Automatic Control Laboratory, ETH Zürich
Prof. John Lygeros

Revised by Niklas Schmid
Revision from: September 10, 2022

MATLAB Introduction

Day 1

In the following two days you will be introduced to MATLAB and encounter the great practical use of this program for any numerical task you may want to solve.

This introduction provides you with the most important commands and examples. In the afternoon, you will implement fun exercises yourself to learn MATLAB in a learning-by-doing fashion. There is always more than one way to solve the tasks, so you can explore what MATLAB has to offer.

Contents

Chapter 1

Theory

If you start MATLAB you will see the following user-interface. For now, just focus on the

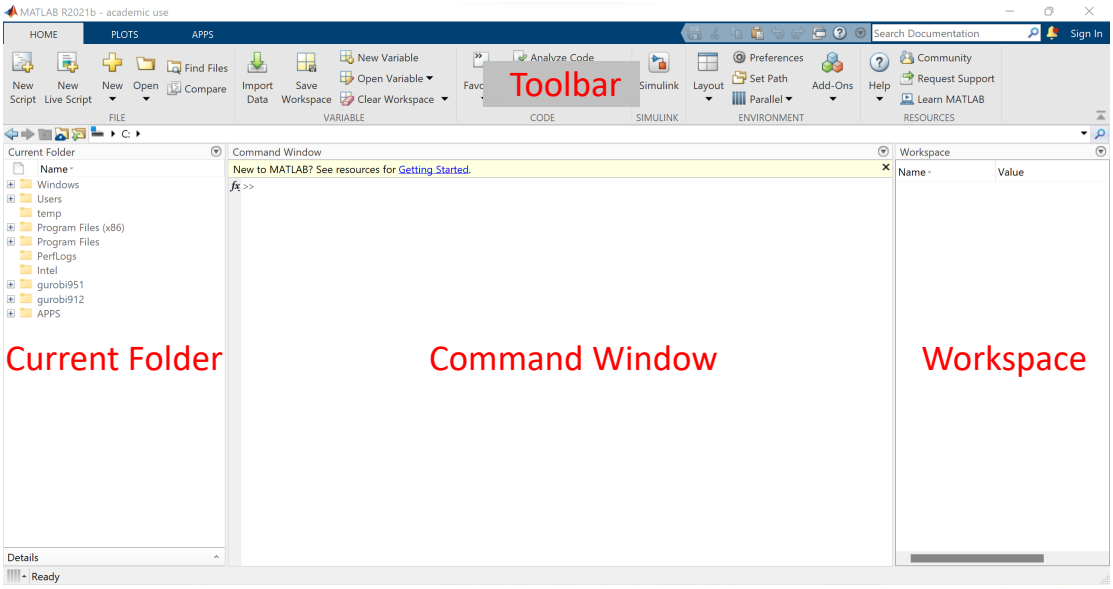

Figure 1.1: MATLAB User Interface

command window. MATLAB is an interpreted language. This means, that you can type any commands into the command window, MATLAB will then interpret your command and do the calculations in the background.

Let's try it. Type

```
a=2
```

into the command window and hit enter. This is how you declare a variable and assign a value to it. You can find all declared variables and their assigned value in the workspace (figure 1.1 on the right).

Once a variable is assigned a value, you can refer to it. Type

```
a
```

and hit enter. MATLAB will print the value of the variable.

You can add, subtract, multiply, divide and take the power of variables using the operators +,-,*,/,^, e.g.,

```
a + a
ans =
      4
```

If an expression is not assigned to a variable, it will be assigned to the intermediate variable *ans*. However, note that the variable *ans* will be overwritten every time this is done.

Also, you may want to suppress MATLAB from printing every result into the console. This can be done using a semicolon.

```
b = ans*a;
```

Notice, that MATLAB distinguishes between lower and upper case letters! A variable called A is not the same as a.

1.1 Getting help

MATLAB features a great amount of built-in functions for frequently used operations, such as computing the eigenvalues of a matrix *eig(A)* or exponentials *exp(a)*. All these commands are very well documented. If you have questions about a specific command you can either highlight the command and hit *F1* or type *help* followed by the name of the function into the console, e.g.,

```
help eig
```

In any case, the internet is your best friend. *Ctrl+c*, *Ctrl+v* are the programmer's favourite keys after all.

1.2 Datatypes

As in every other programming language, every variable you define and assign values to is associated with a special data type. This data type defines what type of data your variable contains (a number, text, etc.). Let's look at some data types MATLAB supports.

1.2.1 Numeric arrays

The most important data type in MATLAB are numeric arrays. Numeric arrays are just collections of numbers in the form of a scalar ($\mathbb{R}^{1\times1}$), a vector ($\mathbb{R}^{n\times1}$ or $\mathbb{R}^{1\times n}$) or a matrix ($\mathbb{R}^{n\times m}$). MATLAB thinks of all of them as the same type of data, just with different dimensionality.

You already learned how to define a scalar. You can define vectors and matrices using square brackets. Elements are separated using an empty space or a comma, and the start of a new row is indicated by using a line break or a semicolon.

```
A = [1,2;3,4]
A =
      1   2
      3   4
```

You can also enter matrices as elements of a matrix if you need to stack matrices.

```
B = [A
    2*A+4]
B =
       1    2
       3    4
       6    8
      10   12
```

You can get the size of a matrix using the command *size()*.

```
[n,m] = size(matrix1)
n =
     4
m =
     2
```

Structured matrices

Some matrices are used more often than others, so MATLAB gave them their own command. Below are some examples.

```
ones(1,3)
ans =
      1   1   1

zeros(2,3)
ans =
      0   0   0
      0   0   0

eye(2)
ans =
      1   0
      0   1
```

Also, MATLAB features commands to generate vectors with a specific spacing and range. You can use the double-dot operator to generate a vector with *start:step_length:end* or use the commands *linspace(start,end,steps)* or *logspace(start,end,steps)* for linear and logarithmic spacings, respectively.

```
1:3:11
ans =
      1   4   7   10

linspace(0,10,5)
ans =
      0   2.5   5   7.5   10

logspace(0,2,3)
ans =
      1   10   100
```

Accessing matrices

You can access specific values of a matrix using brackets and the indices of the rows and columns. You can define a range via *start_index:end_index*. If you want all elements starting at index j until the end you can write *j:end*. If you just put a double dot, that means that you want all elements in the specific dimension.

```
A = [1,2,3;4,5,6;7,8,9;10,11,12]
A =
    1    2    3
    4    5    6
    7    8    9
    10   11   12
```

```
A(1,2)
ans =
    2

A(2,:)
ans =
    4    5    6

A(2:3,:)
ans =
    4    5    6
    7    8    9
A(2:end,[1,3]])
ans =
    4    6
    7    9
    10   12
```

Unlike other programming languages, the indexing in MATLAB always starts with one, not zero!

Linear algebra

You can add, substract and multiply matrices using $+,-,^*$. The operator $'$ computes the transpose of a matrix. A matrix A is inverted using $A\hat{}(-1)$ or inv(A).

```
eye(3)*ones(2,3)
Error using *
Incorrect dimensions for matrix multiplication.

eye(3)*ones(2,3)'
ans =
    1    1
    1    1
    1    1
```

Elementwise operations can be done by placing a dot in front of the operator $.*$, $./$, $.\hat{}$. For instance, $c = a./b$ computes a matrix of the same size as a and b with entries $c(i,j) = a(i,j)/b(i,j)$.

Of course MATLAB does not only feature matrices.

Strings

Strings are variables containing text. To declare a variable as a String you can use apostrophes or quotation marks (MATLAB does not differentiate them). You can combine text by just adding two strings together.

```
some_text = "Hello ";
some_more_text = 'MATLAB';
some_text + some_more_text
ans =
   "Hello MATLAB"
```

1.2.2 Cell arrays

Cell arrays are a simple way of storing multiple matrices of different size and dimension. They are defined and accessed using curly brackets.

```
aCellArray = {1.3, 5*ones(20), zeros(2,4)}
  1x3 cell array
    {[1.3000]}   {20x20 double}   {2x4 double}

aCellArray{3}
ans =
     0     0     0     0
     0     0     0     0
```

1.2.3 Structures

Structures are a nice way to keep your code clean and sort related variables in a group. You just give your structure a name, e.g., player1, and associate attributes to it using a dot.

```
player1.name = 'John';
player1.class = 'Mathemagician';
player1.intelligence = 18;
player1.armor = 25;
```

or type

```
player2 = struct("name","Florian","class","Fighter","strength",17,"armor",41)
```

1.3 Files and Scripts

Working in MATLAB's console is sufficient if you just use it as a calculator. But if you are working on complex algorithms, you may want to save, debug and rerun your code multiple times without having to type every line into the console again. Therefore, MATLAB allows you to write scripts.

1.3.1 Scripts

Scripts are just files (ending with .m) containing MATLAB commands. You can open a new script by pressing the *New Script* button on the top left, on the toolbar. A new window will appear on top of the command window called *Editor*.

Type a few MATLAB commands into the command window that you have just learned. Then press the *Run* button in the toolbar in the Editor tab. MATLAB now asks you to store the file somewhere on your computer. Once you saved it, the commands in the script will be executed one after another, just as if you would have typed them into the console. However, you can now edit specific lines of your code without having to type everything into the console again.

You can add comments to your code using the percentage symbol. Everything following that symbol will not be interpreted as a command and thus will not be executed.

```
% Robot-arm parameters
arm_l = 2.3;    % Length (m)
arm_r = 0.1;    % Radius (m)
arm_m = 2;      % Mass   (kg)
```

You can only run files which are in the *Current Folder* shown in the respective MATLAB window. If you want to run files from a different directory, you can use the *addpath* command.

Hit *ctrl + s* to save the current status of your file. Do it. And do it often.

1.3.2 Functions

Very often you need specific sequences of code multiple times in your algorithm. For this purpose it is helpful to define your own functions in MATLAB. You can do this by creating a new script. Give the **same name** to your script file as you want to name your function, e.g. *my_awesome_function.m*. In the first line of the file you now define the outputs, the name of the function and the inputs to the function.

```
function [y1, y2] = my_awesome_function(x1, x2)
    y1 = x1 + x2;
    y2 = x1 - x2;
end
```

Then we can call the function inside another script, the console, or inside another function.

```
[u, v] = my_awesome_function(6, 4)
u =
      10
v =
      2
```

Your function does not have access to your workspace, but rather has its own workspace. It only knows about the variables you passed as inputs and the variables defined inside the function. Once the function finished, this workspace gets deleted and the outputs of the function are passed to the workspace of its caller.

1.3.3 Data

In order to save your workspace in a file and load it again inside a script you can use the *save* and *load* commands. E.g., the command

```
save('my_data.mat')
```

saves all the data that is currently in the workspace to a file called *my_data.mat*.

1.4 Displaying things

1.4.1 Displaying text

While you can print the content of your variables to the console by just leaving away the semicolon, you may sometimes want to combine it with text. There are several ways to achieve this. If you just want to display the content of a single variable, use the function *disp*. If you want to combine text with different variables you can use the function *fprintf*.

```
disp('Hello,')
a_number = 4;
fprintf('is it me you''re looking %d?',a_number)
```

The expression *%d* acts as a placeholder telling MATLAB that a number, in this case the variable *a_number*, shall be inserted.

1.4.2 Plotting

Matlab has excellent functions to design pretty plots and images. Let's create a new window for MATLAB where it can generate plots by typing the command

```
figure()
```

into the console. An empty figure appeared. Let's fill it with some plots. Therefore, type

```
x1=1:10
y1=x.^2
plot(x1,y1)
```

into the Matlab console. The plot command will automatically generate a graph plotting x against y and connecting the coordinates with a line. This means that the first coordinate is $(x(1), y(1))$, the second coordinate is $(x(2), y(2))$ and so on.

We can add a title, labels and change the limits of the x- and y-axis using the commands

```
title('My first plot')
xlabel('The x-axis')
ylabel('The y-axis')
xlim([1,9])
ylim([25,120])
```

If we just want to plot the coordinates without connecting them through a line we can use the *scatter()* command

```
x1=1:10
y1=x1.^2
scatter(x1,y1)
```

Doing so will overwrite the old plot. In order to plot multiple graphs we can

- create a new figure by typing *figure()* again
- create subplots using the command *subplot()*, where we create multiple independent plots within a figure
- plot multiple graphs in one figure using the command *hold on* to inform Matlab to draw again in the same plot

E.g.,

```
figure()
x=1:10
y=x.^2
plot(x,y)
hold on
plot(x,2*y)
```

or

```
figure()
x=1:10
y=x.^2
```

```
subplot(2,1,1)
plot(x,y)
subplot(2,1,2)
plot(x,2*y)
```

where the first two arguments of the *subplot* command are the number of rows and columns of plots that shall be put into the figure and the last argument is the index of the plot.

1.5 Loops and conditions

If condition

The syntax for if conditions is straightforward:

```
if condition
    statements
elseif
    statements
else
    statements
end
```

E.g., we can write

```
chocolate_consumption = 1.55;
productivity = 1.73;
if productivity>chocolate_consumption
    disp('I need chocolate.')
elseif productivity==chocolate_consumption
    disp('I keep my balance.')
else
    disp('I like chocolate.')
end
```

to compare two variables and print the result to the console. The *else* and *elseif* blocks are optional.

For loop

You can use the *for* command to design loops by iterating through elements of a vector. The for loop has the following structure:

```
for element = vector
    some commands
end
```

In every iteration, the variable *element* will take the value of one element of the vector. E.g., we can write

```
my_vector=1:10
for my_element = my_vector
    some_value = my_element^2-2;
    disp(some_value)
end
```

There also exists a while loop in MATLAB.

```
countdown = 5;
while countdown > 0
    disp(countdown)
```

```
    countdown = countdown - 1;
end
```

1.6 Debugging

If your script is producing unexpected outputs you can pause it at specific lines during execution and check the state of your workspace variables. Therefore, you can add break points by pressing the line number at which you would like your code to pause.

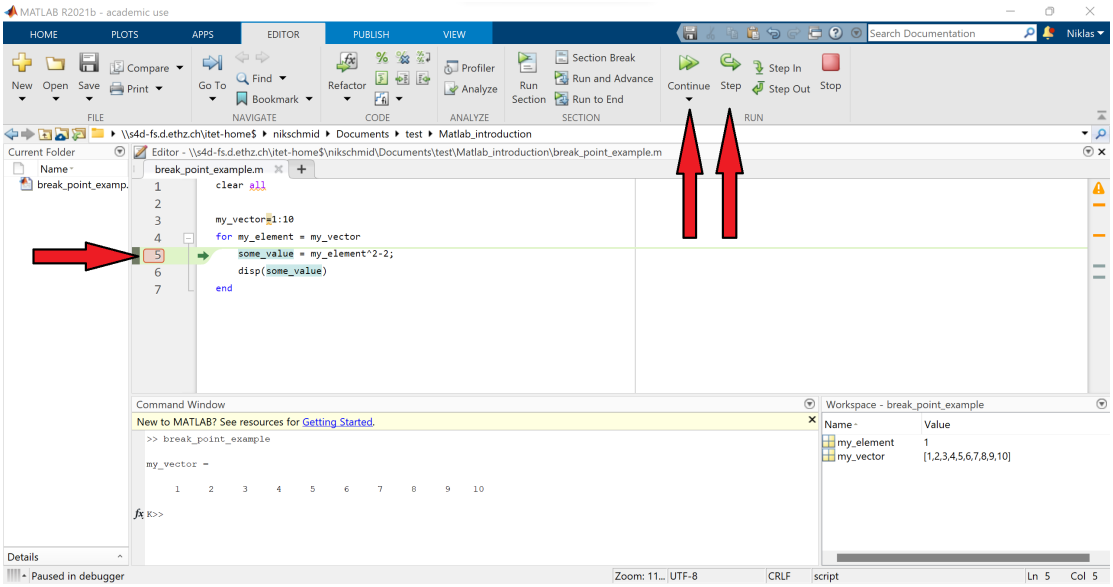

Figure 1.2: Debugging in MATLAB.

You can then walk through your code line by line and analyse its behaviour by pressing the *Step* button on the top, or you can run the code until it hits the next break point by pressing the *Continue* button.

Don't forget: If you execute a script or command, all variables will be stored in your workspace. One way to solve many unexpected issues is to clear your workspace using the *clear all* command. We recommend you to put this command in the beginning of every script. Otherwise it may happen, that your script uses variables inside your workspace that are remainders of a prior execution of a script.

Chapter 2

Practice

That was a lot of theory. But we want to learn programming in MATLAB, so let's jump straight into practice and consolidate some commands.

For your own sanity, generate a new file for every task. Do not try to solve them in the command prompt!

1. Generate a vector $[-5, -4, \ldots, 5]$ using the notation with the double-dots as well as using the command *linspace()* (see the part about *structured matrices in section 1.2.1*).
2. Compute and plot the function $f(x) = x^2$ on the domain $[-5, 5]$.
3. It is not quite christmas yet, but since we are getting closer and our anticipation rises, let's solve a riddle that most European children know very well: Drawing the The House of Nicholas (figure 2.1). It is now your task to plot this house. But there is a catch: You are only allowed to draw the house without lifting the pen or drawing a line twice. We impose this constraint by allowing you to only use vectors x and y consisting of nine elements each. Then you plot the house using the command *plot(x,y)*. Can you solve this riddle? (You do not need to plot it in different colors.)

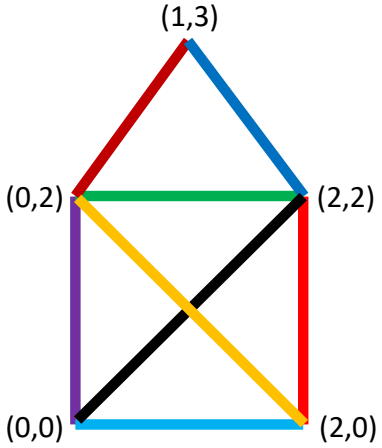

Figure 2.1: House of Nicholas

4. Design an animation: Plot the house line by line and add half a second breaks in between. *Helpful commands: pause()*
5. In this task we will approximate the number π: Generate a thousand random data points uniformly on a two dimensional plane in the interval $[-1, 1] \times [-1, 1]$. Check how many of your generated points lie **within** the unit circle $c_U = \{(x, y) : x^2 + y^2 = 1\}$. *Helpful commands: rand(), pi*

(a) How do you approximate π using this information (hint: What is the surface of the unit circle and the unit box)?

(b) Compute your approximation of π using this method.

(c) Let's visualize the results. Plot a **blue** box surrounding the interval $[-1, 1] \times [-1, 1]$ as well as the unit circle in **red**. Plot all your generated points as **black crosses** using the *scatter()* command. Hint: The unit circle is define by the coordinates $(sin(\theta), cos(\theta))$ for $\theta \in [0, 2\pi)$.

(d) What happens if you generate more points? Program a function *pi = approximatePi(N)* that computes an approximate value of π by generating N random points. Print your computed value of π for $10, 100, 1.000, 10.000$ points.