

Matlab Programming Laboratory

GNG1103 – Engineering Design

Objective

To learn the basics of the MATLAB software environment and be able to solve mathematical problems. To utilize MATLAB to graphically visualize data and solve multi-variable problems.

Background

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include: Math, computation, algorithm development, modeling, simulation, prototyping, data analysis, exploration, visualization, scientific and engineering graphics, application development, including Graphical User Interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software. MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

The MATLAB Language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

References

You can find more information about MATLAB in MATLAB official website:
https://www.mathworks.com/help/matlab/learn_matlab/plots.html

Pre-lab

Before arriving to the lab, students should review the lab manual and become familiar with the lab setup and procedures. It is **highly** recommended to read the first part of Appendix I to understand the basics of MATLAB.

Pre-lab Questions

What does MATLAB stand for?

What are the capabilities of MATLAB?

What is the basic mathematical element that MATLAB uses?

What is the window called where you type commands?

How do you clear the workspace?

IMPORTANT: During this lab you will be taking screenshots for each part. Paste these screenshots in a Word document (or other) as you go through this lab manual so that you can make a PDF document of all your screenshots of your work. This PDF document of your screenshots will be submitted as a deliverable for this lab.

PART A. General Concepts

Read the following parts in Appendix I:

- Understanding MATLAB interface
- Working with MATLAB variables
- Understanding data structures

Exercises:

- 1) Write three matrixes X, Y and Z. X is a 3x3 matrix, Y is a 3x4 matrix and Z is a 4x3 matrix. Print them in the command Window screen.
- 2) Take a screenshot of the Command Window. You'll use it in your lab report.
- 3) Clear the Command Window.
- 4) Write the following code:

$x = 1:2:30$
 $y = 1:3:30$
 $z = 1:4:30$

Explain what happens in each case

- 5) By using the example illustrated below in appendix, create a matrix that contains a **name**, an **address** and a **phone number**. Print them on the command Window screen all at once.
- 6) Take a screenshot of your Command Window. You'll use it in your lab report.
- 7) Clear the Command Window.

PART B. Core MATLAB Syntax

Read the following parts:

- Basic commands
- Using built-in functions and variables
- Working with matrix and scalar operations
- Control flow

Exercises:

- 1) Considering three matrixes A = 'University', B = 'Of' and C = 'Ottawa'. Find a way to print in the command window the text: "**University Of Ottawa**". Think back to the Excel lab where you needed to combine a first and last name.
- 2) Take a screenshot of the Command Window. You'll use it in your lab report.
- 3) Clear the Command Window.
- 4) Write two other matrixes Y and Z where:

$$Y = \begin{bmatrix} 2 & 1 & 4 \\ 3 & 2 & 1 \\ 1 & 4 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 2 & 3 & 6 \\ 1 & 5 & 7 \\ 3 & 4 & 1 \end{bmatrix}$$

- 5) Using MATLAB, perform the following operations:

$$T = Y+Z$$
$$U = Y-Z$$
$$V = Y*Z$$

$$W = Y.*Z$$

- 6) Take a screenshot of the Command Window. You'll use it in your lab report.
- 7) What is the difference between V and W?

- 8) What is the value in matrix T corresponding to row 3 column 2? Write your answer, as well as the MATLAB syntax used to access this index.

- 9) What does typing $W(:,2)$ into the Command Window and hitting enter give you? Explain what the colon does when accessing indices.

PART C. Graph Plotting

Read the following parts:

- Creating basic plots
- Adding annotations

- 1) Open a new script.
- 2) Write the following code:

```
x = -20:2:20;  
y = (x.^2);  
plot(x,y)
```

- 3) Run it and observe the graph.
- 4) Add some information to your graph: x axis, y axis and a title.
- 5) Take a screenshot of the graph. You'll use it in your lab report.
- 6) Write the following code:

```
x = 0:0.1:20  
y1 = sin(x)  
y2 = cos(x)  
subplot(211)  
plot(x,y1)
```

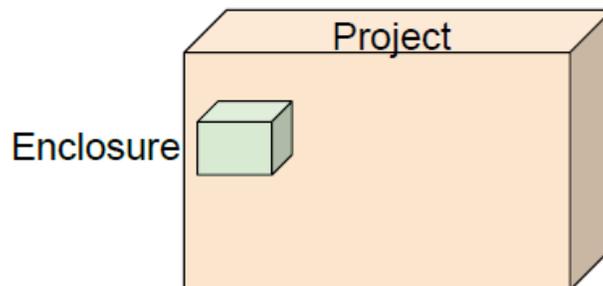
```
xlabel('x')
ylabel('y1')
title('Sine function')
subplot(212)
plot(x,y2)
xlabel('x')
ylabel('y2')
title('Cosine function')
```

- 7) Run the code.
 - 8) Replace subplot(211) by subplot(121) and subplot(212) by subplot(122)
 - 9) Take a screenshot of the graphs.
Explain what happens?
-
-
-

PART D. Enclosure calculation

Many projects will need an enclosure for electronics or other components, we will go through the process of calculating which material and which fastening method is best. Let's take the case where this box is mounted to the side of your project like in the figure below.

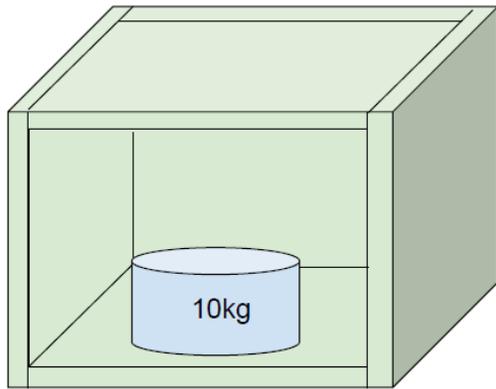
***Note:** Make sure that your units make sense and are consistent!



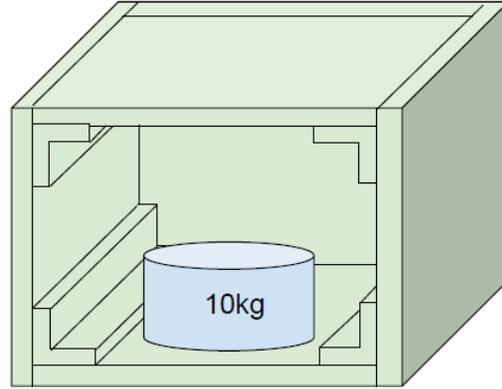
The available materials are listed here with some of their properties:

- MDF: thickness=0.003m, $\rho=750 \text{ kg/m}^3$, $E=4 \text{ GPa}$
- Acrylic: $t=0.003\text{m}$, $\rho=1\ 190 \text{ kg/m}^3$, $E=3.2 \text{ GPa}$
- Aluminium: $t=0.001\text{m}$, $\rho=2\ 700 \text{ kg/m}^3$, $E=69 \text{ GPa}$
- Stainless steel: $t=0.001\text{m}$, $\rho=7\ 820 \text{ kg/m}^3$, $E=180 \text{ GPa}$

Here are 2 ways this box can be assembled depending on fastening method:



Glue or welding



Glue or bolts

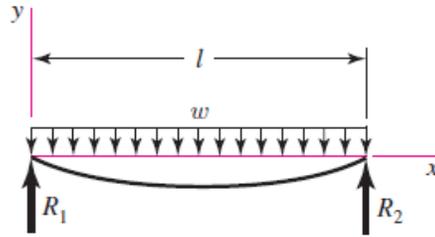
The fastening methods include:

- Acrylic glue: $\tau_{\text{allowable}}=13.8$ MPa based on room temperature
- Gorilla glue (polyurethane): $\tau_{\text{allowable}}=6.9$ MPa based on room temperature
- Welding: $\tau_{\text{allowable}}=79$ MPa based on a shear load on a fillet weld with bare electrodes
- Bolts: $S_p=310$ MPa for low/medium carbon steel bolts

You are calculating the amount of deflection (δ) on the bottom plate because of the weight (components + box itself) and calculating whether the strength of the joints will be enough to hold the weight.

- 1) Open a new script.
 - 2) Write a 4x3 matrix M with the proper information for all materials.
 - 3) Calculate the volume of each part of the box assuming the dimensions are: $0.1 \times 0.1 \times 0.1$ m and that each panel is the same size (i.e. 6 panels for 1 box). Neglect the L brackets. You should obtain two different values in cubic metres (for the two wall thicknesses), $V1$ for a 1mm and $V3$ for a 3mm wall thickness.
 - 4) Calculate the weight of the box for each material in a matrix W . Don't forget to add the mass inside the box ($W=g*\text{mass}$, $g=9.81$ m/s²).
*Ex: $W(1) = g*V3*M(1,2) + g*\text{mass}$;*
What does each part of this equation represent?
-
-

- 5) Calculate the moment of inertia ($I1$ and $I3$) for the bottom plate. $I = (\text{width}*\text{thick}^3)/12$
- 6) Assume the total weight of the mass and the box is equally distributed and the maximum deflection is in the centre of the plate.



$$\delta_{max} = \frac{5wl^4}{384EI}$$

Where w is the distributed weight ($w=W/l$) and E is the modulus of elasticity in Pa. Calculate the deflection δ (or D) for all materials, you should have $D= [1 \times 4]$ matrix with values in m.

Ex: $D(1)=5(W(1)/l)*(l^4)/(384*M(1,3)*(10^6)*I3);$

- 7) Take a screenshot of your results.
- 8) Which material deflects the most under these conditions? How does this differ from what you expected?

The connections of the bottom plate are in shear and you want to find the amount of force they can withstand and compare against the amount of force (weight) applied on them. Choose a design margin factor, n_d (n_d is usually set to 2 or 3, corresponding to two or three times more than the bare minimum requirements).

Glue: It can be expressed with $\tau_{allowable} = \frac{F}{A} n_d \therefore F = \frac{\tau_{allowable} A}{n_d}$ where A is the area of the lap joint (surfaces that are in contact) and τ the lap shear strength. Measure A twice, with and without the L brackets.

Welding: It can be expressed with $F_{allowable} = \tau_{allowable} \frac{hL}{\sqrt{2}n_d}$ where h is the width and L the length of the weld. Assume anything for the length, but justify your choice, and $h=10$ mm.

Bolts: It can be expressed with $\tau = \frac{F}{X\pi d^2/4} = 0.577 \frac{S_p}{n_d} \therefore F = 0.577 X \pi d^2 \frac{S_p}{4n_d}$ where X is the number of bolts and S_p is the minimum strength. Assume the bolt is 0.002 m in diameter and pick a specific number of bolts, justifying your selection, as required.

- 9) Calculate the maximum force that the joints can hold, according to each of these 3 methods.
- 10) Take a screenshot of your results.
- 11) Compare your results with the weight that is being applied on these joints. Which type of joint is good or the best in this situation?

PART E. Battery calculation

Many systems will also require batteries to power them. When choosing the type of battery, it is important to calculate the total system power requirements. In this step, as a more realistic exercise, you will be searching for your own formulas for the different calculations.

Assume that you need a stepper motor, which is powering a mechanism that needs to be provided with 8 kg-cm of torque.

Here is a list of possible motors:

1. Unipolar stepper motor (<https://www.robotshop.com/ca/en/rbsoy03-soyo-unipolar-stepper-motor.html>)
2. Nema 17 bipolar stepper motor (<https://www.robotshop.com/ca/en/12v-17a-667oz-in-nema-17-bipolar-stepper-motor.html>)
3. Unipolar stepper motor Nema 23 (<https://www.robotshop.com/ca/en/12v-068a-125oz-in-unipolar-stepper-motor-nema-23.html>)

What is the torque capability of each motor?

Which motor(s) are suitable for your needs?

Choose the best motor for this design. What is the voltage, current consumption, resistance and number of phases for this motor?

Using MATLAB, your task is to calculate the power, current and lifetime for a battery to power this system properly and with sufficient margin.

- 1) Open a new script.
- 2) Calculate the power used from the stepper motor (watts). Don't forget to include the different number of phases (or coils) as appropriate for your choice of motor.
- 3) Calculate the current needed from a battery with a voltage that is appropriate for the motor that you selected.
- 4) Assuming that your system needs to run for 2 hours every day, how many mAh does it need per day?

- 5) Find a battery with the right voltage and life (mAh). How many years or months will it last before the battery needs to be replaced or recharged, depending on the battery technology you selected. Don't forget to use the appropriate design margin factors (e.g. to handle temperature variations, battery voltage variation, finite wire resistance, etc.).?
- 6) Take a screenshot including your calculations and your chosen battery.

SUBMISSION

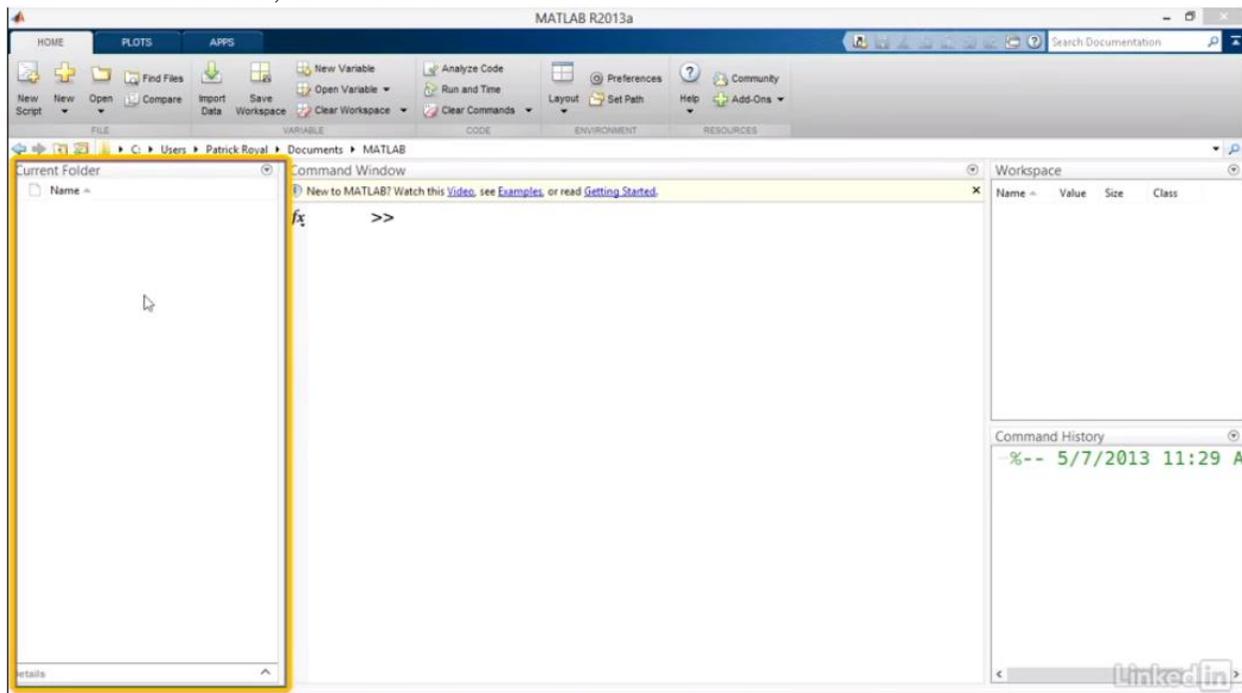
Submit this lab manual as a printed PDF with the answers to the questions on the lines in this document by uploading to Virtual Campus before the due date. You must also include the PDF document with your screenshots in your submission. All uploaded files must be submitted as one submission with many uploads, not many individual submissions.

Appendix I: MATLAB documentation

This document contains all the information necessary to answer the questions of the laboratory.

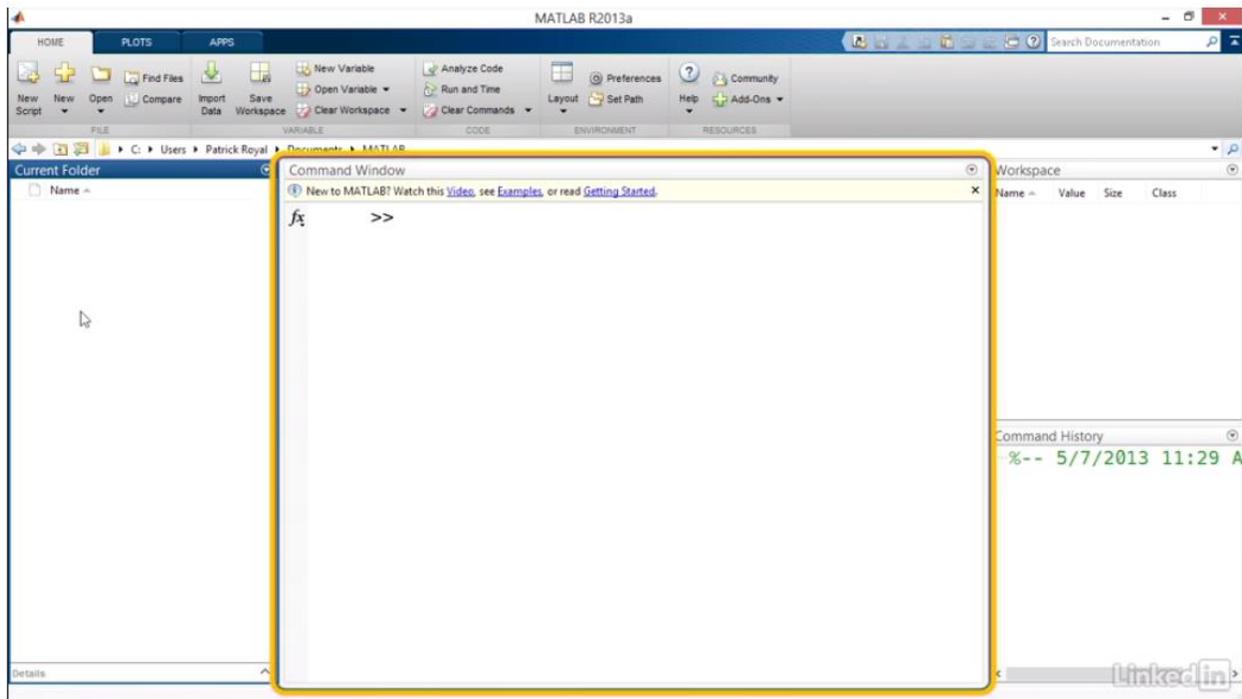
Understanding the MATLAB interface

When you open MATLAB for the first time, you will see that the interface is divided into four windows. On the left, the current folder window.



This is where all relevant MATLAB files are placed in the folder where you are. But it's also where all the functions and scripts you create will be stored. It is important to ensure that you are in the right folder when you execute scripts and functions because MATLAB will consider the active folder for the purpose of performing functions. By default, MATLAB will define the current folder in a folder named MATLAB in your Documents folder.

In the middle of the screen is the command window.

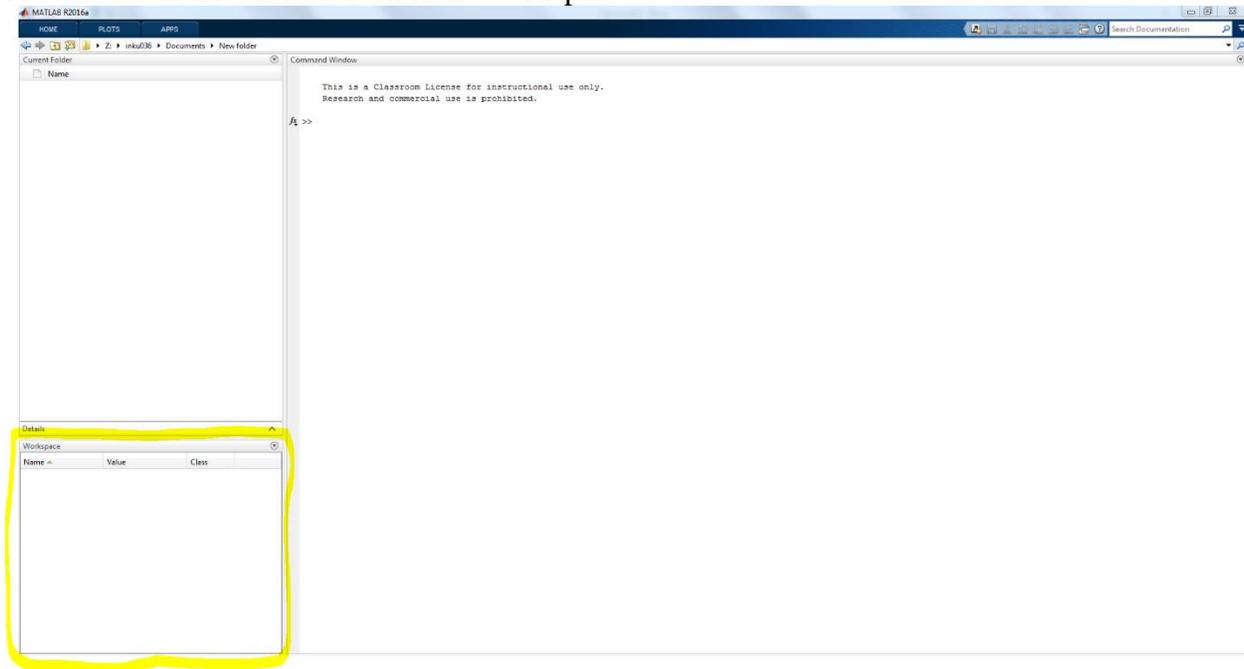


This area contains all your inputs and outputs when you perform various functions. Whenever you execute a script or function, any output will be displayed here. Technically, the command window has all the same functions as the script editor. Thus, it is possible to write programs directly on this line. However, in practice, you will probably want something that you can record, edit, and run multiple times.

At any time, if the command window becomes too cluttered, you can delete it by typing `clc` and pressing Enter, or by clicking the arrow in the upper-right corner of the tile and choosing Clear Command Window.



At the bottom left of the screen is the workspace.



It contains the list of all the active variables in your simulation. To modify the information available on your variables, right-click on the columns and select the information you want to display.

The workspace displays all the variables you used in a given MATLAB session, regardless of the script or function they came from. So, after a while, it can be cluttered with old data. To clean the workspace, click the drop-down arrow at the top right of the Workspace screen and choose Clear Workspace. After confirming, MATLAB will delete all existing variables.

Working with MATLAB variables

Let's take a look at how to create and manipulate basic MATLAB variables. For starters, we are going to need some variables. For now, we will use the simplest method of creating variables, by typing in the command window. Defining a new variable in MATLAB is very easy. Unlike other programming languages like C or Java, you do not need to add a variable declaration statement. In other words, you do not need to tell MATLAB anything like, this variable is an integer, or this variable is a string. Instead, all you need to do is tell MATLAB what your variable is equal to, so you can do it now.

In the command window, type A equal to 1 and press Enter. The command window (Command Window) makes you the result A is equal to 1. The workspace now displays a new variable called A with a value of 1 and a size of 1 by 1. That says just that A is a scalar.

MATLAB also supports variables that are vectors or matrices. To create a vector in your variable declaration, place brackets around your term and separate each value in square brackets with a comma or space. For example, $C = [1\ 2\ 3]$ causes C to be created as a vector one by three with the three values stored there.

```
Command Window

This is a Classroom License for instructional use only.
Research and commercial use is prohibited.

>> C = [1 2 3]

C =

     1     2     3
```

To create a matrix, the same principle applies, except that you use a semicolon to separate the lines. For example, $D = [1,2; 3,4]$ causes D to be created as a 2 by 2 matrix with these values.

```
Command Window

>> D = [1,2;3,4]

D =

     1     2
     3     4

fx >> |
```

Understanding data structures

Structures are built into the MATLAB storage mechanism that allows you to associate multiple pieces of data together. For example, you might have a list of names and addresses, and you might want to match each name to its address. To create a structure and add data to it, just define each data field with a dot. In our example, we will say something like $s.name = John\ Doe$. And then $s.address = 123\ Fake\ Street$.

To add additional data, you can add data for $s(2)$, $s(3)$, and so on. For example, we could say $s(2).name = Jane\ Doe$ and $s(2).address = 124\ Fake\ street$. As usual, this is considered a matrix, so your structure can have any number of dimensions and may even contain substructures. Access to data works the same way. Entering $s(1,1).name$ will return only the name of the first data point, while $s(1,1)$ without qualifier will return the entire entry. If you enter a name with a row or column number, you will return all the names of all the fields.

Basic commands

To set variables as text rather than a numeric value, place single quotation marks around the text that you define. If you want your string to be variable rather than fixed, the functions *num2str* and *strcat* will be useful. *Num2str* converts a numeric value into a string, which adds a non-predetermined number to the string. *Strcat* concatenates two strings strictly to form a longer string.

```
>> a = 'Hello'

a =

Hello

>> b = 'World'

b =

World

>> c = strcat(a,b)

c =

HelloWorld
```

Unlike Java, MATLAB does not require you to place a semicolon at the end of each line. If you do, MATLAB will interpret it as a command to suppress the output for everything that happens on the previous line. This is an extremely useful feature, because by default, MATLAB displays the results of each function, equation, variable definition, or loop during the program. For larger scripts, this can quickly overwhelm the command window and make the program unusable.

In general, unless you specifically want to see the output of a line, it is good practice to end each line with a semicolon.

```
>> a = 'Hello';
>> b = 'World';
>> c = strcat(a,b);
>> % If we want to see the value of a we just write :
>> c

c =

HelloWorld
```

Using built-in functions and variables

Basically, a function differs from a script because it has input channels and fixed output channels. This makes the functions extremely useful when you are writing complicated programs that can repeat the same calculations over and over again. Rather than rewrite your code, you can simply rewrite that part of it in a separate function. MATLAB comes with a wide variety of functions already implemented. So, let's start by looking at how to use them. The most commonly used functions are matrix generation functions that are used to create a matrix with some startup data.

As an example, write `a = ones (2,3)` in the command window. This tells MATLAB to generate a new matrix with two rows and three columns, then set all the values in that matrix equal to 1. The **zeros** function does the same thing with zeros, the **rand** function does the same thing with numbers random and so on. The important thing to note here is that all these functions use the same syntax. Type the name of the parenthesized tracking function that contains each of the function entries separated by commas.

```
Command Window
>> a = ones(2,3)

a =

     1     1     1
     1     1     1

>> zeros(2,3)

ans =

     0     0     0
     0     0     0

>> rand(2,3)

ans =

     0.8147     0.1270     0.6324
     0.9058     0.9134     0.0975
```

Working with matrix and scalar operations

Let's take a look at the basic matrix operations used by MATLAB. Since MATLAB treats all variables as if it were a matrix, it is important to distinguish between operations such as the multiplication of the matrix with respect to multiplication of the scalar. We will practice different types of matrix operations.

Let's use the matrices below:

```
Command Window
>> a = [1,2,3;5,6,7]

a =

     1     2     3
     5     6     7

>> b = [1,1,1;0,4,5]

b =

     1     1     1
     0     4     5

>> c = [3,1;2,2;0,0]

c =

     3     1
     2     2
     0     0

>> d = [4,4,2;-1,2,4]

d =

     4     4     2
    -1     2     4

>> e = 1:4

e =

     1     2     3     4

>> f = [2,1,4,3]

f =

     2     1     4     3
```

To begin, type `a + b` in the command window. Since the matrix `a` and the matrix `b` have the same dimension, MATLAB will interpret it as piecewise addition and simply add the corresponding values in each cell of the matrices.

```
Command Window
>> a+b

ans =

     2     3     4
     5    10    12
```

On the other hand, if you type $a + 2$, MATLAB will interpret this as a scalar addition, so that it will add two to each cell in A.

```
Command Window
>> a+2

ans =

     3     4     5
     7     8     9
```

The same principle applies to the multiplication of matrices. If we enter $c * d$, MATLAB sees that these two variables are matrices, and their internal dimensions correspond (the number of columns of c is equal to the number of lines of d). So, it will automatically perform the matrix multiplication.

```
Command Window
>> c*d

ans =

    11    14    10
     6    12    12
     0     0     0
```

On the other hand if you type $c * 2$ MATLAB considers 2 as a constant and multiplies each term in c by 2.

```
Command Window
>> c*2

ans =

     6     2
     4     4
     0     0
```

Matrix multiplication is a fairly straightforward process. But what if we wanted MATLAB to perform a piecewise multiplication of the corresponding matrix entries. Matrixes a and b have the same dimensions, but if you enter just once b, MATLAB does not know that we want to multiply the corresponding entries. So, it assumes that we want to multiply the matrices and display an error because the inner dimensions of these two matrices do not match. Instead, we can add a dot in front of the asterisk.

```
Command Window
>> a*b
Error using *
Inner matrix dimensions must agree.

>> a.*b

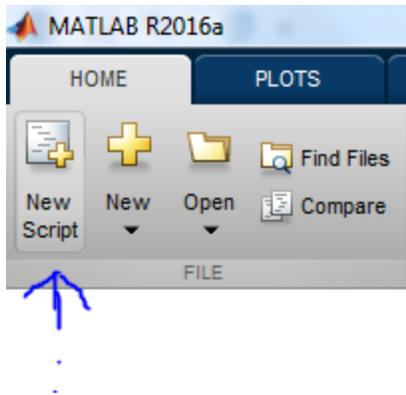
ans =

     1     2     3
     0    24    35
```

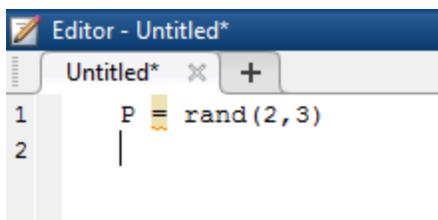
Adding a point in front of any operation, addition, subtraction, multiplication, division, exponents or even equality, tells MATLAB to execute the operation in pieces. This is a really effective way to handle large amounts of data at a time.

Control flow

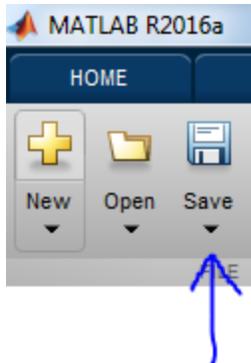
Creating a new program is easy. Just click the New Script button to generate the script and automatically open the Script Editor window.



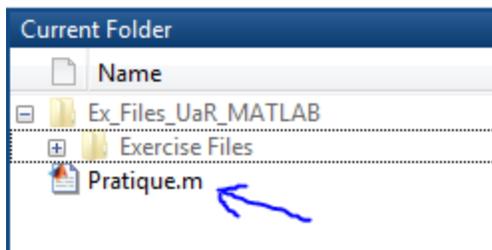
Here you can type any code that will be executed as part of the program. For example, we will simply create a very simple script that generates a two by three random matrix. So $P = \text{rand}(2,3)$.



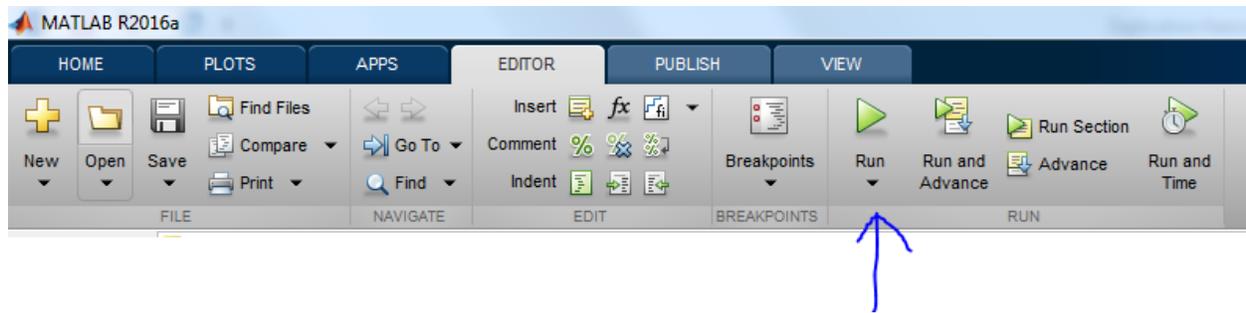
When you are ready to save, click the Save button and give the program a name and it will automatically save to the active folder.



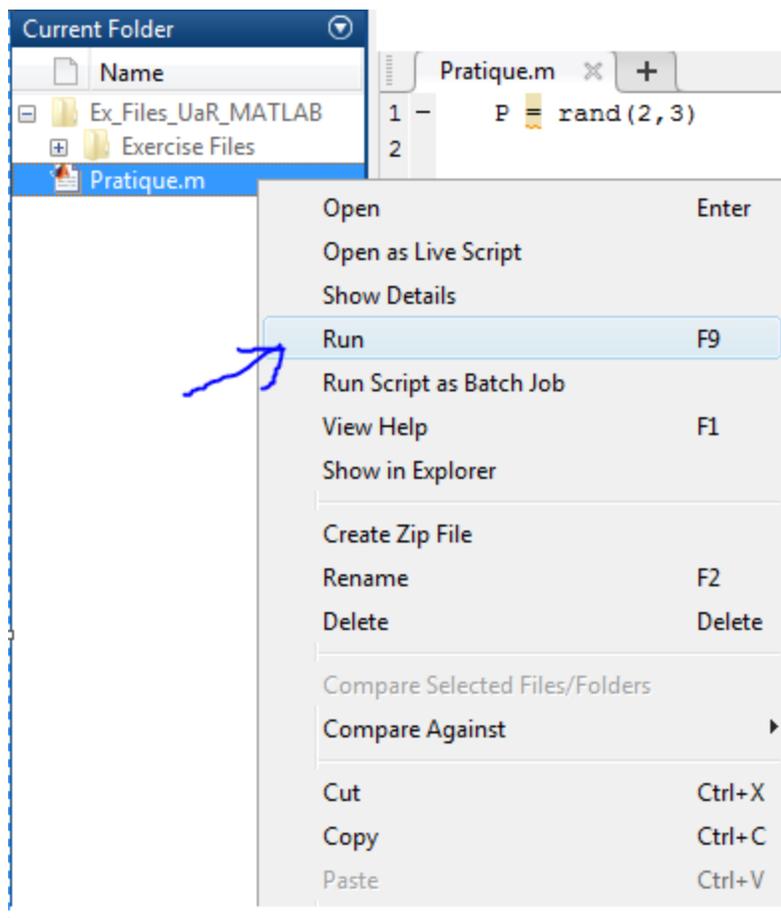
Now, every time you want to go back to the script to edit it, just double-click on the name in the current folder pane and this will reopen the window and allow editing the script exactly as before.



To run the program, there are several different options. You can first click the Run button in the edit window. This runs the program immediately without closing the window, allowing you to quickly and easily see how changes made to the program affect the output.



Second, you can right-click on the script from the current folder window. And choose Run or press F9.



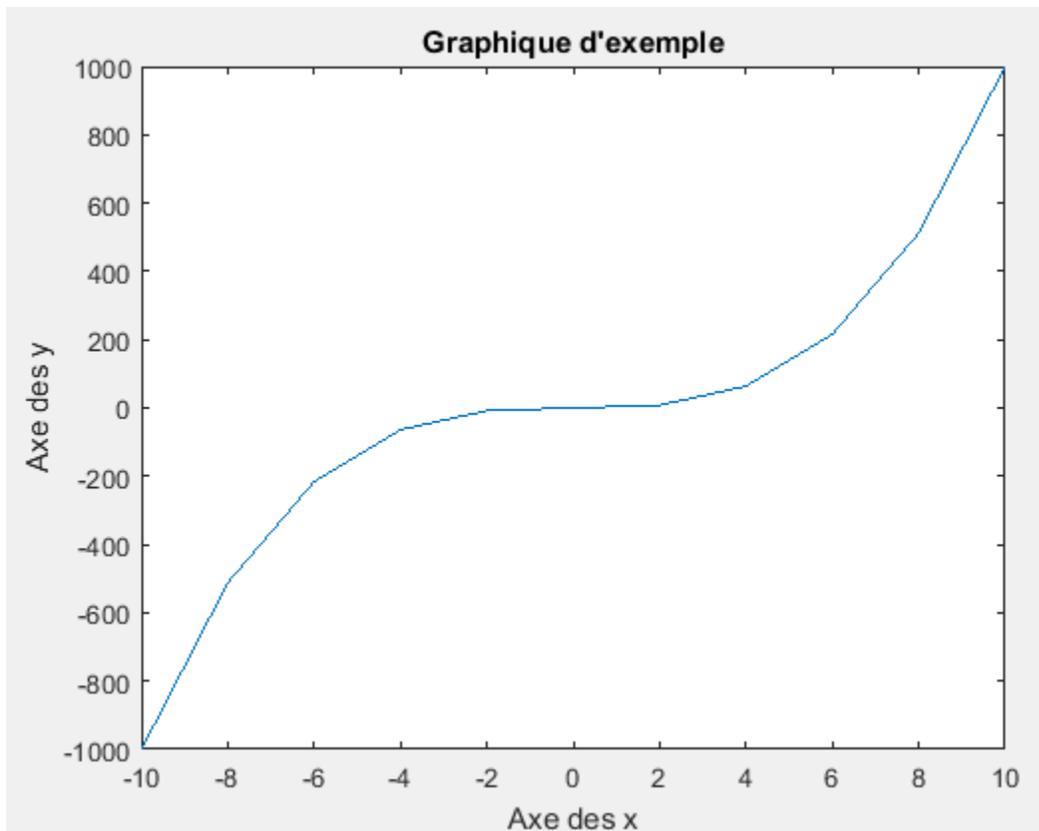
Creating basic plots

Let's look at ways to display data in MATLAB. The basic function for drawing a two-dimensional graph in MATLAB is the `plot` function. There is a wide variety of syntax for this function, allowing you to change many of the parameters as you create it. The simplest plot takes a single data vector and plots the data values against the number of indexes of the data in the vector: `plot(x, y)`

Adding annotations

To add information on a graph like the title, the axes, ... just type the functions `xlabel`, `ylabel` or `title` followed by the information to enter.

```
x = -10:2:10;  
y = (x.^3);  
plot(x, y)  
xlabel('Axe des x');  
ylabel('Axe des y');  
title('Graphique d''exemple');
```



Appendix II: Introduction to matrices

A **matrix** is a rectangular arrangement of numbers into rows and columns. For example, matrix A has two **rows** and three **columns**.

$$A = \begin{bmatrix} -2 & 5 & 6 \\ 5 & 2 & 7 \end{bmatrix}$$

3 columns

↓ ↓ ↓

← ← 2 rows

Matrix dimensions

The dimensions of a matrix tells its size: the number of rows and columns of the matrix, *in that order*.

Since Matrix A has two rows and three columns, we write his dimension as 2x3.

In contrast, matrix B has 3 rows and 2 columns so it's a 3x2 matrix.

$$B = \begin{bmatrix} -8 & -4 \\ 23 & 12 \\ 18 & 10 \end{bmatrix}$$

Matrix elements

A matrix element is simply a matrix entry. Each element in a matrix is identified by naming the row and column in which it appears.

For example, consider matrix G:

$$G = \begin{bmatrix} 4 & 14 & -7 \\ 18 & 5 & 13 \\ -20 & 4 & 22 \end{bmatrix}$$

The element $G_{3,1}$ is the entry in the 3 row and first column which is **-20**

Matrix Operations

- Addition

To add two matrices of the **same dimensions**, simply add the entries in the corresponding positions.

Remember, the matrices need to have the same dimension, which means the same number or rows and columns.

Example:

$$\begin{bmatrix} 3 & 7 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 2 \\ 8 & 1 \end{bmatrix} = \begin{bmatrix} 3+5 & 7+2 \\ 2+8 & 4+1 \end{bmatrix} \\ = \begin{bmatrix} 8 & 9 \\ 10 & 5 \end{bmatrix}$$

- Multiplication

a) Scalar by matrix

The term scalar multiplication refers to the product of a real number and a matrix. In scalar multiplication, each entry in the matrix is multiplied by the given scalar.

$$2 \cdot \begin{bmatrix} 5 & 2 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 2 \cdot 5 & 2 \cdot 2 \\ 2 \cdot 3 & 2 \cdot 1 \end{bmatrix} \\ = \begin{bmatrix} 10 & 4 \\ 6 & 2 \end{bmatrix}$$

b) Matrix Multiplication (i.e. product of two matrices)

Matrix multiplication refers to the product of two matrices, which is not the same as multiplying a matrix with a scalar number. You can multiply two matrices if, and only if, the number of columns in the first matrix equals the number of rows in the second matrix. Otherwise, the product of two matrices is undefined.

The product matrix's dimensions are: (rows of first matrix) \times (columns of the second matrix)

As examples, we can multiply a 2×3 matrix with a 3×1 matrix and the resulting product matrix will be 2×1 . Similarly, we can multiply a 3×4 matrix with a 4×2 matrix, to get a 3×2 matrix. However, we cannot multiply a 3×4 matrix and a 2×4 matrix together at all.

$$\begin{array}{c}
 \mathbf{2 \times 3} \\
 \left| \begin{array}{ccc}
 \mathbf{r11} & \mathbf{r12} & \mathbf{r13} \\
 \mathbf{r21} & \mathbf{r22} & \mathbf{r23}
 \end{array} \right|
 \end{array}
 \times
 \begin{array}{c}
 \mathbf{3 \times 1} \\
 \left| \begin{array}{c}
 \mathbf{t11} \\
 \mathbf{t21} \\
 \mathbf{t31}
 \end{array} \right|
 \end{array}
 =
 \begin{array}{c}
 \mathbf{2 \times 1} \\
 \left| \begin{array}{c}
 \mathbf{M11} \\
 \mathbf{M21}
 \end{array} \right|
 \end{array}$$

Here is how we get M_{11} and M_{12} in the product. These individual terms are each called dot products. A dot product is a single scalar number. This is not the same as the matrix product, which can be a matrix, itself, as mentioned.

$$M_{11} = r_{11} \times t_{11} + r_{12} \times t_{21} + r_{13} \times t_{31}$$

$$M_{21} = r_{21} \times t_{11} + r_{22} \times t_{21} + r_{23} \times t_{31}$$