1 | *Julia* Onramp

Student's name	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

Goals

★ Learn the essentials of *Julia* on commonly used features & workflows.

The notebook is available at `https://github.com/a-mhamdi/jlai` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-1* ⟶ *Jupyter* ⟶ *julia-onramp.ipynb*

Task № 1:

a) Develop a `Julia` console application that computes electricity bills for standard low-voltage customers in Tunisia based on **STEG**'s official tariff structure. The application should reference the official tariff information and provide accurate bill calculations with detailed cost breakdowns for verification and transparency.

b) Implement the same electricity bill calculation functionality using `KNIME`, creating a workflow that produces equivalent results to the Julia application.

▼ Remark 2

To Launch KNIME *Application, go to your already opened terminal and type* `knime` *in a new tab.*

2 | Tipping Problem

Student's name	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

Goals

- ★ Analyze and comprehend the code examples for the tipping problem implemented in `Julia` and `Matlab`.
- ★ Design and implement an equivalent fuzzy logic system for tipping using `KNIME`.
- ★ Test and evaluate all implementations with various service and food quality ratings.
- ★ Accurately display and interpret the calculated tip for each test scenario.

The notebook is available at `https://github.com/a-mhamdi/cosnip/` → *Matlab* → *Fuzzy* → *Tipper.**

Matlab is a powerful tool widely used by engineers. To use *Matlab*, you primarily need a basic understanding of the problem at hand and some familiarity with its syntax—there is no need to spend extra time configuring your environment. Serving as both an interactive matrix calculator and a comprehensive programming platform, *Matlab* simplifies both general-purpose tasks and those requiring high precision. Its design allows you to concentrate on solving your problem, rather than worrying about the intricacies of programming itself.

Matlab offers an intuitive, user-friendly environment and is classified as a fourth-generation programming language[1] (4GL). As a very high-level language, it makes the programming process straightforward and efficient.

Desktop layout

Similar to other development environments, *Matlab* features a menubar and a toolbar where users can access essential commands and customize their preferences, determining how *Matlab* looks and operates. The main interface is organized into four primary sections, as illustrated in Fig. 2.1.

Area 1

This area is the Command Window, where users can enter `Matlab` commands and immediately view the results on the same screen.

Area 2

This section is called the Workspace. It displays all variables currently in memory. To view detailed information about these variables, simply enter the following command in the Command Window:

```
>> whos
```

The results are the names of variables, the min and max values for every variables, class and the number of bytes needed to save it.

Area 3

This section is the Current Folder, which displays the directory path where `Matlab` is currently running. To run a script or function, the file must be located in this folder or in a folder that is on Matlab's search path. If you need to execute code located elsewhere, you should change the Current Folder to the appropriate directory; otherwise, `Matlab` will display an error in the Command Window indicating that it cannot find the file.

Area 4

This section is the Command History. Here, every command you execute in `Matlab` is automatically recorded and displayed, allowing you to review all instructions entered since the software was installed on your system up to the present moment.

[1] http://en.wikipedia.org/wiki/Fourth-generation_programming_language

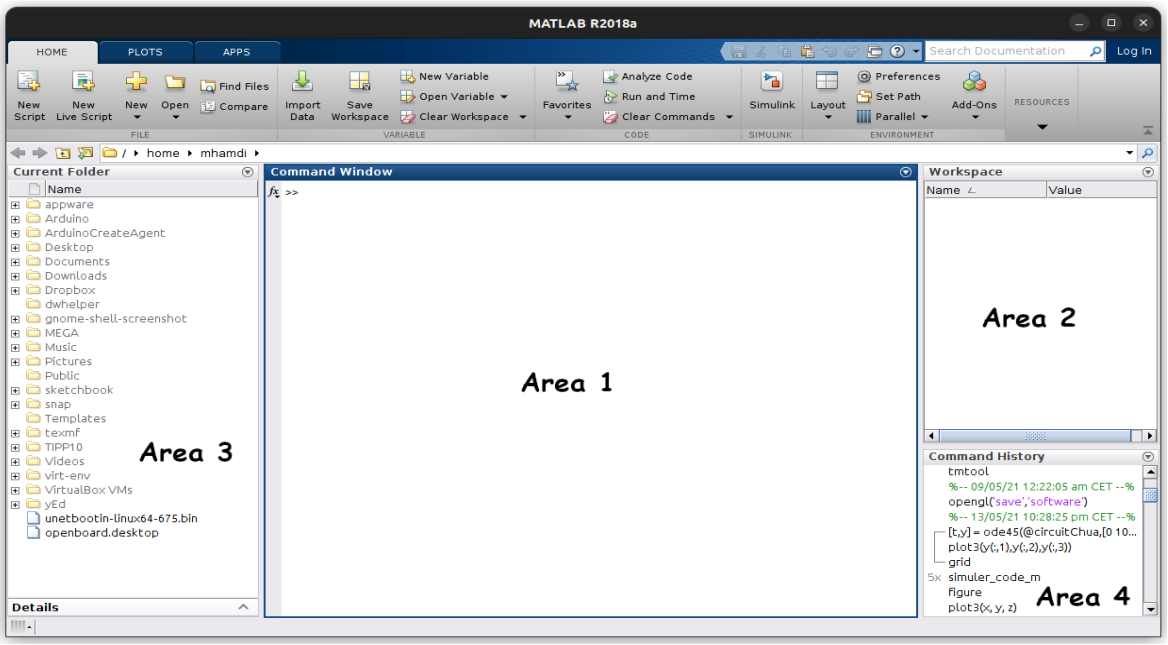

Figure 2.1: `Matlab` UI

How to personalize the desktop layout

To customize the desktop layout in `Matlab`, go to the menu bar and select **Layout**. From there, you can load your saved preferences. For instance, if you have previously saved a personalized layout as My Layout, simply click on it to apply your configuration. The resulting arrangement will appear as illustrated in Fig. 2.2.

Matlab is not only a platform for technical computing and solving equations—it also provides tools for creating graphical user interfaces (GUIs). The primary integrated tool for this purpose is called **GUIDE** (Graphical User Interface Development Environment).

To access the GUIDE environment, enter the following command in the Command Window:

```
>> guide
```

Alternatively, you can access a shortcut for GUIDE by hovering your mouse over the toolbar—you will see an icon like this:  . Clicking this icon will launch the graphical environment, displaying a dialog as shown. From here, you can select a template or simply click “Next” to proceed. Now, the environment is ready to be used. As shown in Fig. 2.3 and Fig. 2.4.

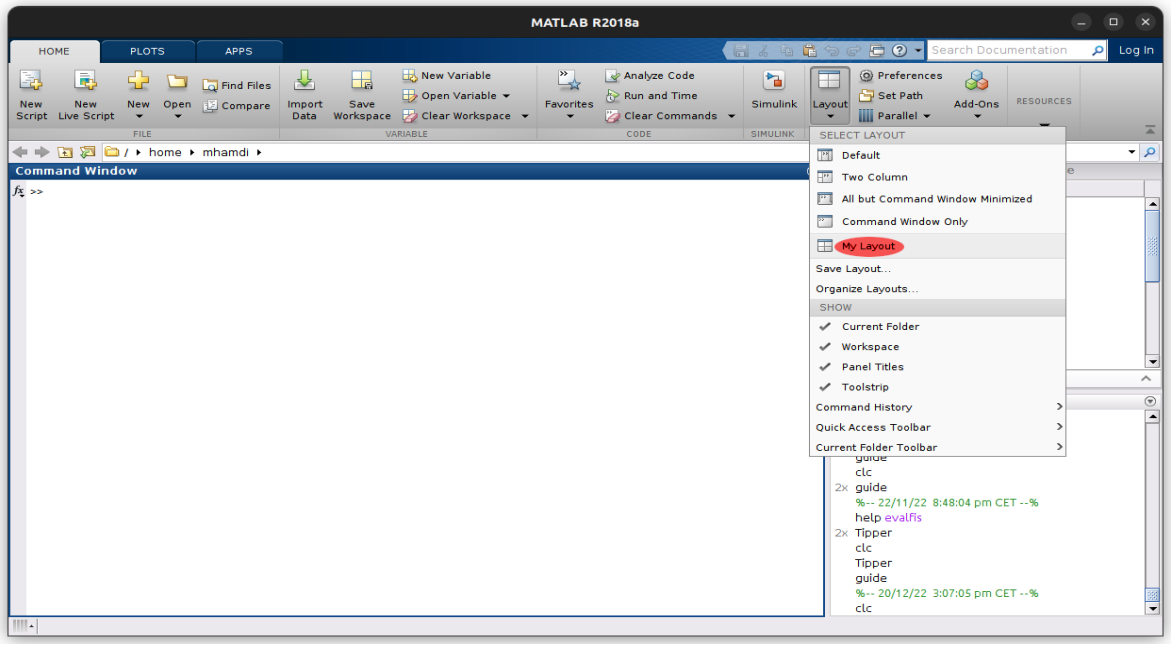

Figure 2.2: `Matlab` layout

After saving your work, you will see that two files are created in your directory with the same name but different extensions: one with the extension **.fig* and the other as an *m-file*.

The *.fig* file stores the graphical layout and instantiation of all GUI components you have used. The main interface provides a palette with easily accessible, preconfigured elements such as "pushbutton", "toggle button", and "slider". If needed, you can add additional controls using ActiveX, although that is beyond the scope of this discussion.

For each object in your GUI, you can use the Object Browser to view and edit its properties as well as see the associated methods. If you want to group similar elements, consider using a *uipanel* to organize them visually.

Once your design is complete, you can launch and interact with your application. An example of a final interface is shown in Fig. 2.5.

It is preferable to load the fuzzy inference system, denoted hereafter by `fis`, in the opening function of the gui.

```
% --- Executes just before Tipper is made visible.
function Tipper_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

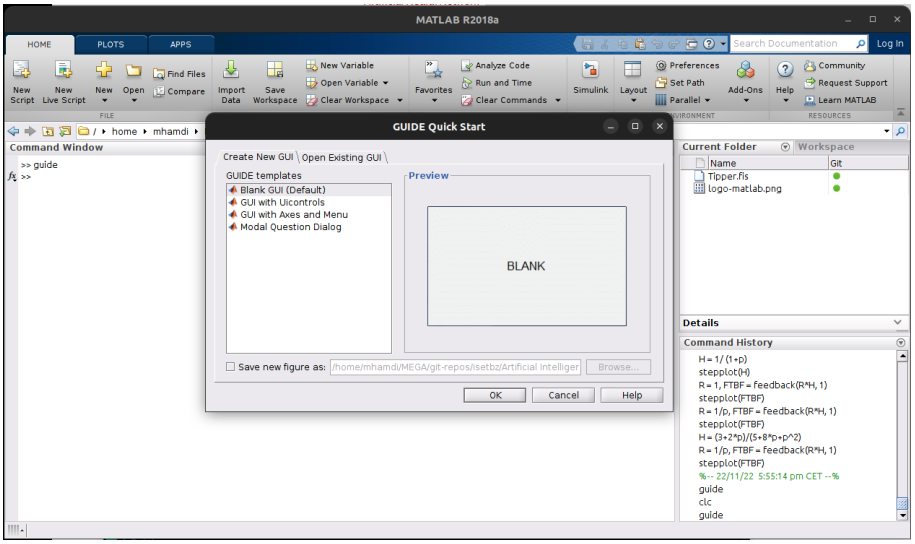

Figure 2.3: GUIDE environment - Step 1

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Tipper (see VARARGIN)
fis = readfis('Tipper.fis');
handles.fis = fis;
```

Before, quitting the function, you have to update the GUI and save all variables in the general structure with handles and user data[2].

```
% Choose default command line output for Tipper
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Tipper wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Executes on button press in compute.
function compute_Callback(hObject, eventdata, handles)
% hObject    handle to compute (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

[2]Visit: http://www.mathworks.com/help/matlab/gui-building-basics.html

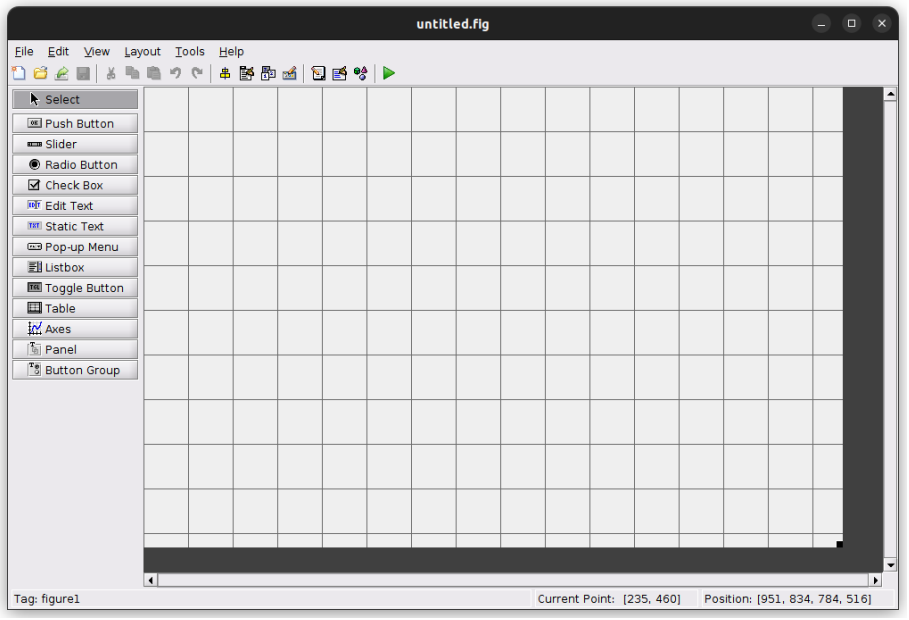

Figure 2.4: GUIDE environment - Step 2

```
% handles    structure with handles and user data (see GUIDATA)
foodVal = str2num(get(handles.food, 'String'));
serviceVal = str2num(get(handles.service, 'String'));

fis = handles.fis;
tip = evalfis([foodVal, serviceVal], fis);

set(handles.tip, 'String', num2str(tip));
```

The ultimate application will behave like the example displayed in the following figures. Enter your desired values and then just click **"Compute"**. The result will be displayed on the right half side of the interface as indicated in FIG. 2.6.

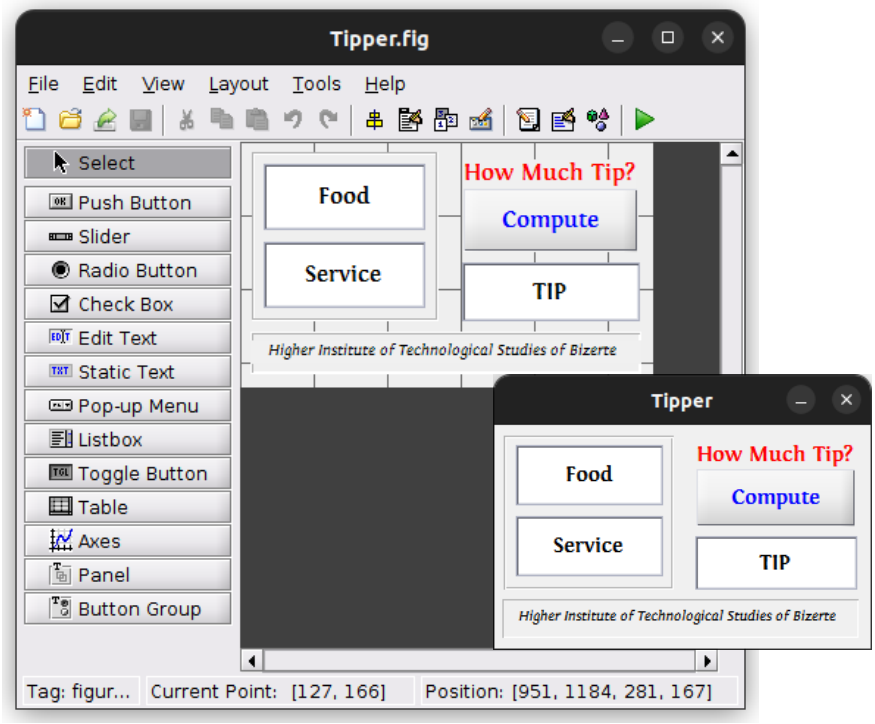

Figure 2.5: Final interface

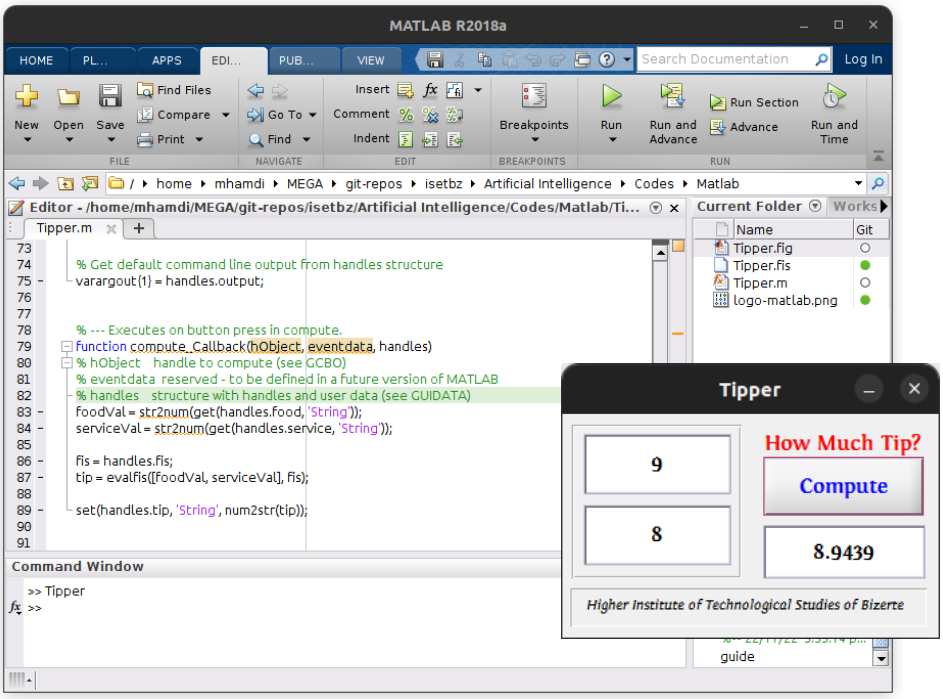

Figure 2.6: Numeric simulation

Julia Code

The notebook is available at https://github.com/a-mhamdi/jlai/ → *Codes* → *Julia* → *Part-1* → *Jupyter* → *tipper.ipynb*

Builtin Membership Functions

Using the '*Fuzzy.jl*' package, there are five types of fuzzifiers:

Triangular fuzzifier

```
    TriangularMF(l_vertex, center, r_vertex)

- `l_vertex`, `center`, and `r_vertex` are the vertices of the
  triangle, in order.
```

Gaussian fuzzifier

```
    GaussianMF(center, sigma)

- `center` is the center of the distribution;
- `sigma` determines the width of the distribution.
```

Generalised Bell fuzzifier

```
    BellMF(a, b, c)

- `a`, `b`, and `c` are the usual bell parameters with `c`
  being the center
```

Trapezoidal fuzzifier

```
    TrapezoidalMF(l_bottom_vertex, l_top_vertex, r_top_vertex,
       r_bottom_vertex)

- `l_bottom_vertex`, `l_top_vertex`, `r_top_vertex` and `
  r_bottom_vertex` are the vertices of the trapezoid, in order
  .
```

Sigmoid fuzzifier

```
    SigmoidMF(a, c, limit)

- `a` controls the slope;
- `c` is the crossover point;
- `limit` sets the extreme limit.
```

```
using Markdown

md"Import required librairies"
using Fuzzy
using Plots

md"`score` denotes the horizontal axis"
score = range(0, 10, length=100)

md"`food` is the 1st fuzzy input"
food = Dict(
            "Rancid" => TrapezoidalMF(0, 0, 2, 4),
            "Delicious" => TrapezoidalMF(6, 8, 10, 10)
            )
food_chart = chart_prepare(food, score)

md"`service` is the 2nd fuzzy input"
service = Dict(
            "Poor" => TrapezoidalMF(0, 0, 2, 4),
            "Good" => TrapezoidalMF(3, 4, 6, 7),
            "Excellent" => TrapezoidalMF(6, 8, 10, 10)
            )
service_chart = chart_prepare(service, score)

md"`tip` is the fuzzy output"
tip = Dict(
           "Cheap" => TrapezoidalMF(0, 0, 1, 3),
           "Average" => TrapezoidalMF(2, 4, 6, 8),
           "Generous" => TrapezoidalMF(7, 9, 10, 10)
            )
tip_chart = chart_prepare(tip, score)

md"Design the rules set"
rule_1 = Rule(["Rancid", "Poor"], "Cheap", "MAX")
rule_2 = Rule(["", "Good"], "Average", "MAX")
rule_3 = Rule(["Delicious", "Excellent"], "Generous", "MAX")

md"`rules` aggregates all individual rules"
rules = [rule_1, rule_2, rule_3]

md"Plot the fuzzy membership variables"
#= GRAPHS =#
```

```
p1 = plot(score, food_chart["values"], ylabel="Food", label=food_chart["names"], legend=:bottomright)
p2 = plot(score, service_chart["values"], ylabel="Service", label=service_chart["names"], legend=:bottomright)
p3 = plot(score, tip_chart["values"], xlabel="Score", ylabel="Tip", label=tip_chart["names"], legend=:bottomright)
graphs = plot(p1, p2, p3, layout=(3, 1), lw=2)
# savefig(graphs, "mf-graphs.pdf")

md"## FUZZY INFERENCE SYSTEM: MAMDANI"
fis = FISMamdani([food, service], tip, rules)
eval_fis(fis, [9., 8.])
```

Task № 2:

Develop a workflow in KNIME that implements the fuzzy logic system for tipping described in the previous exercise. The workflow should include the following components:

a) Input nodes for service and food quality ratings.

b) Fuzzy membership function nodes for converting crisp ratings into fuzzy sets.

c) Fuzzy inference nodes for applying the fuzzy rules.

d) Defuzzification nodes for converting fuzzy outputs into crisp tip amounts. 4

e) Output nodes for displaying the calculated tip.

3 | Selection Process

Student's name	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

Goals

- ★ Identify and justify the main evaluation criteria for candidate selection (such as academic record, relevant skills, and motivation).
- ★ Define suitable fuzzy membership functions for each criterion and formulate the fuzzy rules to reflect the admission logic.
- ★ Implement the fuzzy decision system and illustrate its operation with example candidate data, including a discussion on the system's outputs.

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-1* ⟶ *Jupyter* ⟶ *selection-process.ipynb*

You are asked to design an algorithm that allows picking the best candidates out of the applicants applying for some master program.

In order to keep things simple, we were given only two criteria to judge:

1. their score when submitting their application;

2. their score in the interview.

```
using Markdown

md"Let's begin by importing the `Fuzzy` module. `Plots` is used later to draw the membership functions."
using Plots
using Fuzzy

md"""
## Input
We denote later by `input` all the plausible values of concern in each particular situation. `input` is often referred to as the universe of discourse or universal set $(u)$.
"""
input = range(0, 10, length = 1000);

md"""
## Application
The first criterion to be used in our case is `application`. This latter represents the score given for a submitted application. We thought of using four membership functions to describe the status of any particular submission:
1. Weak
1. Moderate
1. Good
1. Strong
"""
application = Dict(
        "Weak" => TrapezoidalMF(0, 0, 2, 4),
        "Moderate" => TrapezoidalMF(2, 4, 5, 7),
        "Good" => TrapezoidalMF(4, 6, 7, 9),
        "Strong" => TrapezoidalMF(7, 9, 10, 10)
)

md"In order to better understand the fuzzyfication process, let's plot the chart describing `application`."
data_application = chart_prepare(application, input)
plot(
        input, data_application["values"],
        label=data_application["names"],
        legend=:bottomleft
)
```

```
35
36  md"""
37  ## Interview
38  The variable `interview` describes the score given to an apllicant␣
     ↪after passing the interview test.
39  """
40  interview = Dict(
41          "A" => TriangularMF(0, 0, 2),
42          "B" => TriangularMF(1, 4, 6),
43          "C" => TriangularMF(5, 8, 10),
44          "D" => TriangularMF(9, 10, 10)
45  )
46  data_interview = chart_prepare(interview, input)
47  plot(
48          input, data_interview["values"],
49          label=data_interview["names"],
50          legend=:bottomright
51  )
52
53  md"It is time now to design the variable `criteria` which aggregates␣
     ↪both `application` and `interview`."
54  criteria = [application, interview]
55
56  md"""
57  ## Decision
58  As for the output, we designate by `decision` the final status of any␣
     ↪given application.
59  """
60  decision = Dict(
61          "Rejected" => TrapezoidalMF(0, 0, 2, 7),
62          "Accepted" => TrapezoidalMF(3, 8, 10, 10)
63  )
64  data_decision = chart_prepare(decision, input)
65  plot(
66          input, data_decision["values"],
67          label=data_decision["names"],
68          legend=:inside
69  )
70
71  md"""
72  ## Set of Rules
73  """
74
75  begin
```

```
        rule_w1 = Rule(["Weak", "A"], "Rejected")
        rule_w2 = Rule(["Weak", "B"], "Rejected")
        rule_w3 = Rule(["Weak", "C"], "Rejected")
        rule_w4 = Rule(["Weak", "D"], "Accepted")
end

begin
        rule_m1 = Rule(["Moderate", "A"], "Rejected")
        rule_m2 = Rule(["Moderate", "B"], "Rejected")
        rule_m3 = Rule(["Moderate", "C"], "Accepted")
        rule_m4 = Rule(["Moderate", "D"], "Accepted")
end

begin
        rule_g1 = Rule(["Good", "A"], "Rejected")
        rule_g2 = Rule(["Good", "B"], "Accepted")
        rule_g3 = Rule(["Good", "C"], "Accepted")
        rule_g4 = Rule(["Good", "D"], "Accepted")
end

begin
        rule_s1 = Rule(["Strong", "A"], "Accepted")
        rule_s2 = Rule(["Strong", "B"], "Accepted")
        rule_s3 = Rule(["Strong", "C"], "Accepted")
        rule_s4 = Rule(["Strong", "D"], "Accepted")
end

rules = [
        rule_w1, rule_w2, rule_w3, rule_w4,
        rule_m1, rule_m2, rule_m3, rule_m4,
        rule_g1, rule_g2, rule_g3, rule_g4,
        rule_s1, rule_s2, rule_s3, rule_s4
]

md"""
## Fuzzy Inference System
"""

fis = FISMamdani(criteria, decision, rules)

md"Let's make some predictions"
test_in = [9., 5.]
eval_fis(fis, test_in)
```

julia

Task № 3:

Develop a workflow in KNIME that implements the fuzzy logic system for candidate selection described in the previous exercise. The workflow should include the following components:

a) Input nodes for academic record, relevant skills, and motivation.

b) Fuzzy membership function nodes for converting crisp ratings into fuzzy sets.

c) Fuzzy inference nodes for applying the fuzzy rules.

d) Defuzzification nodes for converting fuzzy outputs into crisp candidate rankings.

e) Output nodes for displaying the calculated candidate rankings.

4 Fuzzy Control of an Articulated System

Student's name	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

Goals

- ★ Design a fuzzy logic regulator for controlling the position of a disk in `LabVIEW`;
- ★ Implement and test the fuzzy controller within the `LabVIEW` environment.

The repository is available at `https://github.com/a-mhamdi/disk_and_beam/`

Fuzzy logic is a type of logical system that allows for the representation and manipulation of uncertainty in a formalized way. It is often used in the design of control systems because it is able to handle the imprecision and uncertainty that is often present in real-world systems.

By using fuzzy logic to represent and manipulate uncertainty, you can design control systems that are able to handle imprecision and adapt to changing conditions.

To use fuzzy logic in a control system, you will need to follow these steps:

1. Define the input variables for the system, such as the current temperature or the speed of a motor.
2. Define the output variables for the system, such as the desired temperature or the power supplied to the motor.
3. Define fuzzy sets for each input and output variable. A fuzzy set is a set of values with a degree of membership ranging from 0 (not a member) to 1 (fully a member).
4. Define the fuzzy rules that describe the relationships between the input and output variables. These rules should be written in the form of "if-then" statements, such as "if temperature is hot then power is high".
5. Use the input variables and fuzzy rules to compute the output variables using a fuzzy inference system.
6. Defuzzify the output variables to obtain crisp (numeric) values that can be used to control the system.

Task № 4:

> Identify and describe the physical components of the system, including the disk, the supporting beam, the motor responsible for moving the disk, sensors for measuring position or angle, and the connections to the `LabVIEW` interface. Then, using `LabVIEW`, design and implement a fuzzy logic controller user interface that allows for the adjustment of input parameters, displays sensor measurements, and enables fuzzy-based control of the disk's position along the beam.

5 Exploring the fundamentals of ANN

Student's name	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

Goals

- ★ Introduce the foundational concepts of artificial neural networks (ANNs);
- ★ Illustrate how ANNs can be trained on data and used to generalize beyond their training examples.

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-1* ⟶ *Jupyter* ⟶ *xor-gate.ipynb*

ANNs are a type of neural network that are inspired by the structure and function of the human brain, and they are commonly used for tasks such as image and speech recognition, natural language processing, and prediction.

The XOR (exclusive OR) gate is an useful example to study when learning about artificial neural networks (ANNs) because it is a simple yet non-linear function that cannot be represented by a single perceptron, which is a basic building block of ANNs.

By studying the XOR gate, you learn about the limitations of single perceptron models and how they can be overcome by using multiple perceptrons or other types of ANN architectures. Additionally, the XOR gate is often used as a benchmark test for evaluating the performance of different ANN models, so it is useful to understand how it works and what challenges it presents.

```
using Markdown

md"This code considers the XOR problem. In the terminal, run `julia -e
 ↪'import Pkg; Pkg.activate(\".\"); include(\"Part-1/xor-gate.jl\")'`"

using Flux

md"Create the dataset for an \"XOR\" problem"
X = rand(Float32, 2, 1_024);
# vscodedisplay(X, "X")
y = [xor(col[1]>.5, col[2]>.5) for col in eachcol(X)]
# vscodedisplay(y, "y")

md"Scatter plot of `X`"
using Plots; # unicodeplots()
sc = scatter(X[1,:], X[2,:], group=y; labels=["False" "True"])
loader = Flux.DataLoader((X, y), batchsize=32, shuffle=true)

md"`mdl` is the model to be built"
mdl = Chain(Dense( 2 => 4, tanh ),
            Dense( 4 => 4, tanh ),
            Dense( 4 => 1,   ),
            )

md"Raw output before training"
y_raw = mdl(X)

md"`opt` designates the optimizer"
opt = Adam(.01)

md"`state` contains all trainable parameters"
state = Flux.setup(opt, mdl)

md"## TRAINING PHASE"
vec_loss = []
using ProgressMeter
@showprogress for epoch in 1:1_000
    for (Features, target) in loader
```

```
                # Begin a gradient context session
        loss, grads = Flux.withgradient(mdl) do m
            # Evaluate model:
            target_hat = m(Features) |> vec # loss function expects size(ŷ) = (1, :) to match size(y) = (:,)
                        # Evaluate loss:
            Flux.binarycrossentropy(target_hat, target)
        end
        Flux.update(state, mdl, grads[1])
        push(vec_loss, loss)  # Log `loss` to `losses` vector `vec_loss`
    end
end

md"Predicted output after being trained"
y_hat = mdl(X)
y_pred = (y_hat[1, :] .> .5)

md"Accuracy: How much we got right over all cases _(i.e., (TP+TN)/(TP+TN+FP+FN))_"
accuracy = Flux.Statistics.mean( (y_pred .> .5) .== y )

md"Plot loss vs. iteration"
plot(vec_loss; xaxis=(:log10, "Iteration"), yaxis="Loss", label="Per Batch")
sc1 = scatter(X[1,:], X[2,:], group=y; title="TRUTH", labels=["False" "True"])
sc2 = scatter(X[1,:], X[2,:], zcolor=y_raw[1,:]; title="BEFORE", label=:none, clims=(0,1))
sc3 = scatter(X[1,:], X[2,:], group=y_pred; title="AFTER", labels=["False" "True"])

md"Plot of both ground truth and results after training"
plot(sc1, sc3, layout=(1,2), size=(512,512))
```

Task № 5:

> Using the KNIME Analytics Platform, implement an artificial neural network to solve the XOR logic gate problem. Your implementation should explore three distinct learning rules: (i) Hebbian learning, (ii) the Rosenblatt perceptron rule, and (iii) gradient descent. For each learning rule:

a) Briefly describe the learning rule and how it is applied in the context of neural networks;

b) Build and configure the workflow in KNIME accordingly;

c) Evaluate and compare the performance of each approach on the XOR dataset.

6 Binary Classifier using ANN

Student's name	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

Goals

- ★ Prepare and preprocess a categorical dataset loaded from a CSV file for neural network training.
- ★ Use an artificial neural network to predict and classify categorical outcomes based on the prepared data.

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-1* ⟶ *Jupyter* ⟶ *classifier-ann.ipynb*

A classifier using artificial neural networks (ANNs) is a machine learning model that is trained to identify the class or category of an input based on certain features or characteristics.

In the case of a classifier, an ANN is trained on a labeled dataset, where each input has a corresponding class label. The classifier learns to map the input features to the correct class label by adjusting the weights and biases of the connections between the neurons in the network. Once trained, the classifier can make predictions on new, unseen data by applying the learned mapping to determine

the most likely class label for the input.

```
using Markdown

md"Import the required librairies"
using CSV, DataFrames
using MLJ
using Flux

md"Hyperparameters tuning"
 , epochs, batchsize = .001, 1_00, 64

md"Load data for csv file"
df = CSV.read("../Datasets/Churn_Modelling.csv", DataFrame)

md"Choose the target vector `y`"
ydf = select(df, :Exited)
#coerce!(ydf, :Exited => Multiclass)
y = ydf.Exited

md"Specify the features matrix `X`"
Xdf = select(df, Not([:RowNumber, :CustomerId, :Surname, :Exited]))
coerce!(Xdf,
    :Geography => Multiclass,
    :Gender => Multiclass
)

md"Onehotencoding of multiclass variables"
ce = ContinuousEncoder(drop_last=true)
Xdf = machine(ce, Xdf) |> fit! |> MLJ.transform

md"Features scaling"
sc = Standardizer()
Xdf = machine(sc, Xdf) |> fit! |> MLJ.transform

md"Extract only the values for `X`, i.e, rm the headers."
n, m = size(Xdf)
X = Array{Float32, 2}(undef, (n, m));
for i in 1:m
    X[:, i] = Xdf[!, i];
end

```

```
md"Design the architecture of the classifier, denoted hereafter by `clf`
 ↪"
clf = Chain(
            Dense( 11 => 8, relu ),
            Dense(  8 => 8, relu ),
            Dense(  8 => 8, relu ),
            Dense(  8 => 1 )
            )

md"Permute dims: ROW => features and COL => observation"
X = X'; # permutedims(X)
y = y'; # permutedims(y)

md"Optimizers and data loader"
opt = Flux.Adam( );
state = Flux.setup(opt, clf);
loader = Flux.DataLoader((X, y); batchsize=batchsize, shuffle=true);
vec_loss = []

md"**Training phase**"
using ProgressMeter
@showprogress for _ in 1:epochs
    for (X, y) in loader
        loss, grads = Flux.withgradient(clf) do mdl
            ŷ_ = mdl(X);
            Flux.Losses.logitbinarycrossentropy(ŷ_, y);
        end
        Flux.update!(state, clf, grads[1]); # Upd `W` and `b`
        push!(vec_loss, loss); # Log `loss` to the vector `vec_loss`
    end
end

md"Plot the loss vector `vec_loss`"
using Plots
plot(vec_loss, label="Loss")
extrema(vec_loss)

md"Some metrics"
ŷ = clf(X) |>  ;
ŷ = (ŷ .  .5);

md"Basic way to compute the accuracy"
accuracy = mean( ŷ .== y )

```

```
md"Confusion Matrix"
displayed_cm = MLJ.ConfusionMatrix(levels=[1, 0])(ŷ, y)
cm = ConfusionMatrices.matrix(displayed_cm)

md"Other metrics"
TP, TN, FP, FN = cm[2, 2], cm[1, 1], cm[2, 1], cm[1, 2];
accuracy_ = (TP+TN)/(TP+TN+FP+FN) # MLJ.accuracy(cm)
precision_ = TP/(TP+FP) # MLJ.precision(cm)
recall_ = TP/(TP+FN) # MLJ.recall(cm)
f1score_ = 2/(1/precision_ + 1/recall_) # MLJ.f1score(cm)
```

julia

Task № 6:

> Using the KNIME Analytics Platform and the provided customer churn dataset, design and implement an artificial neural network (ANN) to predict the probability that a customer will churn. Your analysis should include:
>
> **a)** Exploring and preprocessing the churn dataset for use in a neural network;
>
> **b)** Building and configuring the ANN workflow in KNIME for classification;
>
> **c)** Evaluating model performance and interpreting the results to understand churn behavior.

7 Project Assessment

The final project will provide you the chance to go into greater detail about a topic that was covered in class and that you are interested in learning more about. The main objective is to provide you a difficult yet doable test that lets you show off your understanding of fuzzy logic and neural networks.

Some possible topics for fuzzy logic could include fuzzy set theory, fuzzy rule-base systems, and applications of fuzzy logic in control systems. For neural networks, possible topics could include feedforward neural networks, convolutional neural networks, and applications of neural networks in image classification and natural language processing.

You must include all required materials, including relevant datasets, sample code, and a set of slides on which to display your work. By revisiting the fundamentals of fuzzy logic or neural networks and being acquainted with pertinent programming languages and libraries, you should be able to show that you comprehend the information presented in this course. The final project is comprised of:

1. proposal;
2. report documenting your work, results and conclusions;
3. presentation;
4. source code *(You should share your project on* **GitHub***.)*

It is about two pages long. It includes:

- Title
- Datasets *(If needed!)*
- Idea
- Software *(Not limited to what you have seen in class)*
- Related papers *(Include at least one relevant paper)*
- Teammate *(Teams of three to four students. You should highlight each partner's contribution)*

It is about ten pages long. It revolves around the following key takeaways:

- Context *(Input(s) and output(s))*
- Motivation *(Why?)*
- Previous work *(Literature review)*
- Flowchart of code, results and analysis
- Contribution parts *(Who did what?)*

Typesetting using LaTeX is a bonus. You can use **LyX** (https://www.lyx.org/) editor. A template is available at https://github.com/a-mhamdi/ailab-isetbz/tree/main/LyX. Here what your report might contain:

1. Provide a summary which gives a brief overview of the main points and conclusions of the report.
2. Use headings and subheadings to organize the main points and the relationships between the different sections.
3. Provide an outline or a list of topics that the report will cover. Including a table of contents can help to quickly and easily find specific sections of your report.
4. Use visuals: Including visual elements such as graphs, charts, and tables can help to communicate the content of a report more effectively. Visuals can help to convey complex information in a more accessible and intuitive way.

If you are using `Julia`, you can generate the documentation using the package **Documenter.jl**. It is a great way to create professional-looking material. It allows to easily write and organize documentation using a variety of markup languages, including **Markdown** and LaTeX, and provides a number of features to help create a polished and user-friendly documentation website.

I will assess your work based on the quality of your code and slides, as well as your ability to effectively explain and demonstrate your understanding of the topic. I will also consider the creativity and originality of your projects, and your ability to apply what you have learned to real-world situations. I also make myself available to answer any questions or provide feedback as you work on your projects.

The overall scope of this manual is to introduce **Artificial Intelligence (AI)** , through either some numerical simulations or hands-on training, to the students at **ISET Bizerte**.

The topics discussed in this manuscript are as follow:

① Getting started with *Julia*

Get familiar with *Pluto* Notebook.

② Fuzzification, inference system & defuzzification

Membership functions; COG.

③ System control using fuzzy logic

④ How to build an ANN

Julia; REPL; *Jupyter Lab*; *Pluto*; *Fuzzy*; *Flux*; *Matlab*; *KNIME*; *LabVIEW*; *ANN*.