Introduction to MATLAB Environment and Programming

Department of the Built Environment, Building Acoustics Group

7LS8M0 Architectural Acoustics

version 1.0

February 2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Contents

1 Introduction

In the course 7LS8M0 Architectural Acoustics, the MATLAB programming language is going to be used for the purposes of the assignments and exercise sets. In general, MATLAB is used in many fields for the analysis of data, the development of algorithms, the creation of models and applications, as well as the visualization of different data.

This document focuses on the description of main commands in the MATLAB environment. More specifically, the first section explores the MATLAB environment and its main windows. In the second section, the MATLAB-used definitions are presented. The third section discusses the main operations in MATLAB related to numerical, relational, and logical operations. In the fourth section, the definition, and the construction of functions in MATLAB are summarized. In the fifth section, the different ways of visualizing data are described. At the end of the document a set of exercises is included, focusing on the tasks into this document.

2 MATLAB Environments

This section focuses on two main environments in MATLAB, corresponding to the general and the editor (i.e., script) environment.

2.1 HOME Environment

When the MATLAB programming software is operated for first time, the following screen is presented, including different windows (Figure 1).

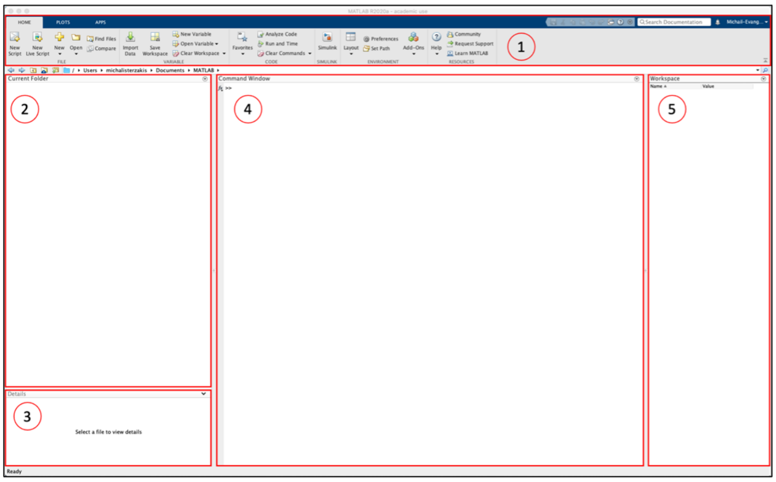


Figure 1: The `HOME` environment.

Five main windows have been distinguished in the HOME environment. The first window includes the main functions divided into three main tabs (i.e., `HOME`, `PLOTS`, and `APPS`). The second window, corresponding to `Current Folder`, indicates the location (i.e., directory path), in which the MATLAB operations are executed. The current directory is used by MATLAB as a reference point. In the case that a file is not included into the main directory path, it is needed either to be included into the current directory path or the directory path to be changed. In the third window (i.e., `Details`), information related to the selected file can be viewed. The `Command Window` could be characterized as the most important window. In this window, a variety of commands can be operated. The definition of variables, the running of MATLAB scripts and functions as well as the mathematical/arithmetical calculations are some of the main operations, which can be executed after pressing the enter key. It must be noticed that all the executable operations are saved in the history automatically and maintained even after the ending of MATLAB sessions. The user is able to retrieve previous operations, using the up-key arrow and browse to the history with both up- and down-key arrows. In addition, MATLAB errors are presented in this window, indicating the type of errors and sometimes the line where the line is occurred. For matter of convenience, the errors are presented in red coloured texts followed by a "beep" sound. The `Workspace` window is the second most important window, storing all the defined variables, including their names, their type as well as their dimensions. By double-clicking in any variable, detailed information is presented in a new window. In contrast to `Command Window`, the stored variables are cleaned up after quitting MATLAB. However, the work-space can be saved by the user in *.mat format.

2.2 Editor Environment

The editor is used for coding programs (M-files), following the string, `Home` → `New` → `Script`. When a new or an existing M-file is open, three new tabs are presented (i.e., `EDITOR`, `PUBLISH`, and `VIEW`). An illustration is presented in the Figure 2.

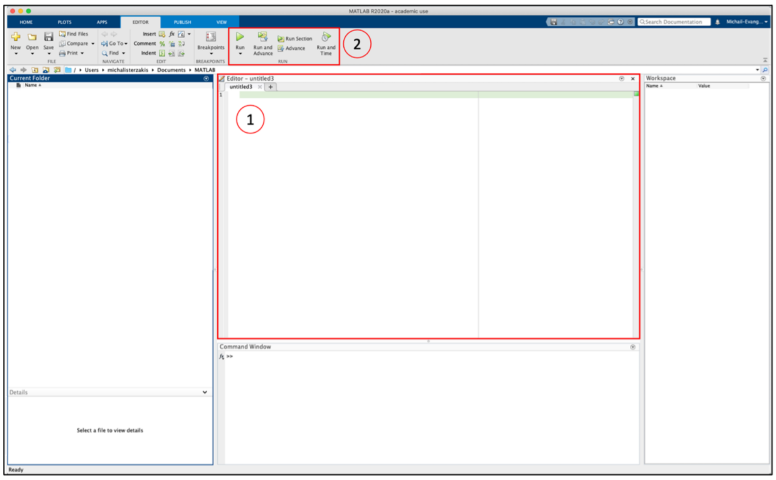


Figure 2: The `Editor` Environment.

The `Editor` window is used for writing codes or scripts. When a script is accomplished, the user can run the code by pressing the `Run` icon in the `Editor` Tab. Before running the script, it is important the script to be saved from the `File` sub-tab. From the same section, the running of a specific section can be executed. The `Run Section` operation does not require the saving of the constructed script beforehand. However, it is always recommended to save the scripts before any operation. In addition, it should be noted that the scripts are automatically saved before any `Run` execution.

3 Definitions

In programming languages, different definitions hold. These correspond to the definition of special characters, the eligible name of the variables, the types of variables, as well as the counting of the elements (i.e., indexes). It has to be noted that only numerical variables are considered for the purposes of this guideline. In the case that the reader is interested in other types of the variables, it is recommended to visit MATLAB official site.

3.1 Special Characters

In MATLAB, some special characters have been defined, executing specific procedures. The most important special characters have been summarized in the Table 1.

Table 1: Special characters in MATLAB.

Special Character Name	Symbol	Description
Squared Brackets	`[ ]`	Array construction
		Array concatenation
		Empty matrix and array element deletion
		Multiple output argument assignment
Parentheses	`( )`	Operator precedence
		Function arguments enclosure
		Indexing
Curly Brackets	`{ }`	Cell array assignment
		Cell array contents
Equal Sign	`=`	Assigning statements
Full-Stop Mark	`.`	Decimal point
		Element-by-element operation
		Structure field access
		Object property or method specifier
Ellipsis	`...`	Continuation to next line
Comma	`,`	Matrix subscripts
		Function arguments
Semicolon	`;`	Signify the end of a row
		Suppressing the output of a code line
Colon	`:`	Creation of a vector of equal space
		Indexing
		For-loop iteration
Single Quotes	`' '`	Character array constructor
Double Quotes	`" "`	String Constructor
Percent	`%`	Comment
		Conversion Specifier
Double Percent	`%%`	Specifying Section
Percent Curly Bracket	`%{%}`	Block comments
Exclamation Point	`!`	Operating system command
At Symbol	`@`	Function handle, construction and reference
		Calling superclass methods

3.2 Numerical Values

The numbers in MATLAB are defined based on conventional notation (e.g., `2`, `9.145`, or `100`). The minus sign defines the negative numbers, such as `-0.2123` or `-12`. The character "e" is used for specifying the power of ten scaling factors (e.g., `7e12`, `-9.25e-4`, or `3.2e-10`). Imaginary numbers are also defined in the MATLAB, by using the character `i` or `j` as a suffix (e.g., `94e-21j`, `-1i`, or `2.768j`).

3.3 Names of Variables

In MATLAB, the name of the variables should consist of

- at least one letter (e.g., `a`, `Ab`, or `aBc`),
- at least one letter, following by numbers (e.g., `a1`, `a1A`, or `AB2`), or
- at least one letter, following by underscore (e.g., `a_1`, `Ac_b2`, or `a_B_3`).

It is important to be noticed that MATLAB does not recognize uppercase and lowercase letter as the same. In addition, some variables have already been defined from MATLAB, such as `pi` variable, indicating that `pi= 3.14`.

3.4 Types of Numerical Variables

For the purposes of this document, only matrix-based numerical types of variables are considered. For sure, there are more types of variables with respect to the field of study. The Table 2 summarizes the most common array-based numerical variables.

Table 2: Numerical variables in MATLAB.

Numeric Variable	Definition	Description
Single Number	`a = 1`	The value of `1` is given to the variable with name `a` (i.e., scalar)
Row Vector	`r = [1 2]`	The values of `1` and `2` are stored in the row vector with name `a` (i.e., 1x2 vector)
Column Vector	`c = [1;2]`	The values of `1` and `2` are stored in the column vector with name `a` (i.e., 2x1 vector)
Matrix	`M = [[1 2];[3 4]]`	The values of `1,2` and `3,4` are respectively stored in the first and second row of a matrix with name `a` (i.e., 2x2 matrix)

In addition to the Table 2, the elements in the row vectors can also be separated by a comma, such as `r = [1,2]`. The same procedure can be also used in matrices, such as `M = [[1,2];[3,4]]`. In the case of matrices, the squared brackets, defining the elements in the rows, can be neglected such, `M = [1 2 ; 3 4] = [1,2 ; 3,4]`.

3.5 Elements Counting (Indexes)

When vectors and matrices are defined, the counting of their elements starting always with one. For example, consider the row vector (or row array) `a = [4 5]`, the elements `4` and `5` are stored in the positions with indexes one and two, respectively (i.e., `a(1)` and `a(2)`). Some examples are presented below.

```
r = [4 5 6]      % Definition of a row array with dimensions 1x3.
r(2)             % Extract the second element of the row array, or similarly
r(1,2)           % Extract the second element of the row array.
c = [4;5;6]      % Definition of a column array with dimensions 3x1.
c(2)             % Extract the second element of column array, or similarly,
c(2,1)           % Extract the second element of column array.
```

By considering matrices, the same procedure can be implemented. However, since multiple rows and columns are included, it is strongly recommended the definition of both rows and columns for extracting the values under-consideration. Some examples are given below.

```
m = [[5 6 7]; [8 9 10];[11 12 13]]  % Assignment of a 3x3 matrix in a variable named m.

m(3,3)              % Extract the element stored into 3rd row and 3rd column.
m(2,:)              % Extract all the elements stored into 2nd row.

m(2,[1:2])          % Extract the 1st and 2nd element stored into the 2nd row.
m(2,[1,3])          % Extract the 1st and 3rd element stored into the 2nd row.

m(:,3)              % Extract all the elements stored into 3rd column.

m([1:2],3)          % Extract the 1st and 2nd element stored into the 3rd column.
m([1,3],2)          % Extract the 1st and 3rd element stored into the 2nd column.

m([1:3],[1:3])      % Extract the elements stored into the 1st, 2nd, % and 3rd row and column.
m(1:2:3,1:2:3)      % Extract the elements stored into the 1st and 3rd row and column.
m(1:2:end,1:2:end)  % Extract the elements stored into the 1st and 3rd row and column.
```

4 Operations with MATLAB

Numerical, rational, and logical operations are allowed in MATLAB. In this section the most important operations have been summarized.

4.1 Numerical Operations

Numerical operations can be assigned into MATLAB, following a conventional notation. The numerical operations have been summarized in the Table 3.

Table 3: Numerical Operators in MATLAB.

Symbol	Numerical Operation
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Right Division
$\backslash$	Left Division
$\wedge$	Power
'	Transpose

Attention, when arrays or matrixes are considered in numerical operations, the inclusion of a full-stop punctuation mark (.) before a multiplication, division, and power operation is needed for the conduction of element-wise operations.

4.2 Relational Operations

In MATLAB, the relational operators have been defined such these presented in Table 4.

Table 4: Relational Operators in MATLAB.

Symbol	Operation
$==$	Equal to
$>$	Greater than
$<$	Less than
$>=$	Greater than or equal to
$<=$	Less than or equal to
$\sim=$	Different than

Relational operators provide comparison between the elements in two arrays. True (i.e., assigned by `1`) and False (i.e., assigned by `0`) values are returned, indicating the holding or not of the relation. The returned logical array is of the same size as the relational arrays. This indicates that the arrays or matrices, which are compared, are needed to be of the same size. An exemption is hold when a relational operation is occurred between a scalar value and an array or a matrix. In this case, the returned logical array is of the size of the array or matrix. Some examples are given below.

Table 5: Relational operation examples.

Input	Output
`[1 2 3]>[4 5 6]`	`[0 0 0]`
`[1 2 3]<[4 5 6]`	`[1 1 1]`
`2>[1 2 3]`	`[1 0 0]`
`2<=[1 2 3]`	`[0 1 1]`
`[1 2 3 ; 4 5 6]>[ 3 4 5 ; 9 10 1]`	`[0 0 0 ; 0 0 1]`
`5>=[ 3 4 5 ; 9 10 1]`	`[1 1 1 ; 0 0 1]`

4.3 Logical Operations

For the evaluation of logical operations, the short-circuit `AND` and `OR` Boolean operators defined by the symbols `&&` and || can be respectively used in MATLAB. Similar to rational operations, the logical operations return the value of one, if the evaluation is true, and zero, if the evaluation is false. The syntax of these logical operations with short-circuiting is presented in the Table 6.

Table 6: Logical operation syntax.

Logical Operations
`Expression A && Expression B`
`Expression A || Expression B`

5 Functions

In MATLAB users have the opportunity to use its pre-defined functions as well as to construct their own functions. In this section, the most important basic and intermediate MATLAB functions are summarized. The command help can be used for further information related to a specific function (i.e., either predefined or user-based defined) in the `Command Window`, such `help name_of_function`.

5.1 Predefined Functions

A high number of functions is provided by MATLAB, helping users to program their own scripts, following the same structure and rules. Some of the most important build-in functions are summarized below (Table 7).

Table 7: Pre-defined functions.

Function	Description	Function	Description
`sin(x)`	Sine	`max(A)`	Maximum elements of an array
`asin(x)`	Inverse sine	`inv(A)`	Matrix inverse
`sinh(x)`	Hyperbolic sine	`sum(A)`	Sum of elements
`asinh(x)`	Inverse hyperbolic sine	`mean(A)`	Average or mean value
`cos(x)`	Cosine	`median(A)`	Median value
`acos(x)`	Inverse cosine	`std(A)`	Standard deviation
`cosh(x)`	Hyperbolic cosine	`cov(A)`	Covariance
`acosh(x)`	Inverse hyperbolic cosine	`eye(n)`	Identity Matrix
`tan(x)`	Tangent	`diag(A)`	Diagonals of a matrix
`atan(x)`	Inverse tangent	`sign(n)`	Sign function
`tanh(x)`	Hyperbolic tangent	`sqrt(x)`	Square root
`atanh(x)`	Inverse hyperbolic tangent	`zeros(m,n)`	Matrix of zeros (m-by-n)
`cot(x)`	Cotangent	`ones(m,n)`	Matrix of ones (m-by-n)
`acot(x)`	Inverse Cotangent	`rand(m,n)`	Uniformly distributed random numbers
`coth(x)`	Hyperbolic cotangent	`randn(m,n)`	Normally distributed random numbers
`acoth(x)`	Inverse hyperbolic cotangent	`size(A)`	Array dimensions
`conv(x,y)`	Convolution	`length(A)`	Length of an array
`fft(x)`	Fast Fourier Transform	`numel(A)`	Number of elements in array
`ifft(x)`	Inverse Fast Fourier Transform	`flip(A)`	Flip the order of elements
`exp(x)`	Exponential	`repmat(A,n)`	Repeat copies of arrays
`log(x)`	Natural logarithm	`any(A)`	Check if any/all elements are nonzero
`log10(x)`	Base 10 logarithm	`nnz(A)`	Number of nonzero array elements
`abs(x)`	Absolute value	`find(A)`	Find indices of nonzero elements
`real(z)`	Real part	`round(x)`	Rounds towards nearest decimal/integer
`imag(z)`	Imagine part	`floor(x)`	Round towards minus infinity
`conj(z)`	Complex conjugate	`ceil(x)`	Round towards plus infinity
`angle(z)`	Phase angle in radians	`sort(A)`	Sort in ascending or descending order
`min(A)`	Minimum elements of array	`linspace(x1,x2)`	Generation of linearly spaced vector

It has to be noted that the arguments in the trigonometric functions are in radians. Hence, it is strongly recommended to use radians-based trigonometric functions rather than degrees-based ones.

5.2 User-based Defined Functions

In MATLAB, users have the opportunity to create their own functions, reducing the length in their main scripts. User-based functions can be used in any script, similar to build-in functions. After creating a function (i.e., `Home` → `New` → `Function`), the following editor is presented.

```
1: function [outputArg1,outputArg2] = untitled3(inputArg1,inputArg2)
2: %UNTITLED3 Summary of this function goes here
3: %   Detailed explanation goes here
4: outputArg1 = inputArg1;
5: outputArg2 = inputArg2;
6: end
```

Focusing on its editor, a reference code is presented. All the functions start with their definitions (line 1) and finish with the keyword end (line 6). In the definition line, the name of the function is given, substituting the untitled name. It is a good practice, the given name to describe the scope of the function. For example, if a function is constructed, calculating the roots of a quadratic polynomial function, a given name could be `root2fun` or `Roots2Poly`. In the same line, the input arguments (i.e., given in the parentheses and separated by commas) and outputs arguments (i.e., given in the squared brackets and separated by commas) are defined. By considering the aforementioned example, suppose that a user needs to provide the polynomial coefficients a, b, and constant c, for extracting the output related to the calculated roots r1 and r2. In the line 2, a summary of the constructed function is strongly recommended to be given, helping users to understand the scope of the user-based function. A more detailed explanation of the defined function could be given in the next line. It has to be mentioned that the lines 2 and 3 are in comments. Comments after these blocks are ignored by MATALB. The final part, corresponding to the lines 4 and 5 indicate the main body of the function. There, the coding of the under-consideration task is conducted. In contrast to the scripts, functions cannot be running in a direct way. For running the functions, this can be achieved in the script, where the function is called, or in the command window. In addition, the calculated values into the functions are not presented in the `Workspace`. The example of calculating the roots of a quadratic polynomial function is presented below.

```
function [r1,r2] = Roots2Poly(a,b,c)
%Roots2Poly: Calculation of the roots of a quadratic polynomial.
%-------------------------------------------------------------
%   Inputs:
%-------------------------------------------------------------
%   a: The coefficient of x^2 (Scalar).
%   b: The coefficient of x (Scalar).
%   c: The constant (Scalar).
%-------------------------------------------------------------
%   Outputs:
%-------------------------------------------------------------
%   r1: Root 1 (Scalar).
%   r2: Root 2 (Scalar).

SqNom = sqrt(b^2 - 4*a*c);% Calculation of the square root term.

r1 = (-b + SqNom)/(2*a);   % Calculation of the first root.
r2 = (-b - SqNom)/(2*a);   % Calculation of the second root.

end
```

6 Conditional/Control Statements

Conditional and control statements are used for the evaluating statements. In this section, the most important conditional and control statements are presented.

6.1 if Statements

The simplest conditional statement is an `if` statement. This statement evaluates an expression and executes the condition of a group of statements when it is true. An expression is defined as true when, i) its result is non-empty and ii) its result contains only non-zero elements. In any other case, the expression is characterized as false. When multiple groups are needed to be evaluated, the optional blocks `elseif` and `else` can be used. An if conditional statement starts with the keyword if, following by the under-evaluation statement, and finishes with the end keyword, indicating the end of the conditional procedure. A simple example is given below, indicating that A is less than B.

```
A = 1;    % Definition of an arbitrary number A.
B = 1.01; % Definition of an arbitrary number B.

if A > B
    disp('A is greater than B!')
elseif A < B
    disp('A is less than B!')
elseif A==B
    disp('A is equal to B!')
else
    disp('Error!')
end
```

6.2 for-loop Control Statements

`for` loop control statements are used when a specific number of loops is needed to be conducted, keeping each iteration with respect to a variable of an incrementing index. The loop operations can be characterized as slow. Hence, it is suggested to be replaced by vector operations, whenever this is possible to be implemented. The for-loop control statements are always start with the operation procedure and finishes with the keyword end. In-between, the main operated procedures are conducted. An example is given below, extracting `r = [2 4 ... 8 16 32]` and `c = [3;9;27;81;243]`

```
% Definition of a row vector.
x = [1 2 3 4 5];

% Zero padding the lengths of row and column vector, ensuring vectors have
% lengths equal to x vector.
r =zeros(1,length(x));
c =zeros(length(x),1);

for i = 1:length(x) % Start of loop (in total 5 iteration)
    % Storing the calculated values in a row vector.
    r(i) = 2^(x(i));
    % Storing the calculated values in a column vector.
    c(i,1) = 3^(x(i));
end % Ending of each iteration and loop
```

7 Plotting in MATLAB

MATLAB provides a high number of graphical illustrations. Here, the basic commands and parameters needed to be defined, constructing high quality figures, are summarized. There are different functions for plotting data in MATLAB. In the Table 8, some of the available plotting functions have been summarized with respect to the type of the plot.

Table 8: Plotting functions.

Type of Plot	Function	Description
Line	`plot(x,y)`	2-D line plot
	`plot3(x,y,z)`	3-D point or line plot
	`loglog(x,y)`	Log-log scale plot
	`semilogx(x,y)`	Semi-log plot (i.e., log scale: x-axis)
	`semilogy(x,y)`	Semi-log plot (i.e., log scale: y-axis)
	`errorbar(x,y,err)`	Line plot with error-bars
	`line([xmin,xmax],[ymin,ymax])`	Line plot with defined coordinates
Scatter	`scatter(x,y)`	2-D scatter plot
Data Distribution	`histogram(x)`	Histogram plot
	`histogram2(x,y)`	Bivariate histogram plot
	`pie(x)`	Pie chart
	`heatmap(tbl,x,y)`	Heatmap chart
	`plotmatrix(x,y)`	Scatter plot matrix
Discrete Data	`bar(x)`	Vertical bar graph
	`barh(x)`	Horizontal bar graph
	`stem(x,y)`	Discrete sequence data graph
Polar Plots	`polarplot(theta,rho)`	Plot line in polar coordinates
	`polarscatter(th,r)`	Scatter chart in polar coordinates
	`compass(u,v)`	Arrows emanating from origin
Contour	`contour(z)`	Contour plot of matrix
Surface/Mesh	`surf(x,y,z)`	Surface plot
	`mesh(x,y,z)`	Mesh surface plot

When visualizing different data, it is important to control their representation. Control functions ensure that all the information is included and well-observable, indicating high-quality of figures. The most important control functions are summarized in the Table 9.

Table 9: Control plot functions.

Controlling Function	Description
`title('text')`	Setting a title to the plot
`xlabel('text')`	Setting a label to the x-axis
`ylabel('text')`	Setting a label to the y-axis
`zlabel('text')`	Setting a label to the z-axis
`xlim([xmin,xmax])`	Setting the range of x-values in x-axis
`ylim([ymin,ymax])`	Setting the range of y-values in y-axis
`zlim([zmin,zmax])`	Setting the range of z-values in z-axis
`legend('text')`	Setting a legend to the plot
`set(H,Name,Value)`	Set graphics object properties

In addition, it is feasible to control further the colour of the lines, the style of the lines, as well as to define types of the markers. The following commands can be used when data is plotting with the plot command.

Table 10: Colour, line style, and marker commands.

Colour (`'color'`)		Line-style (`'linestyle'`)		Marker	
Symbol	Description	Symbol	Description	Symbol	Description
`'y'`	Yellow	`'-'`	Solid line	`'o'`	Circle
`'m'`	Magenta	`'--'`	Dashed line	`'+'`	Plus-sign
`'c'`	Cyan	`':'`	Dotted line	`'*'`	Asterisk
`'r'`	Red	`'-.'`	Dash-dot line	`'.'`	Point
`'g'`	Green			`'x'`	Cross
`'b'`	Blue			`'_'`	Horizontal line
`'w'`	White			`'d'`	Diamond
`'k'`	Black			`'s'`	Square

The figures are defined with the `figure(n)` command, where n is the counting number of the figure, before starting plotting the data. If the `figure` number is not specified, MATLAB provides a number to the plotted figure by itself. By using the hold on command, it is possible the visualization of multiple plots in the same figure.

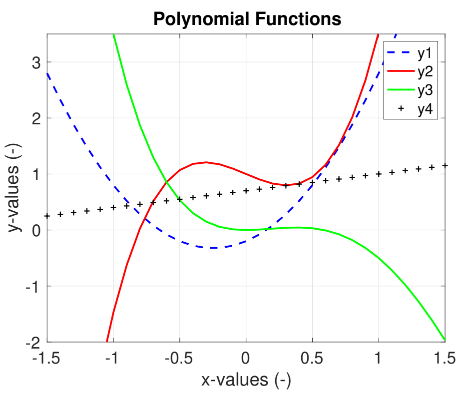
An example of plotting polynomial functions is presented below. Here, an arbitrary x-vector is defined, for plotting the polynomial functions. As it is expected, the vectors x and y in the `plot` functions are of the same length. In addition, in the command set, the `gca` keyword is used in order to set the font size to the current axes. Finally, the `grid on` command, enables the presence of grids in the plot.

```
% Definition of a row x-vector, taking the from -1.5 and to 1.5 with a step 0.1
x = -1.5:0.1:1.5;

% Definition of four polynomial functions (vectors)
y1 = 2*x.^2 + x - 0.2;
y2 = 3.5*x.^3 + 0.03*x.^2 - x + 1;
y3 = 0.5*x.^4 - 2*x.^3 + x.^2;
y4 = 0.3*x + 0.7;

% Definition of the figure number.
figure(1)

% Plotting the 1st polynomial with a 1.6 linewidth, a double-dashed line-style and a blue colour.
plot(x,y1,'linewidth',1.6,'color','b','linestyle','--')

% Holding the figure for plotting more data.
hold on

% Plotting the 2nd polynomial with a 1.6 linewidth, and red colour.
plot(x,y2,'linewidth',1.6,'color','r')

% Plotting the 3rd polynomial with a 1.6 linewidth, and a green colour.
plot(x,y3,'linewidth',1.6,'color','g')

% Plotting the 4th polynomial with a plus sign, and a colour of black.
plot(x,y4,'+','color','k')

% Definition of the legends.
legend('y1','y2','y3','y4')

% Definition of the title of the figure.
title('Polynomial Functions')

% Definition of the x and y axes labels of the figure.
xlabel('x-values (-)')
ylabel('y-values (-)')

% Definition of the limits of the x and the y axis.
xlim([-1.5,1.5])
ylim([-2,3.5])

% Definition of the font-size.
set(gca,'fontsize',18)

% Enabling of grid.
grid on
```

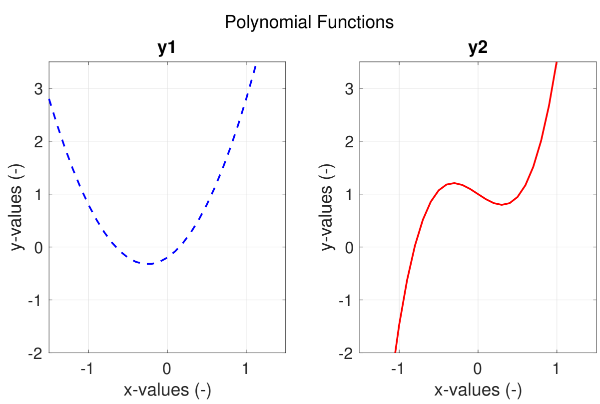
By using the `subplot(m,n,p)` function, the separation of plots to different blocks can be occurred. This function divides the current figure into an `m`-by-`n` grid, creating axes in the specified position `p`. The following example illustrates this procedure based on the two defined polynomials.

```
% Definition of a row x-vector
x = -1.5:0.1:1.5;

% Definition of polynomial functions
y1 = 2*x.^2 + x - 0.2;
y2 = 3.5*x.^3 + 0.03*x.^2 - x + 1;

% Plotting the polynomial functions
figure(1)

% Plotting the polynomial functions y1 to a 1-by-2 grid in the 1 position
subplot(1,2,1)
plot(x,y1,'linewidth',1.6,'color',...
'b','linestyle','--')
xlabel('x-values (-)')
ylabel('y-values (-)')
title('y1')
xlim([-1.5,1.5])
ylim([-2,3.5])
set(gca,'fontsize',15)
grid on

% Plotting the polynomial functions y1 to a 1-by-2 grid in the 2 position
subplot(1,2,2)
plot(x,y2,'linewidth',1.6,'color','r')
xlabel('x-values (-)')
ylabel('y-values (-)')
title('y2')
xlim([-1.5,1.5])
ylim([-2,3.5])
set(gca,'fontsize',15)
grid on

% Definition of a global title
GlT = sgtitle('Polynomial Functions');
GlT.FontSize = 20;
```

Finally, it is needed to be mentioned that the editing of the plots can also be conducted in the figure pop-up window. However, it should be noted that the editor-based changes are not stored after quitting MATLAB. Hence, it is important to save it.



(a) plot example.



(b) subplot example.

Figure 3: Examples of plotting polynomials.

8 Exercises

In the first line of your scripts, include the commands clear all, close all, and clc. These commands, clear the workspace, close the open MATLAB windows, and clear the command window, respectively. These commands are very useful, when programming, since previous stored data and variables are deleted.

When scripts are written, it is important to provide comments in your scripts, full descriptions and comments into the user-based functions, assign all the important information to the figures as well as the figures should be of high-quality.

Exercise 1: (Numerical Operations)

Firstly, assign the following variables into your script, and then answer the following questions and tasks.

$$r = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, c = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}, M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What are the dimensions of the vectors and matrix?

Perform the following numerical operations and assign the results to new variables.

$$c + r^T, rM, Mc, M^T M.$$

Perform the following numerical operations as scalar and element-by-element product and assign the results to new variables.

$$rc$$

What are the dimensions of the new vectors and matrices?

Exercise 2: (if-Statement and Function)

Create a function that the user will provide its grade in percentage, returning its grade to a letter format. For this purpose, the following equalities are assumed.When the user provides wrong numerical value, an error message is needed to be displayed. Note: Do not assign an output to your function!

Letter Grade	Percent Grade
E/F	<65
D	65-69
C	70-79
B	80-89
A	90-100

Exercise 3: (For-loop and Plotting)

Assign the following polynomial functions,

$$y_1 = x^5 - 8x^3 + 10x + 6 \tag{1}$$

$$y_2 = x^4 - 8x^2. \tag{2}$$

Calculate the vectors $y_1$and y_2 from -2.5 to 2.5, with a step of 0.01, using a for loop.

Replace the operational procedure with vector operations.

Plot the polynomial functions together into the same figure.

Find the minimum and maximum values of the polynomial y_1.

Assign the maximum and minimum values into your figure, using markers.

Find the cross-point of the polynomials y_1 and y_2.

Assign the cross-point into your figure, using a marker.

Exercise 4: (Function and Plotting)

Create a function in MATLAB, calculating the harmonic motion of a simple sinusoid, given by the expression,

$$y = A \sin(\omega t + \phi) \tag{3}$$

where,

A	:	Amplitude of the harmonic motion.
$\omega = 2\pi f$	:	Angular frequency (rad/s).
t	:	Time vector (s).
ϕ	:	Phase of the harmonic motion (rad).

By calling the function, the user will provide the amplitude A, the frequency f and the phase ϕ, and the function will return the vectors x and t. Hint: For matter of simplicity, use the `linspace` function for the definition of the time vector t, assuming a total duration of 1 second with 100 equal space samples in between. Plot two harmonic motions as well as their addition and subtraction product in separate blocks. Hint: Use the subplot function.

Exercise 5: (Audio and Plotting)

Import the two audio files Audio1 and Audio2 in MATLAB [Use the `audioread` function]. After importing the files, plot the signals over time in the same figure. Hint: For the calculation of the time vector use the expression `t=0:1/fs:(T-1)/fs`, where `T` is the length of the signal. After plotting the audio files, listen to the audio files [Use the `sound(y,fs)` function].