Overview

What is MATLAB (MATrix LABoratory)?

- A scripting (or interpreted) programming language
- Matrix oriented functionality
- Graphical visulization and cross-platform working environment
- Good performance on linear algebra and numerical computations
- Speciality on different areas (Deep learning, signal processing, communication, control, robotics, etc.)
- Approximately 80+ licensed toolboxes and many more 3rd party libraries from MATLAB community
- Frequently used in academic research

Applications

- Data acquisition, test and measurement
- Signal processing and communication
- Image processing and computer vision
- Aerospace industry
- Automotive industry
- Control systems
- Robotics and autonomous systems
- Computational finance
- Computational biology

SIMULINK

- Integrated product in MATLAB
- Modeling based functionality
- Models of physical phenomena and block diagrams
- Real-time simulation and testing

What is the tutorial about?

- Gain a general understanding of MATLAB
- Introduce the basics and simple functionality
- Prepare for the projects throughout this semester
- Implement a mini project

Live vs offline scripting

- .mlx vs .m files
- Live scripts allow to see the outputs of the code section immediately
- Large projects are generally implemented offline

Interface

- Menus, file browser, current folder, command window, workspace, command history, etc.
- Directory/folder

```
% Uncomment the commands starting with lowercase letters
% Print the current directory
% pwd
% Show the items in the current directory
% ls
% Make a new directory (folder)
% mkdir ee101_tutorial
% Copy a file to the new directory
% copyfile ee_101_matlab_tutorial.pdf ee101_tutorial

% Change current directory to the new folder
% cd ee101_tutorial
% pwd
% ls

% Delete a file
% delete ee_101_matlab_tutorial.pdf
% ls
% Return back to the parent directory
% cd ..
% pwd
% ls
% Delete a directory
% rmdir ee101_tutorial
% ls
```

- Command window

```
% Create a variable with a value
x = 2;
% Apply addition operation on x
x + 2
% Most recent answer
% Unless you assign a quantity to a variable, the result
% will be stored in a variable called "ans"
ans

% Clear command window
clc

% Generate some variables with some values
a = 5;
b = a+3;
```

```
% Clear all variables and functions from workspace(memory)
clear all
```

- Multi-line commands

```
% Create a matrix with multiple lines
% "..." allows you to start from the begining
% of the next line of code, you must treat the
% expression as a one-line code
M1 = [11 12 13 14 15; ...
 21 22 23 24 25; ...
 31 32 33 34 35]
% You can also comment out a specific line
M2 = [11 12 13 14 15; ...
 ... 21 22 23 24 25; ...
 31 32 33 34 35]
```

- Commenting

```
% Comments are highly useful to read the code
% You can put your comments to remember where you left
% in your code or what you did last time.
% You can use comments to display units (kg,m,etc.)
%{
 You can make multi-
 line comments using
 "%{ }%" as shown here
%}
% You can also want to comment out some part of your
% code to modify those lines with a new version
% To do this, select the lines of code type Ctrl+R on
% your keyboard. To bring those lines back, this time,
% type Ctrl+Shift+R
```

- Semi colon

```
% Semi colon allows you to hide a result
% It is sometimes useful to see the values on the
% command window for debugging purposes
hiddenVar = sqrt(16);
shownVar = min([4, -3, 21, 0, -5])
```

- Common operators

```
% Arithmetic operators: In addition to four operations
[1;2;3].*[2;4;6] % Element-wise multiplication
[2;8;36]./[2;4;6] % Element-wise right-division
[2;8;36].\[3;4;36] % Element-wise left-division
```

```
[1 1 0; 2 1 -1; 0 1 1]/[2 1 2; 0 1 0; -1 0 0] % Matrix right-division (faster)
[1 1 0; 2 1 -1; 0 1 1]*inv([2 1 2; 0 1 0; -1 0 0]) % Same with matrix inverse
```

```
[1 1 1; 2 1 -1; 0 1 1]\[2 1 2; 0 1 0; -1 0 0] % Matrix left-division (faster)
inv([1 1 1; 2 1 -1; 0 1 1])*[2 1 2; 0 1 0; -1 0 0] % Same with matrix inverse
```

```
M3 = [1 2; 3 4] % Generate a matrix
M3tr = [1 2; 3 4]' % Transpose of M3
```

```
% Relational operators
max(1:5) == mean(2:2:8) % Are both sides equal
[1;2;3] == [1;4;0] % Element-wise equality check
isSame = 5 == 4 % You can assign the result of a relation
isLess = 4 <= 6 % to a variable
```

```
% Logical operators
if (4 ~= 3 && 5 > 0) % if all the expressions are true
 disp("We are good!") % execute this line
end
```

```
if (isequal(4,5) || isnan(sqrt(-4)) || ischar('x')) % Only one expression must
    disp("Hello world!")                            % be true to execute this
end
```

- Saving

```
vec = linspace(.1,10,50) % Generate linearly spaced 50 numbers bw/ 0.1 and 10
mat = eye(2,5)           % Generate a 2x5 identity matrix (although not
                         % mathematically correct)
randVals = rand(2,5)     % Generate uniformly distributed random numbers
                         % bw/ 0 and 1 in a 2x5 matrix
save('savedFile','vec','mat','randVals') % Save the variables above inside
                                         % savedFile.mat file in the current directory
```

- Loading

```
clear all % Make sure no variables exist in workspace
```

```
load('savedFile.mat') % Load all the data inside .mat file
newVec = vec % You can use previously saved variables
newMat = mat*randVals' % as you want
```

Variables

- Special variables & constants

```
var1 = 0.5; var2 = 33; var3 = 1.2e5; % Multiple assignments
who % List the variables in workspace
whos % List the variables with details in workspace
eps % Accuracy of floating-point precision
i,j
1/Inf
Inf == NaN % Undefined numerical result (Not a Number)
```

- Global variables

```
global param
param = 0.5;
myfun2(5,1) % A function must be defined at the bottom of a file
param       % Therefore, myfun2 cannot be defined here
```

Input/Output & Formatting

- Commands

```
var2Disp = num2str(eps) % Convert the value of eps into a
 % string
disp(var2Disp) % Display the value as string
fprintf("This is the value of epsilon : %e",eps) % Print a value without
 % conversion
msg = "Please enter a real number: ";
num = input(msg,"s") % Ask the user to enter a number
format long % Show 16 digits after decimal point
longNum = str2double(num)*pi % Convert the input value into a double
format short % Show 4 digits (default)
shortNum = str2double(num)*pi
```

Decision making

- If statement

```
varDec = 0;
if varDec > 5 % If this is correct
```

```
 disp('There is something wrong in math!') % Execute this
elseif varDec < 0 % If that is correct
 disp('Sorry, I have never seen a negative number') % Execute that
else % If the expressions above are not correct
 disp('Non of the above is true!') % Execute the last function here
end
```

- For, while loops

```
for i=1:25 % Generate a vector to form a counter inside for loop
 disp(i) % Display the counter value
end
```

```
isBelowTen = true; % Define a meaningful logical variable
k = 0; % Iniitialize a variable to manipulate
while isBelowTen % Check if k is less than 10. Execute the loop while
 disp(k) % this is true
 k=k+1; % Increase k
 if k>=10 % Check if k becomes larger than 10
 disp(['This is the final value o k: ' num2str(k)])
 isBelowTen = false; % Make the loop terminate due to this change
 end
end
```

- break, continue

```
a1 = 10;
% while loop execution
while (a1 < 20 )
 fprintf('value of a: %d\n', a1);
 a1 = a1 + 1;
 if( a1 > 15)
 % terminate the loop using break statement
 break;
 end
end
```

```
a2 = 9;
%while loop execution
while a2 < 20
 a2 = a2 + 1;
 if a2 == 15
    % skip the iteration
    continue;
 end
 fprintf('value of a: %d\n', a2);
```

```
end
```

Arrays

- Vectors & matrices

```
rowVec = [0.2 -1 4 2 0]                     % Generate a row vector
colVec = [-2; 11; 3; 4.3; 44]               % Generate a column vector
aConstant = rowVec*colVec                   % Multiply vectors to obtain a scalar
aMatrix = colVec*rowVec                     % Multiply vectors to obtain a matrix
M4 = [1 0.2 0.3; 1.1 1.2 3; 2.1 2.2 2.3]    % Generate a matrix
```

- Cell arrays (Slower than vectors & matrices)

```
c1 = {'a',2,[1;2;3],"Hello World"}  % Generate a cell array with different
                                    % type of elements
c1{1,3}                             % Access the content of the data in a
                                    % cell with curly bracket
c1(1,3)                             % Access only the data in a cell with
                                    % paranthesis
c2 = {{'1',1,[1;1]};{'ee100'}}      % Generate a cell array of two other
                                    % cell arrays
c2{1,1}{3}                          % Access the third element of the cell
                                    % in cell (1,1)
```

- Initializing arrays

```
oneMat = ones(2,3)       % A 2x3 matrix with entries all ones
size(oneMat)             % Obtain the size of a matrix/vector
length(oneMat)           % Obtain the length of a matrix/vector (different than size)
zeroVec = zeros(4,1)     % A 4x1 vector with entries all zeros
ident = eye(3)           % 3x3 identity matrix
ident2 = eye(5,3)        % 5x3 identity matrix
multiDarr = zeros(2,2,3)% A 2x2x3 multi-dimensional (3D) array
randVec = randn(3,4)     % A 3x4 matrix whose elements are random
```

- Logical indexing

```
multiDarr(:,:,2) = [11 12; 21 22]   % Assign the second matrix of the 3D array with a v
a = zeros(3)                        % Generate a 3x3 zero matrix
a(1,1:2) = [1 2]                    % Partial assignment, assign the first two
                                    % elements of the first row with a row vector
```

- Array operations & functions

```
% Some of the useful array functions
% cat
% find
% linspace
% logspace
% max
% min
% prod
% reshape
% sort
% sum
```

Plotting

```
close all % Close all open windows
angles = 0:pi/90:2*pi; % Define angles vector
y = sin(angles); % Find sine of the angles
plot(angles,y) % plot(x axis,y axis)
```

- Axes, limits, labels, titles, legend, hold on

```
grid on     % Enable grid view
axis equal  % Set the aspect ratio to make data units the same in every
            % direction
xlim([0 8]) % Set limits on x axis
ylim([-2 2])% Set limits on y axis
title('Sinusoidal Plot')    % Give the figure a title
xlabel('x')                 % Name the x axis as x
ylabel('sin(x)')            % Name the y axis as y
hold on                     % Hold the previous plots on the figure before plotting new
angles2 = angles(1:floor(length(angles)/10):end);   % Collect 10 samples from the angle
y2 = sin(angles2);                                  % Generate those 10 points on sine
scatter(angles2,y2,'filled','Color','m')    % Plot the dots
legend('Sine function','Specific values')   % Put a legend on the figure
```

- Stem plot

```
figure
X = linspace(0,2*pi,50)'; % Generate 50 linearly spaced points between 0 and 2pi
Y = [cos(X), 0.5*sin(X)]; % Generate a matrix with sine and cosine given X
stem(Y) % Plot discrete points on Y
```

Help

- help command
- doc command

```
help help
doc plot
```

- Online resources, links, mathworks documentation, stackoverflow, tutorialpoint
- https://www.mathworks.com/help/matlab/
- https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted
- https://stackoverflow.com/
- http://www.hkn.umn.edu/resources/files/matlab/MatlabCommands.pdf
- You can also follow brilliant people on these stuff
- Make use of the power of "how to"

Functions

- Creating and calling functions

```
% Execute custom functions
custom_disp("This is to display only x")
custom_disp('This is to display x', 'and y')
factorial_ee100(4)
add(3,-1)

% Use help to display the documentation of a function
% To make this help function work for a custom  function, create the custom_func
% inside a separate .m file in a desired folder and add this directory to MATLAB's path
% help custom_func
```

```
% Functions with multiple outputs
[maxVal,idx1] = multiOut([0;4;-7;88;12;-132;1e4])    % Execute custom function
[~,idx] = multiOut([0;4;-7;88;12;-132;1e4])          % Same function with only desired o
                                                     % if you are not interested in that
```

```
% Anonymous functions
myfun = @(x,y) sqrt(x+y)*max(x,y)    % You can make an inline function in this form
myfun(3,-2)                          % You must define an anonymous function before exec
```

Mini Project

- Simulate the 2D motion of a ball that collides the flat surface which is made of concrete
- State vector of the ball $\xi = [x;\ v_x;\ y;\ v_y]$

- Positions: x & y
- Velocities: v_x & v_y
- $v_x = \frac{\mathrm{d}}{\mathrm{d}t}x = \dot{x}$ and $v_y = \frac{\mathrm{d}}{\mathrm{d}t}y = \dot{y}$
- $x = x_0 + v_x t$
- $v_x = v_{0x}$: constant
- $y = y_0 + v_{0y}t - \frac{1}{2}\mathrm{g}t^2$
- $v_y = v_{0y} - \mathrm{g}t$
- Numerical derivation: $\frac{d}{\mathrm{d}t}f \simeq \lim_{\Delta t \to 0} \frac{f(t+1) - f(t)}{\Delta t}$
- Discrete time linear equations of motion:
- $x_{k+1} = x_k + v_x \Delta t$
- $v_{x_{k+1}} = v_{x_k}$
- $y_{k+1} = y_k + v_{y_k}\Delta t - \frac{1}{2}g\Delta t^2$
- $v_{y_{k+1}} = v_{y_k} - g\Delta t$
- Matrix representation of the equations: $\xi[k+1] = A\xi[k] + b$, where
- $A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and $b = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{2}g\Delta t^2 \\ -g\Delta t \end{bmatrix}$

Exercise

- Try different surfaces (e.g. a ramp, stairs, curvy surface, etc.)
- Put the ball in a box (Hint: Treat the plot area as inside the box and make each edge as one side as in the example project)
- There is a bug when the ball starts at (0,0). Can you solve it?

Running the mini project

```
% Execute the command below inserting the list of initial conditions as an argument
bouncing_ball([0.2; 1; 10; 1])
```

- Global variables

```
function res = myfun2(x,y)
 % This function is for implementing a global variable
 % example. It takes x and y as inputs and utilizes the
```

```
 % global variable param and manipulates it.
 global param
 res = param*(x+y);
 param = param+1;
end
```

- Recursion in functions

```
function res = factorial_ee100(num)
 % An example of recursive function. It takes a number as argument
 % and returns its factorial. Note that the function is called inside
 % its definition. This is called recursion in programming
 if num<=0
 res = 1;
 else
 res = num*factorial_ee100(num-1);
 end
end
```

- Use of variable (optional) arguments

```
function custom_disp(x,varargin)
    if nargin > 1 % Check if the number of arguments is greater than 1
        str = strcat(x," ",varargin); % Combine strings
        disp(str{1})
    else
        disp(x)
    end
end
```

- Documenting a function

```
function res = add(a,b)
 % This function takes a and b as arguments and returns their summation
 % as a result
 %
 % Argumments
 % a: Real number
 % b: Real number
 %
 % Output
 % res: Real number
 %
 % Example
 % val1 = 25; val2 = 06;
 % summ = add(val1, val2);
 res = a+b;
end
```

- Arguments & (multiple) outputs of a function

```
function [maxVal, idx] = multiOut(vec)
 % This function takes a vector as input and returns
 % its maximum element together with its index
 [maxVal, idx] = max(vec); % max function itself originally returns
 % multiplt outputs
end
```