TERM: M1-EE

SEMESTER: 1

AY: 2024-2025

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

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[1];

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*

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`

[1]If you prefare using Windows, a similar environment has been setup for you by pressing ⊞ & R . This will open the dialog box **Run**. In the command line, type **cmd**, and then use the ⏎ key to confirm. Next, type `jlai1` and press ⏎ once more.

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*

- Enter commands in `Julia` REPL to create variables and perform calculations;
- Write and save programs;
- Use indexing to extract and modify rows, columns, and elements of `Julia` tensors.

`Julia` is a standalone program which can be downloaded from `https://julialang.org/downloads/`

Getting around

By default, Julia runs in an interactive terminal called the REPL. In this mode, Some useful commands are:

1. `^C` aborts execution
2. `^D` exits Julia
3. `?` enters help mode
4. `;` enters system shell mode
5. `]` enters package manager mode
6. `^l` clears screen

We begin first by activating the environement within the desired folder.

From the REPL interface, either type

```
using Pkg
pkg"activate ."
```

or access the package mode by typing `]` and simply write

```
activate .
```

Always within the package mode, to see the full list of installed packages

```
st
```

[ ]:
```
using Pkg
pkg"activate ."
```

[ ]:
```
]st
```

To add the **Markdown** package for instance, we write

[ ]:
```
]add Markdown
```

To be able to use it, we do as follows

[4]:
```
using Markdown
```

[5]:
```
md"""
This a text inside a code cell, thanks to **Markdown** package. I can _emphasize_
anything. Make other things **bold**
"""
```

[5]: This a text inside a code cell, thanks to **Markdown** package. I can *emphasize* anything. Make other things **bold**

Runnig `Julia` in Jupyer Notebookor Jupyter Lab is pretty handy. We only need to install the appropriate kernel. In order to add `Julia` kernel `IJulia` to Jupyter Notebook and/or `JupyterLab` IDEs, we begin by executing the following commands:

```
using Pkg
Pkg.add("IJulia")
```

If we want to get `JupyterLab` instance running in current directory, we can do:

```
jupyterlab(dir=pwd(), detached=true)
```

In case things do not work, we run the two following commands from `Julia` REPL which launch jupyter environment.

```
using IJulia
installkernel("Julia")
```

The shell mode is also available through the REPL to evaluate some os commands. To do so, simply preface the regular command by semicolon. For instance, `pwd` prints the path to working directory and `ls` allows to list the content of the current directory.

```
[6]: ;pwd
```

```
/home/mhamdi/Work/git-repos/AI-ML-DL/jlai/Codes/Julia/Part-1
```

```
[ ]: ;ls -la
```

Getting Help

In order to seek help on a particular function. We just use the `?` mark. We can use the `Julia` documentation to discover more pieces of information about `Julia` features.

```
[8]: ?cos
```

```
search: cos
cosd
acos
cosc
cosh cis
cot acosh
close
cospi
acosd
const coth
copy
```

[8]:
```
cos(x)
```

Compute cosine of `x`, where `x` is in radians.

See also `cosd`, `cospi`, `sincos`, `cis`.

```
cos(A::AbstractMatrix)
```

Compute the matrix cosine of a square matrix `A`.

If `A` is symmetric or Hermitian, its eigendecomposition (`eigen`) is used to compute the cosine. Otherwise, the cosine is determined by calling `exp`.

1.1 Examples

```
julia> cos(fill(1.0, (2,2)))
2×2 Matrix{Float64}:
  0.291927  -0.708073
 -0.708073   0.291927
```

To print something on the standard output, it is possible to use either `print` and `println`. The last one displays the text and moves the cursor to the next line.

[9]:
```
print("Hello")
print(' ')
print("World")
```

```
Hello World
```

[10]:
```
println("Hello")
println("World")
```

```
Hello
World
```

Data types: Dictionaries

[11]:
```
dict = Dict(
    # Name => # of wheels
    "Unicycle" => 1,
    "Bicycle" => 2,
    "Tricycle" => 3
)
```

[11]:
```
Dict{String, Int64} with 3 entries:
  "Bicycle"  => 2
  "Tricycle" => 3
  "Unicycle" => 1
```

[12]:
```
typeof(dict)
```

[12]:
```
Dict{String, Int64}
```

[13]:
```
dict = Dict([("Unicycle", 1), ("Bicycle", 2), ("Tricycle", 3)])
```

[13]:
```
Dict{String, Int64} with 3 entries:
  "Bicycle"  => 2
  "Tricycle" => 3
  "Unicycle" => 1
```

[14]:
```
dict["Bicycle"]
```

[14]:
```
2
```

[15]:
```
lst = [1, 'a', "abc", true, [0, .5im]]
```

[15]:
```
5-element Vector{Any}:
    1
     'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
     "abc"
 true
     ComplexF64[0.0 + 0.0im, 0.0 + 0.5im]
```

[16]:
```
typeof(lst)
```

[16]:
```
Vector{Any} (alias for Array{Any, 1})
```

[17]:
```
lst[end]
```

[17]:
```
2-element Vector{ComplexF64}:
 0.0 + 0.0im
 0.0 + 0.5im
```

Basic Calculations

[18]:
```
a, b = 1, 1.5
```

[18]:
```
(1, 1.5)
```

[19]:
```
println(typeof(a))
println(typeof(b))
```

```
Int64
Float64
```

[20]:
```
md"""
`varinfo` method allows to display loaded variables.
"""
```

[20]: varinfo method allows to display loaded variables.

[21]:
```
varinfo()
```

[21]:

name	size	summary
Base		Module
Core		Module
Main		Module
PLOTS_DEFAULTS	384 bytes	Dict{Symbol, Symbol} with 1 entry
a	8 bytes	Int64
b	8 bytes	Float64
dict	431 bytes	Dict{String, Int64} with 3 entries
docs	6.466 KiB	Markdown.MD
lst	176 bytes	5-element Vector{Any}
showall	0 bytes	showall (generic function with 1 method)

[22]:
```
?varinfo
```

search: **varinfo** @info versioninfo

[22]:
```
varinfo(m::Module=Main, pattern::Regex=r""; all=false, imported=false, recursive=false, sortby::Symbol=:na
```

Return a markdown table giving information about public global variables in a module, optionally restricted to those matching pattern.

The memory consumption estimate is an approximate lower bound on the size of the internal structure of the object.

- all : also list non-public objects defined in the module, deprecated objects, and compiler-generated objects.
- imported : also list objects explicitly imported from other modules.

- recursive : recursively include objects in sub-modules, observing the same settings in each.
- sortby : the column to sort results by. Options are :name (default), :size, and :summary.
- minsize : only includes objects with size at least minsize bytes. Defaults to 0.

The output of varinfo is intended for display purposes only. See also names to get an array of symbols defined in a module, which is suitable for more general manipulations.

[23]:
```
println("Sum of $a and $b is $(a+b)")
```

```
Sum of 1 and 1.5 is 2.5
```

Addition, subtraction, multiplication, division, exponent

[24]:
```
a+b, a-b, a*b, a÷b, a^b
```

[24]:
```
(2.5, -0.5, 1.5, 0.0, 1.0)
```

[25]:
```
md"""
**Unicode support**
1. We can use π instead of `pi`
1. Greek letters improe comprehension: α _(alpha)_, β _(beta)_, ...
1. Symbols: ≥ _(>=)_, ≤ _(<=)_, ∈ _(in)_, ...
1. ...
"""
```

[25]: **Unicode support**

1. We can use π instead of pi
2. Greek letters improe comprehension: α *(alpha)*, β *(beta)*, ...
3. Symbols: ≥ *(>=)*, ≤ *(<=)*, ∈ *(in)*, ...
4. ...

[26]:
```
3 ≤ π
```

[26]:
```
true
```

[27]:
```
typeof(3.14)
```

[27]:
```
Float64
```

[28]:
```
Float64 |> supertype |> supertype |> supertype |> supertype
```

```
[28]: Any
```

```
[29]: Integer |> subtypes
```

```
[29]: 3-element Vector{Any}:
       Bool
       Signed
       Unsigned
```

```
[30]: Signed |> subtypes
```

```
[30]: 6-element Vector{Any}:
       BigInt
       Int128
       Int16
       Int32
       Int64
       Int8
```

```
[31]: UInt8 <: Unsigned # UInt8 is one subtype of Unsigned
```

```
[31]: true
```

```
[32]: Signed >: Int8 # Signed is supertype of Int8
```

```
[32]: true
```

```
[33]: typeof(3)
```

```
[33]: Int64
```

```
[34]: tmp::UInt8 = 3
      typeof(tmp)
```

```
[34]: UInt8
```

Mathematical Notation

```
[35]: println(1+2)
      println(+(1, 2))
```

```
3
3
```

[36]:
```julia
println(1-2)
println(-(1, 2))
```

```
-1
-1
```

[37]:
```julia
println(1*2)
println(*(1, 2))
```

```
2
2
```

[38]:
```julia
println(1/2)
println(/(1, 2))
```

```
0.5
0.5
```

[39]:
```julia
println(3/4+7/5)
println(3//4+7//5)
```

```
2.15
43//20
```

Array Transformations

Perform calculations on entire arrays at once.

[40]:
```julia
zeros(3, 2)
```

[40]:
```
3×2 Matrix{Float64}:
 0.0  0.0
 0.0  0.0
 0.0  0.0
```

[41]:
```julia
ones(3, 3, 2)
```

[41]:
```
3×3×2 Array{Float64, 3}:
[:, :, 1] =
 1.0  1.0  1.0
 1.0  1.0  1.0
 1.0  1.0  1.0

[:, :, 2] =
 1.0  1.0  1.0
 1.0  1.0  1.0
 1.0  1.0  1.0
```

```
[42]: fill(π, (2, 2))
```

```
[42]: 2×2 Matrix{Irrational{:π}}:
       π  π
       π  π
```

```
[43]: π .* ones(2, 2)
```

```
[43]: 2×2 Matrix{Float64}:
       3.14159  3.14159
       3.14159  3.14159
```

```
[44]: md"Creates a `BitArray` with all values set to `true`"
```

[44]: Creates a BitArray with all values set to true

```
[45]: var = trues(2, 4)
      println(var)
      typeof(var)
```

```
Bool[1 1 1 1; 1 1 1 1]
```

```
[45]: BitMatrix (alias for BitArray{2})
```

```
[46]: md"Creates a `BitArray` with all values set to `false`"
```

[46]: Creates a BitArray with all values set to false

```
[47]: var = falses(2, 4)
      println(var)
      typeof(var)
```

```
Bool[0 0 0 0; 0 0 0 0]
```

```
[47]: BitMatrix (alias for BitArray{2})
```

```
[48]: md"**Comprehension**"
```

[48]: **Comprehension**

```
[49]: str = "Hello Julia"
      [println(el) for el in str];
```

```
H
e
l
l
o

J
u
l
i
a
```

Calling Functions

Call functions to obtain multiple outputs.

[50]:
```
md"""
[Functions in ${\tt Julia}$](https://docs.julialang.org/en/v1/manual/functions/)
"""
```

[50]: Functions in Julia

[51]:
```
md"**Spreading Arguments**"
```

[51]: **Spreading Arguments**

Optional positional arguments

[52]:
```
foo(x=0, y=0, z=0) = x+y+z
```

[52]:
```
foo (generic function with 4 methods)
```

[53]:
```
foo(), foo(1, 2, 3)
```

[53]:
```
(0, 6)
```

[54]:
```
foo([1, 2, 3]...) # Splat `...` operator
```

[54]:
```
6
```

Keywords arguments

[55]:
```
bar(; a::Real=0, b::Real=0, c::Real=0) = a+b+c
```

[55]:
```
bar (generic function with 1 method)
```

[56]:
```
bar()
```

[56]:
```
0
```

[57]:
```
bar(; Dict(:a => 3, :b => 5.4, :c => -1.2)... ) # ; kwargs...
```

[57]:
```
7.2
```

[58]:
```
# THROW AN ERROR
try bar([1, 2, 3]...)
catch error
    println(error)
end
```

```
MethodError(Main.bar, (1, 2, 3), 0x000000000000683c)
```

[59]:
```
md"**Multiple Dispatch**"
```

[59]: **Multiple Dispatch**

[60]:
```
# 1st method signature
function f(x::Int)
    x^2
end
```

[60]:
```
f (generic function with 1 method)
```

[61]:
```
# 2nd method signature
f(x::Float64) = x^2+1
```

[61]:
```
f (generic function with 2 methods)
```

[62]:
```
# 3rd method signature
f(x::Char) = x*'y'*'z'
# 4th mehod signature
f(x::String) = x*x
```

[62]:
```
f (generic function with 4 methods)
```

[63]:
```
methods(f)
```

[63]:
```
# 4 methods for generic function "f" from Main:
 [1] f(x::String)
     @ In[62]:4
 [2] f(x::Char)
```

```
      @ In[62]:2
 [3] f(x::Float64)
      @ In[61]:2
 [4] f(x::Int64)
      @ In[60]:2
```

```
[64]: f(1), f(1.), f('x'), f("abc")
```

```
[64]: (1, 2.0, "xyz", "abcabc")
```

```
[65]: mycos(x) = cos(x)
      mycos(adj, hyp) = adj/hyp # Extension to `mycos` function
```

```
[65]: mycos (generic function with 2 methods)
```

```
[66]: methods(mycos)
```

```
[66]: # 2 methods for generic function "mycos" from Main:
       [1] mycos(adj, hyp)
            @ In[65]:2
       [2] mycos(x)
            @ In[65]:1
```

```
[67]: @which mycos(π)
```

```
[67]: mycos(x)
           @ Main In[65]:1
```

```
[68]: @which mycos(5, 3)
```

```
[68]: mycos(adj, hyp)
           @ Main In[65]:2
```

```
[69]: mycos(adj, hyp=10) = adj/hyp
```

```
[69]: mycos (generic function with 2 methods)
```

```
[70]: @which mycos(π)
```

```
[70]: mycos(adj)
           @ Main In[69]:1
```

Function Chaining applies a function to the preceding argument.

[71]:
```
g(x) = x+1
h(x) = x^2
x = 2 |> g |> h
```

[71]:
```
9
```

[72]:
```
md"Another pssible way is t use ` `_\circ{tab}_ symbol"
```

[72]: Another pssible way is t use \circ{tab} symbol

[73]:
```
(h g)(2)
```

[73]:
```
9
```

[74]:
```
md"Definition of a function can be done on the fly"
```

[74]: Definition of a function can be done on the fly

[75]:
```
y = 5 |> (x->x^2) |> √
```

[75]:
```
5.0
```

[76]:
```
md"""
**Metaprogramming:** Code is optimized by nature in ${\tt Julia}$
"""
```

[7]: **Metaprogramming:** Code is optimized by nature in Julia

[77]:
```
function Foo(x::Integer)
    y = x
    for i=1:100
        y += i^2
    end
    return y
end
```

[77]:
```
Foo (generic function with 1 method)
```

[78]:
```
@code_llvm Foo(3)
```

```
; Function Signature: Foo(Int64)
;  @ In[77]:1 within `Foo`
define i64
@julia_Foo_19508(i64 signext
%"x::Int64") #0 {
```

```
top:
  %0 = add i64 %"x::Int64",
338350
;  @ In[77]:6 within `Foo`
  ret i64 %0
}
```

[79]:

```
?@code_llvm
```

[79]: @code_llvm

Evaluates the arguments to the function or macro call, determines their types, and calls code_llvm on the resulting expression. Set the optional keyword arguments raw, dump_module, debuginfo, optimize by putting them and their value before the function call, like this:

```
@code_llvm raw=true dump_module=true debuginfo=:default f(x)
@code_llvm optimize=false f(x)
```

optimize controls whether additional optimizations, such as inlining, are also applied. raw makes all metadata and dbg.* calls visible. debuginfo may be one of :source (default) or :none, to specify the verbosity of code comments. dump_module prints the entire module that encapsulates the function.

See also: code_llvm, @code_warntype, @code_typed, @code_lowered, @code_native.

Plotting Data

Visualize variables using Julia's plotting functions.

[ ]:

```
]add Plots
```

[81]:

```
using Plots # GR is the default backend
```

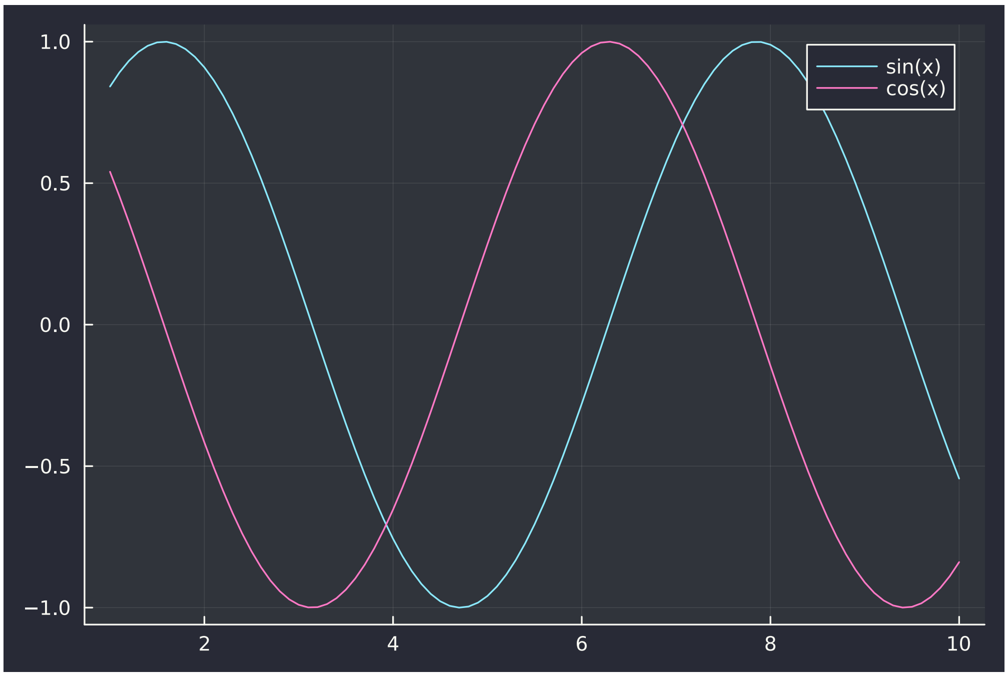
```
WARNING: using Plots.bar in module Main conflicts with an existing identifier.
```

[82]:

```
x = 1:.1:10
y = sin.(x)
z = cos.(x)

plot(x, y, label="sin(x)")
plot!(x, z, label="cos(x)") # Hold on the previous plot
```

[82]:

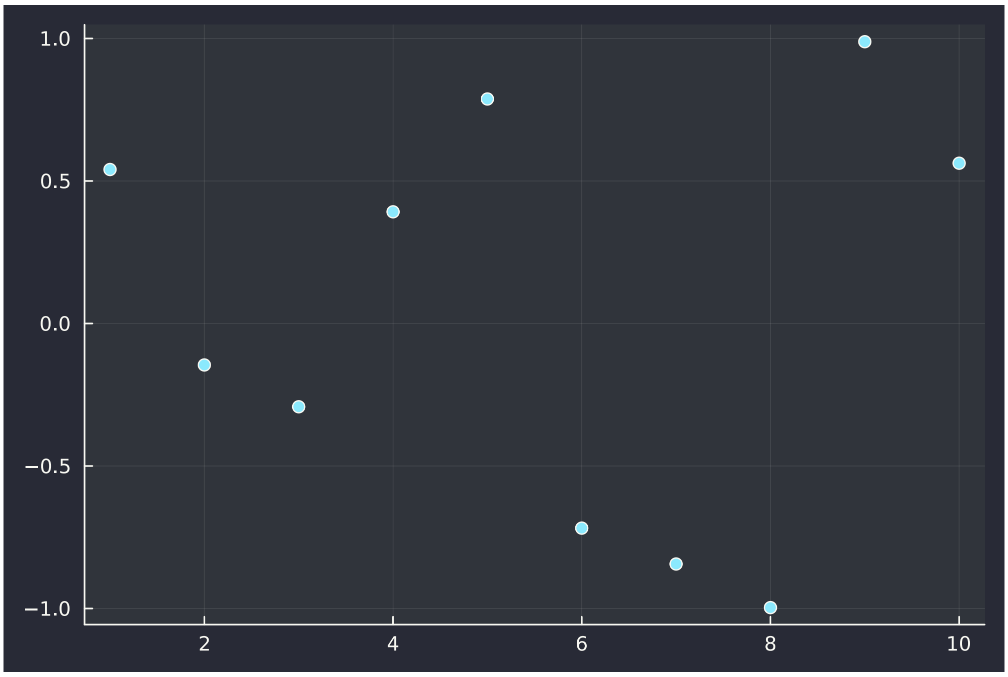


[83]:
```
md"**Scatter Plot**"
```

[83]: **Scatter Plot**

[84]:
```
x = range(1, 10)
y = cos.(x.^3)
scatter(x, y, legend=false)
```

[84]:



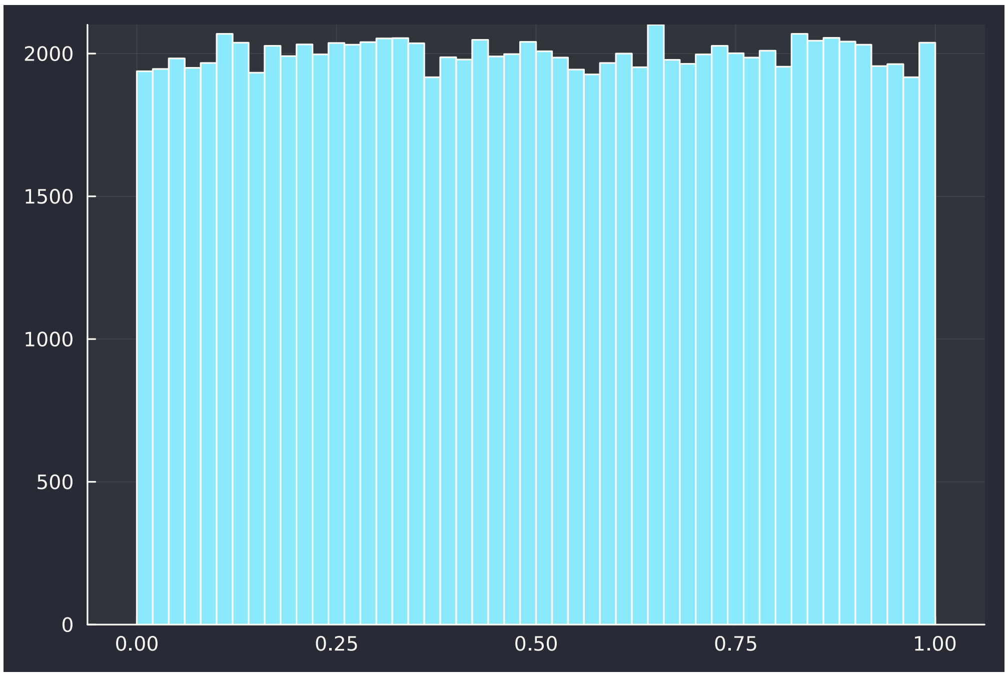
```
[85]: md"**Uniform Distribtion**"
```

[85]: **Uniform Distribtion**

```
[86]: ?rand;
```

```
[87]: x = rand(10^5)
      histogram(x, bins=64, legend=false)
```

[87]:



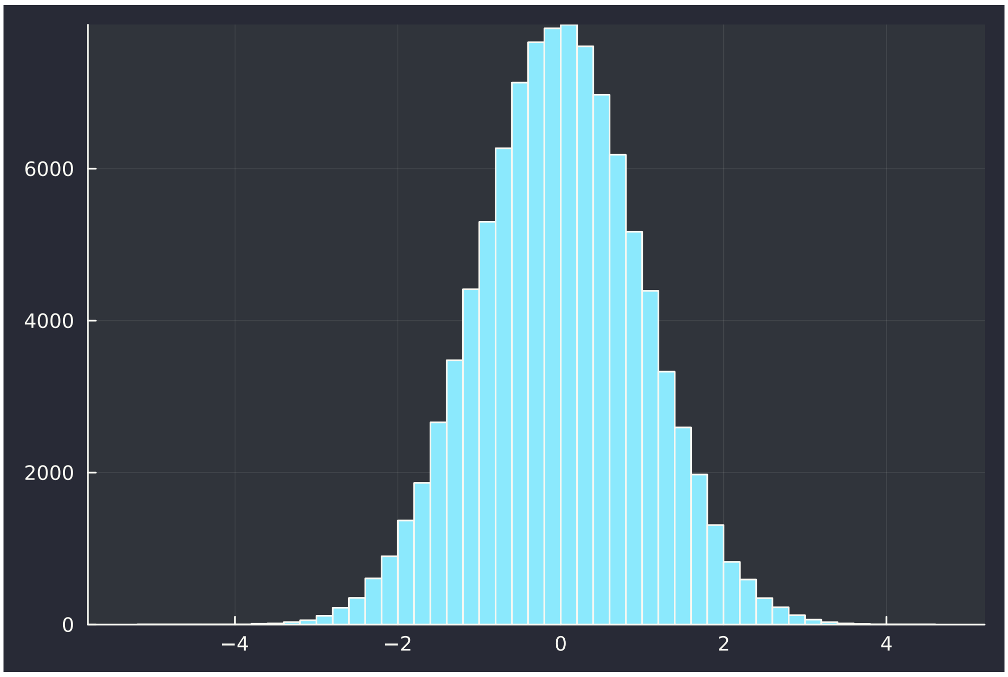
[88]:
```
md"**Normal Distribution**"
```

[88]: **Normal Distribution**

[89]:
```
?randn;
```

[90]:
```
x = randn(10^5)
histogram(x, bins=64, legend=false)
```

[90]:

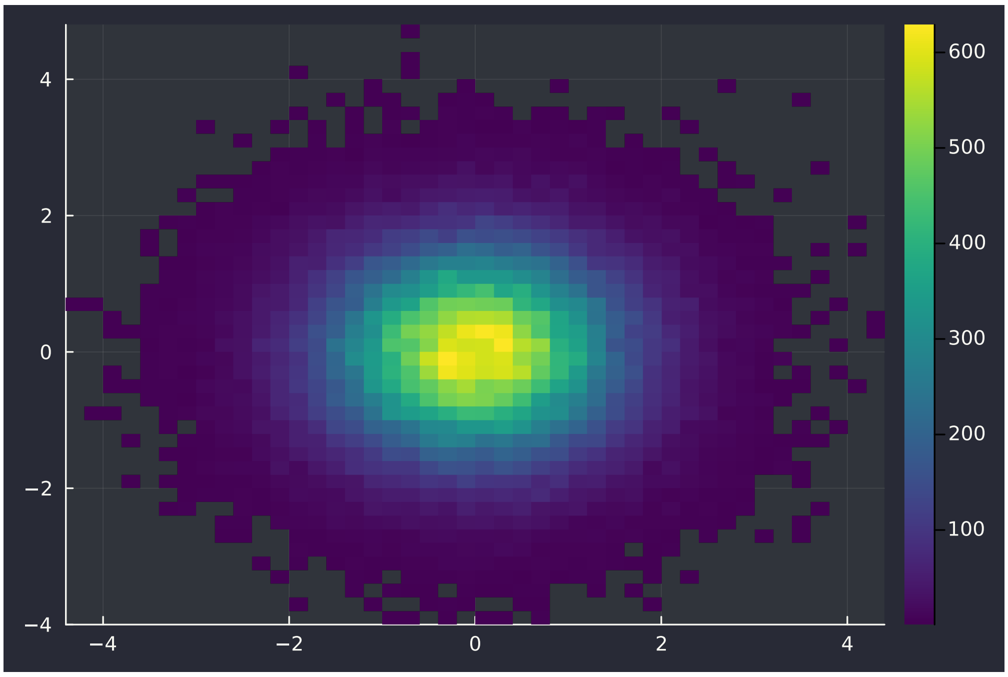


[91]:
```
md"**Histogram in 2D**"
```

[91]: **Histogram in 2D**

[92]:
```
x = randn(10^5)
y = randn(10^5)
histogram2d(x, y, bins=(64, 64))
```

[92]:



Importing Data

Bring data from external files into Julia.

Data is typically stored in files, such as *CSV* or *JSON* files. In order to train and test machine learning models, the data needs to be loaded into the program. Additionally, the results of the training and testing process, such as model weights and performance metrics, also need to be saved to files. Therefore, the ability to manipulate files is essential for loading and saving data and model information in the machine learning process.

[93]:
```julia
using Pkg
Pkg.add("DataFrames")
Pkg.add("CSV")
```

```
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions…
  No Changes to `~/Work/git-repos/AI-ML-
DL/jlai/Codes/Julia/Part-1/Project.toml`
  No Changes to `~/Work/git-repos/AI-ML-
DL/jlai/Codes/Julia/Part-1/Manifest.toml`
   Resolving package versions…
  No Changes to `~/Work/git-repos/AI-ML-
```

```
DL/jlai/Codes/Julia/Part-1/Project.toml`
  No Changes to `~/Work/git-repos/AI-ML-
DL/jlai/Codes/Julia/Part-1/Manifest.toml`
```

[94]:
```
md"Create new CSV file"
```

[94]: Create new CSV file

[95]:
```
using CSV, DataFrames
```

[96]:
```
md"`touch` command allows to create a file if it doesn't exist. Otherwise, it changes
the file timestamps."
```

[96]: touch command allows to create a file if it doesn't exist. Otherwise, it changes the file timestamps.

[97]:
```
touch("test-file.csv")
```

[97]:
```
"test-file.csv"
```

[98]:
```
;ls -la test-file.csv
```

```
-rw-rw-r-- 1 mhamdi mhamdi 66 Nov  7 18:47 test-file.csv
```

[99]:
```
file = open("test-file.csv", "w")
```

[99]:
```
IOStream(<file test-file.csv>)
```

[100]:
```
md"Let's create some imaginary data"
```

[100]: Let's create some imaginary data

[101]:
```
df = DataFrame(
        Student = ["Mohamed", "Aymen", "Rami", "Ala"],
        Id = [1, 2, 3, 4],
        Marks = [18, 7, 12, 5.5]
        )
```

[101]:

	Student	Id	Marks
	String	Int64	Float64
1	Mohamed	1	18.0
2	Aymen	2	7.0
3	Rami	3	12.0
4	Ala	4	5.5

[102]:
```
md"Write `df` to file"
```

[102]: Write df to file

[103]:
```
CSV.write("test-file.csv", df)
```

[103]:
```
"test-file.csv"
```

[104]:
```
md"Open the CSV file and add some contents. See what happens when we load it again."
```

[104]: Open the CSV file and add some contents. See what happens when we load it again.

[105]:
```
CSV.read("test-file.csv", DataFrame)
```

[105]:

	Student	Id	Marks
	String7	Int64	Float64
1	Mohamed	1	18.0
2	Aymen	2	7.0
3	Rami	3	12.0
4	Ala	4	5.5

Logical Arrays

Use logical expressions to help extracting elements of interest from Julia arrays.

[106]:
```
x = [1, 2, -5, 7.2, 3im]
println(x)
typeof(x)
```

```
ComplexF64[1.0 + 0.0im, 2.0 + 0.0im, -5.0 + 0.0im, 7.2 + 0.0im, 0.0 + 3.0im]
```

[106]:
```
Vector{ComplexF64} (alias for Array{Complex{Float64},
1})
```

[107]:
```
idx = [false, true, false, false, true]
print(x[idx])
```

```
ComplexF64[2.0 + 0.0im, 0.0 + 3.0im]
```

[108]:
```
M = Array{Float64, 2}(undef, 5, 4)
```

[108]:
```
5×4 Matrix{Float64}:
 6.11853e-310  6.11857e-310  6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310  6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310  6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310  6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310  6.11857e-310  6.11857e-310
```

[109]:
```
row_idx = [true, false, true, true, false];
col_idx = [false, true, true, false];
```

[110]:
```
M[row_idx, :]
```

[110]:
```
3×4 Matrix{Float64}:
 6.11853e-310  6.11857e-310  6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310  6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310  6.11857e-310  6.11857e-310
```

[111]:
```
M[:, col_idx]
```

[111]:
```
5×2 Matrix{Float64}:
 6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310
```

[112]:
```
M[row_idx, col_idx]
```

[112]:
```
3×2 Matrix{Float64}:
 6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310
 6.11857e-310  6.11857e-310
```

Programming

Write programs that execute code based on some condition.

[113]:
```
md"**Conditional Evaluation**"
```

[113]: **Conditional Evaluation**

[114]:
```
a, b = π, π
if a < b
    println("$a is less than $b")
elseif a > b
    println("$a is greater than $b")
else
    println("$a is equal to $b")
end
```

```
π is equal to π
```

[115]:
```
md"**`While` Loop**"
```

[115]: **While Loop**

[116]:
```
fruits = ["Blueberry", "Orange", "Banana", "Raspberry", "Strawberry"]
iter = 1
while iter ≤ length(fruits)
    println("Item #$iter is $(fruits[iter])")
    iter +=1
end
```

```
Item #1 is Blueberry
Item #2 is Orange
Item #3 is Banana
Item #4 is Raspberry
Item #5 is Strawberry
```

[117]:
```
md"**`For` Loop**"
```

[117]: **For Loop**

[118]:
```
vegetables = ["Broccoli", "Garlic", "Mushrooms", "Potatoes"]
i = 1
for item in vegetables
    println("Item #$i is $item")
    i += 1
end
```

```
Item #1 is Broccoli
Item #2 is Garlic
Item #3 is Mushrooms
Item #4 is Potatoes
```

Final Project

Bring together concepts that you have learned with a project.

This simple project consists of implementing a basic calculator. This latter could have the ability to perform basic arithmetic operations like *addition*, *subtraction*, *multiplication*, and *division*.

Here are the steps to be followed:

1. Create a function called `calculator()` that takes two arguments, `x` and `y`, and a char operation that specifies which operation to perform.
2. Use an `if-else` statement to check the value of operation. Depending on the value of operation, call the appropriate function to perform the calculation.
3. Test the calculator function by calling it with different values for `x`, `y`, and operation and printing the result.
4. Once the basic calculator is working, we can improve it by adding more functionality such as handling decimals and negative numbers, or implementing more advanced operations such as square root, power, trigonometry and so on.
5. Finally, we could also experiment with different input types, such as command line arguments or a graphical user interface.

[119]:
```
md"Here is an example of how the basic calculator function could look like:"
```

[119]:
```
Here is an example of how the basic calculator function could look like:
```

[120]:
```julia
function calculator(x::Number, y::Number, op::Char)
    if op == '+'
        return x + y
    elseif op == '-'
        return x - y
    elseif op == '*'
        return x * y
    elseif op in ['/', '÷']
        return x / y
    else
        return "INVALID OPERATION"
    end
end
```

[120]:
```
calculator (generic function with 1 method)
```

[121]:
```julia
println("Summation is $(calculator(5, 3, '+'))")
println("Subtraction is $(calculator(5, 3, '-'))")
println("Multiplication is $(calculator(5, 3, '*'))")
println("Division is $(calculator(5, 3, '÷'))")
println(calculator(5, 3, '×'))
```

```
Summation is 8
Subtraction is 2
Multiplication is 15
Division is 1.6666666666666667
INVALID OPERATION
```

Miscallenous

[122]:
```julia
md"Check your version of Julia"
versioninfo()
```

```
Julia Version 1.11.1
Commit 8f5b7ca12ad (2024-10-16 10:53 UTC)
Build Info:
  Official https://julialang.org/ release
Platform Info:
  OS: Linux (x86_64-linux-gnu)
  CPU: 8 × Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
  WORD_SIZE: 64
  LLVM: libLLVM-16.0.6 (ORCJIT, skylake)
Threads: 1 default, 0 interactive, 1 GC (on 8 virtual cores)
Environment:
  LD_LIBRARY_PATH = /home/mhamdi/torch/install/lib:
  DYLD_LIBRARY_PATH = /home/mhamdi/torch/install/lib:
  JULIA_NUM_THREADS = 8
```

[123]:
```julia
md"The macro `@edit` shows the defintion of a function when invoked with specific
arguments"
# @edit maximum([-1, 0, 1])
```

[123]: The macro @edit shows the defintion of a function when invoked with specific arguments

[124]:
```julia
md"`varinfo` lists all global variables and their corresponding types in the current
scope"
varinfo()
```

[124]:

name	size	summary
Base		Module
Core		Module
Foo	0 bytes	Foo (generic function with 1 method)
M	208 bytes	5×4 Matrix{Float64}
Main		Module
PLOTS_DEFAULTS	384 bytes	Dict{Symbol, Symbol} with 1 entry
a	0 bytes	Irrational{:π}
b	0 bytes	Irrational{:π}
bar	0 bytes	bar (generic function with 1 method)
calculator	0 bytes	calculator (generic function with 1 method)
col_idx	44 bytes	4-element Vector{Bool}
df	827 bytes	4×3 DataFrame
dict	431 bytes	Dict{String, Int64} with 3 entries
docs	7.438 KiB	Markdown.MD
f	0 bytes	f (generic function with 4 methods)
file	380 bytes	IOStream
foo	0 bytes	foo (generic function with 4 methods)
fruits	160 bytes	5-element Vector{String}
g	0 bytes	g (generic function with 1 method)
h	0 bytes	h (generic function with 1 method)
i	8 bytes	Int64
idx	45 bytes	5-element Vector{Bool}
iter	8 bytes	Int64
lst	176 bytes	5-element Vector{Any}
mycos	0 bytes	mycos (generic function with 2 methods)
row_idx	45 bytes	5-element Vector{Bool}
showall	0 bytes	showall (generic function with 1 method)
str	19 bytes	11-codeunit String
tmp	1 byte	UInt8
var	80 bytes	2×4 BitMatrix
vegetables	159 bytes	5-element Vector{String}
x	120 bytes	5-element Vector{ComplexF64}
y	781.289 KiB	100000-element Vector{Float64}
z	768 bytes	91-element Vector{Float64}

Modules

[125]:
```
module MyModule
export a
a = 0
b = true
end
```

[125]:
```
Main.MyModule
```

[126]:
```
varinfo(MyModule)
```

[126]:

name	size	summary
MyModule	2.234 KiB	Module
a	8 bytes	Int64

[127]:
```
a
```

[127]:
```
π = 3.1415926535897...
```

[128]:
```
MyModule.a
```

[128]:
```
0
```

[129]:
```
MyModule.b
```

[129]:
```
true
```

[130]:
```
using .MyModule
```

```
WARNING: using MyModule.a in module Main conflicts with an existing identifier.
```

[131]:
```
a, b
```

[131]:
```
(π, π)
```

[132]:
```
using .MyModule: b
```

```
WARNING: import of MyModule.b into Main conflicts with an existing identifier;
ignored.
```

[133]:
```
b
```

[133]:
```
π = 3.1415926535897...
```

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

★ Build algorithms to help decide in given ambiguous situation.

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

Matlab is an interesting tool for engineers. All you need in *Matlab* is the basics pieces of information needed to solve the problem. There is no need to configure your environment to adopt your algorithm, just know a little bit about the syntax. It is an interactive matrix calculator, with a full-fledged programming environment. *Matlab* allows to handle with general tasks and very delicate purpose. The main focus with is what you need to program not how to program it.

Matlab is an easy to use environment, it is a fourth-generation programming language[1] (4GL). It is a very high level language.

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

Matlab desktop

Desktop layout

Like any other environment, *Matlab* has a menubar and a toolbar where user can find the major commands needed to configure his/her preferences, this means how to let *Matlab* appear or behave. Then, the main interface is split into four majors areas as shown in Fig. **??**.

Area 1

it is the Command Window, where user can strike the Matlab commands and see results displayed on the same screen.

Area 2

It is the Workspace. All variables are saved in this area. If you need to know more about saved variables, just try this command on area 1:

```
>> whos
```

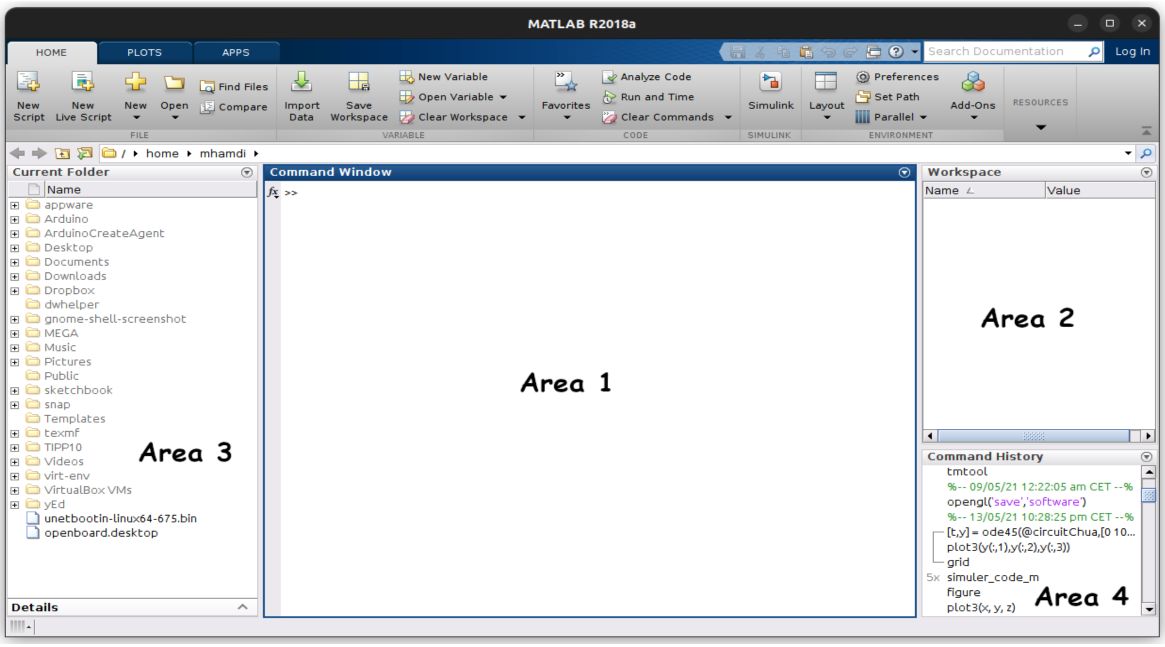
The results are the names of variables, the min and max values for every variables, class and the number of bytes needed to save it.

Area 3

It is the Current Folder, the path indicated by Matlab to execute a particular code. if you need to execute an algorithm which is not on the path shown by are 4, you have then to change the folder. Otherwise, an error message will pop-up on command window if it is not the absolute path.

Area 4

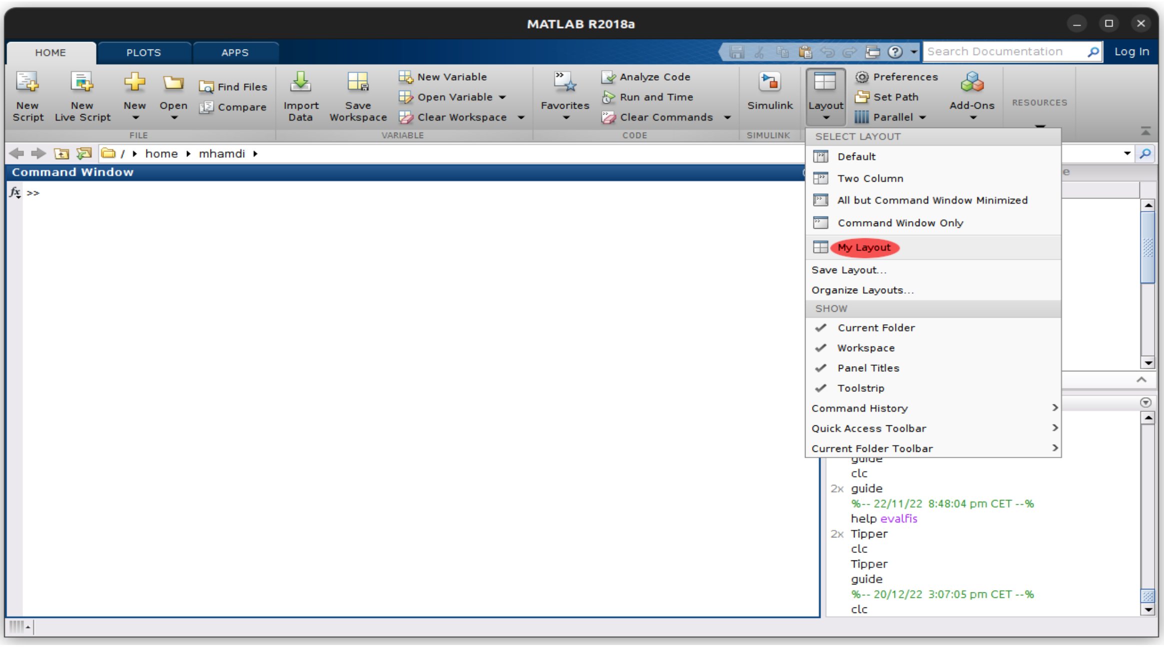
It is the Command History, where all instructions are saved in a panel from the date of installing Matlab on your machine until current time.



```
tmtool
%-- 09/05/21 12:22:05 am CET --%
opengl('save','software')
%-- 13/05/21 10:28:25 pm CET --%
[t,y] = ode45(@circuitChua,[0 10...
plot3(y(:,1),y(:,2),y(:,3))
grid
5x simuler_code_m
figure
plot3(x, y, z)
```

How to personalize the desktop layout

In the menu bar, click on **Layout** then load your preferences. For example, here the customized layout was saved under the name My Layout. You just click on it, and your layout appears like the display shown on the following image.



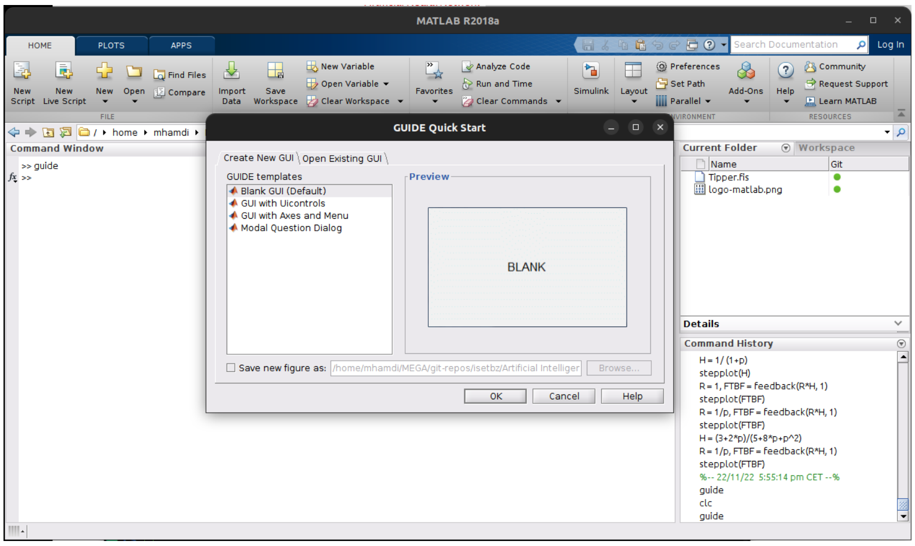
```
guide
clc
2x guide
%-- 22/11/22 8:48:04 pm CET --%
help evalfis
2x Tipper
clc
Tipper
guide
%-- 20/12/22 3:07:05 pm CET --%
clc
```

Matlab is not just an environment for technical computing, where we can solve equations. It is also an environment of graphical interfaces development. This integrated software is called **"GUIDE"**, Graphical User Interface Development Environment.

Try, at the command window, the instruction:

```
>> guide
```

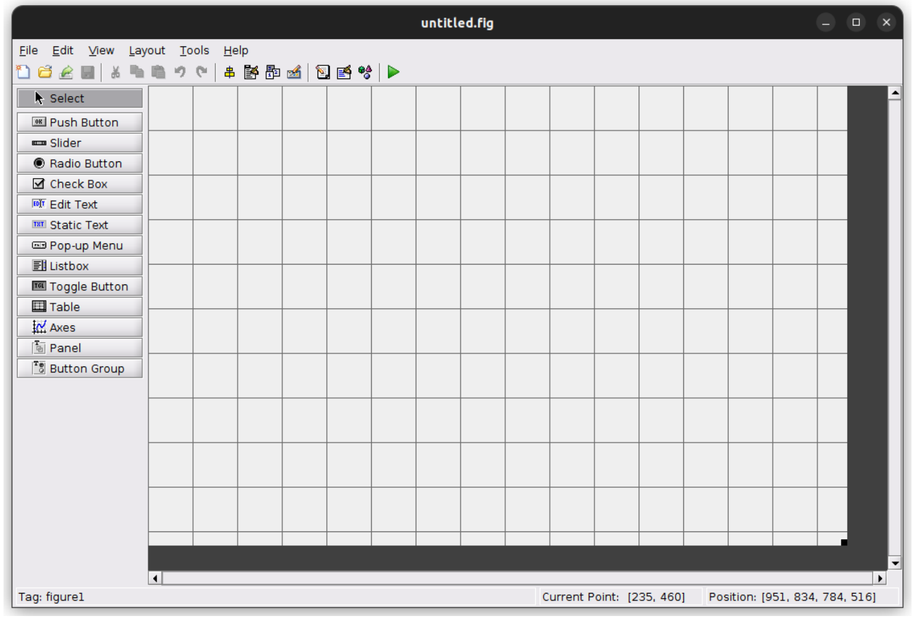
or simply move your mouse on the toolbar, you will have a shortcut like the icon . Just click on it, you will then launch the environment of graphics. You will get on your screen, the dialog shown here. You can access the templates or simply click next.



```
>> guide
>>
```

```
H = 1/ (1+p)
stepplot(H)
R = 1, FTBF = feedback(R*H, 1)
stepplot(FTBF)
R = 1/p, FTBF = feedback(R*H, 1)
stepplot(FTBF)
H = (3+2*p)/(5+8*p+p^2)
R = 1/p, FTBF = feedback(R*H, 1)
stepplot(FTBF)
%-- 22/11/22  5:55:14 pm CET --%
guide
clc
guide
```

Now, the environment is ready to be used.

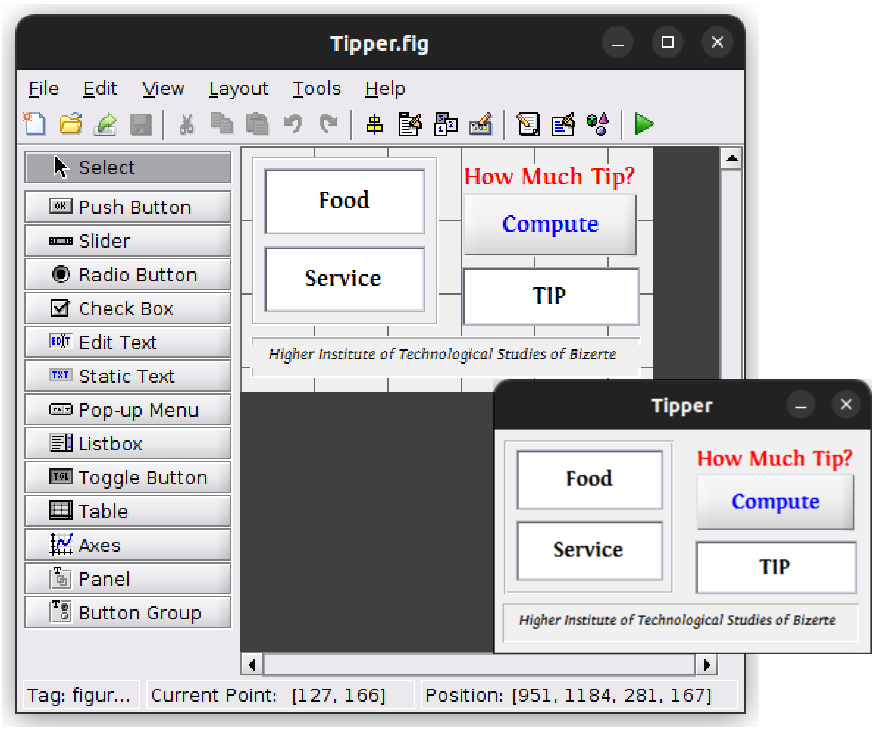


Save your work, you will notice the creation of two files on you directory with the same name, with two different extensions, one has this termination *.*fig* an the second one is an *m-file*.

The Fig-file contains the graphical declaration or the instantiation of the all used graphical objects in your interface. The main GUI that appears to user in order to customize his interface contains a palette where you can easily access the preconfigured objects like “pushbutton”, “toggle button” and “slider”. You can even add some external objects via the activeX control which is not in our current scope. Try to place and organize the required objects.

Notice that you have for any object an Object Browser which, when pointed on an object, list all properties relative to this one and even the set of methods called when evoking it. You may also need to know that there exist an uipanel, use it in case of you need to regroup elements by similarities.

Now, you can launch your application. Yourfinal interface may look like the one indicated below.



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
% 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);

```

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

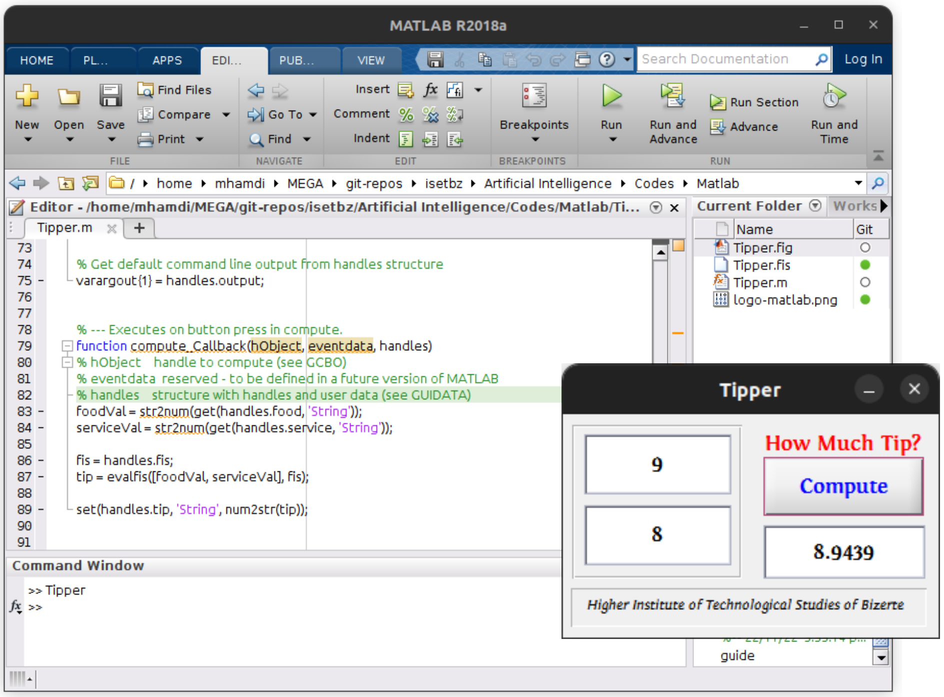
```matlab
% UIWAIT makes Tipper wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```matlab
% --- 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
% 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.



```matlab

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- 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
% 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));

```

```
>> Tipper
>>
```

9	8.9439
8	

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.
```

```julia
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")
```

```julia

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

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

★ Design a fuzzy system to pick the most adequate candidates out of the applicants to some master program.

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

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.

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.

```julia
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
)

md"""
## Interview
The variable `interview` describes the score given to an apllicant after passing the interview test.
"""
interview = Dict(
        "A" => TriangularMF(0, 0, 2),
        "B" => TriangularMF(1, 4, 6),
        "C" => TriangularMF(5, 8, 10),
        "D" => TriangularMF(9, 10, 10)
)
data_interview = chart_prepare(interview, input)
plot(
        input, data_interview["values"],
        label=data_interview["names"],
        legend=:bottomright
)

md"It is time now to design the variable `criteria` which aggregates both `application` and `interview`."
criteria = [application, interview]

md"""
## Decision
As for the output, we designate by `decision` the final status of any given application.
"""
decision = Dict(
        "Rejected" => TrapezoidalMF(0, 0, 2, 7),
        "Accepted" => TrapezoidalMF(3, 8, 10, 10)
)
data_decision = chart_prepare(decision, input)
plot(
        input, data_decision["values"],
        label=data_decision["names"],
        legend=:inside
)

md"""
## Set of Rules
"""
```

```

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)
```

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 and implement a fuzzy regulator to control the position of a disk.

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

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.

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

★ Dipping one's toes into the world of artificial neural networks (ANNs);
★ Demonstrate the ability of ANNs to learn and generalize from training data.

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

The XOR (exclusive OR) gate is a 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, students can 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.

```julia
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, σ ),
            )

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) = (:,)
```

```julia
                        # 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))
```

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

- ⋆ Train an artificial neural network to predict and classify categorical outcomes.

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. 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.

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.

```julia
using Markdown

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

md"Hyperparameters tuning"
η, 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`"
```

```julia
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(η);
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) |> σ;
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)
```

```julia
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)
```

7 Project Assessment

The final project will offer you the possibility to cover in depth a topic discussed in class which interests you, and you like to know more about it. The overall goal is to provide you with a challenging but achievable assessment that allows you to demonstrate your knowledge and skills in fuzzy logic or 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 have to provide all necessary resources, such as sample code, relevant datasets, as well as creating a set of slides to present your work. You are expected to demonstrate your understanding of the material covered throughout this course by reviewing the basics of fuzzy logic or neural networks, as well as familiarizing yourselves with relevant programming languages and libraries. 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**.)*

PROJECT PROPOSAL

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)*

PROJECT REPORT

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/jlai/tree/main/Codes/Report. 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*.