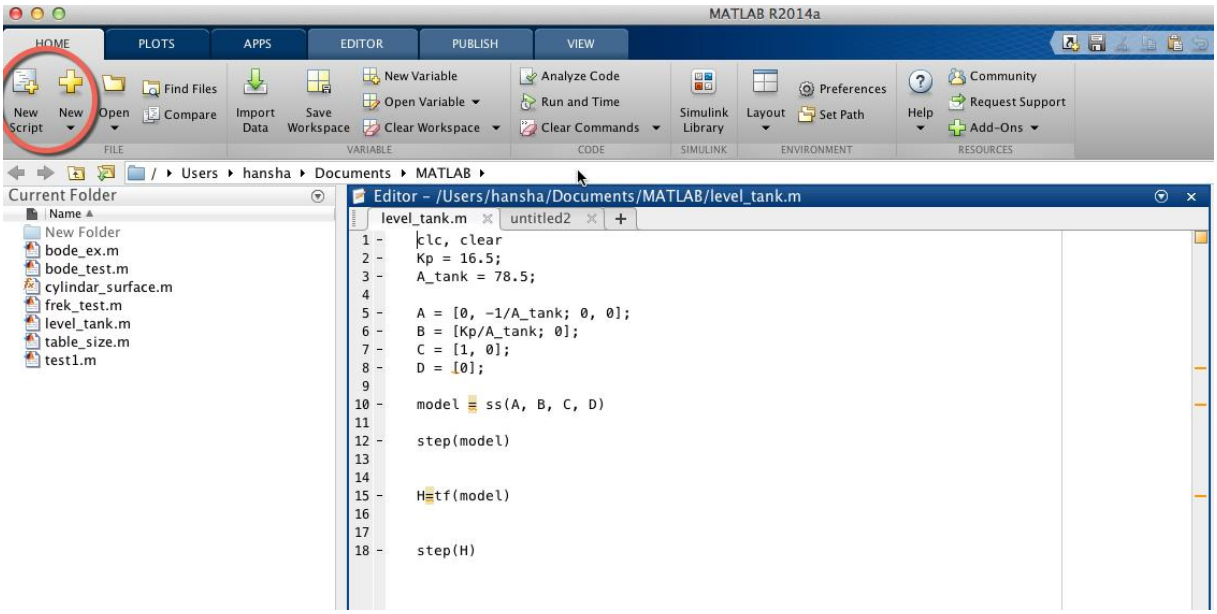
Chapter 4: Programming in MATLAB

Scripts or m-files are text files containing MATLAB code. Use the MATLAB Editor or another text editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends with ".m".

We can either create a **Script** or a **Function**. The difference between a script and a function will be explained below. Both will be saved as m-files, but the usage will be slightly different.

Below we see the **MATLAB Editor** that we use to create Scripts and Functions (both are saved as .m files):



Scripts vs. function Files

It is important to know the difference between a Script and a Function.

Scripts:

- A collection of commands that you would execute in the Command Window

- Used for automating repetitive tasks

Functions:

- Operate on information (inputs) fed into them and return outputs
- Have a separate workspace and internal variables that is only valid inside the function
- Your own user-defined functions work the same way as the built-in functions you use all the time, such as plot(), rand(), mean(), std(), etc.

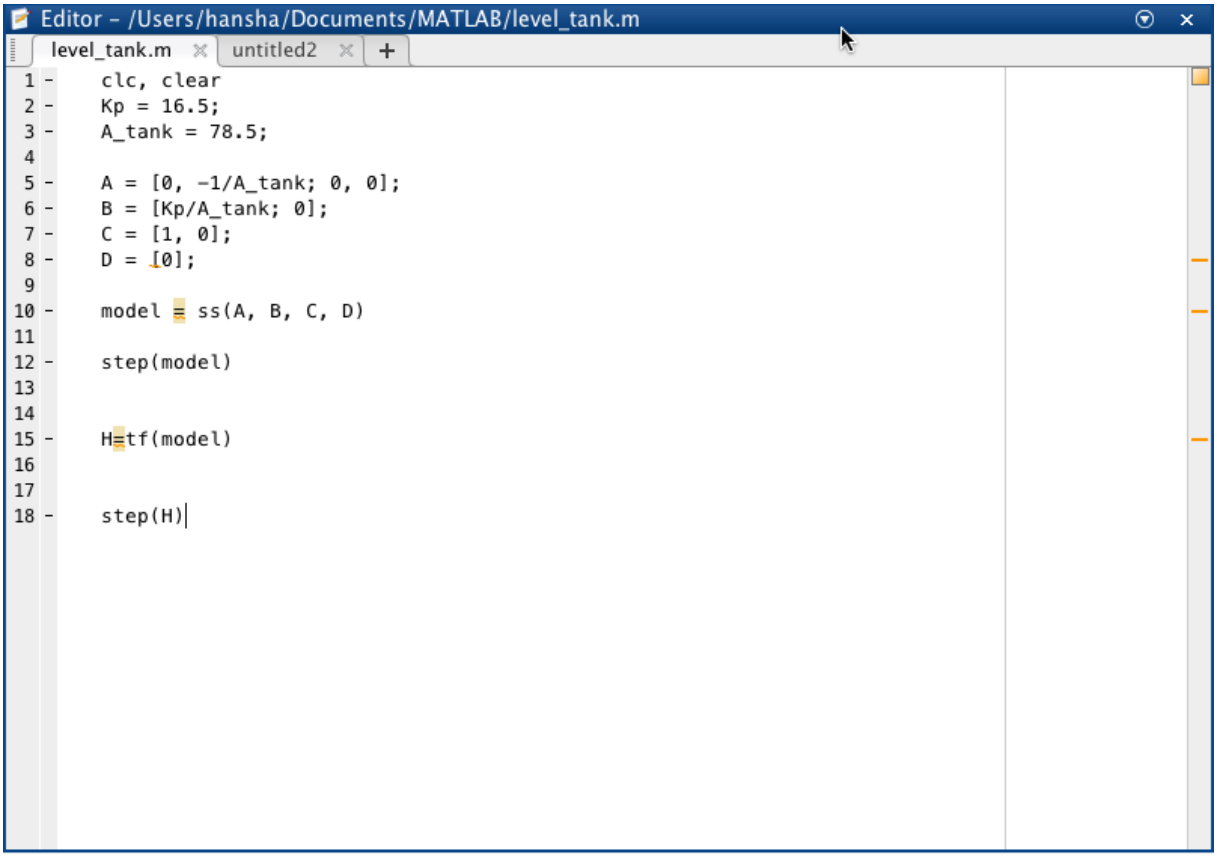
MATLAB has lots of built-in functions, but very often we need to create our own functions (these are called user-defined functions)

Below we will learn more about Scripts and Functions.

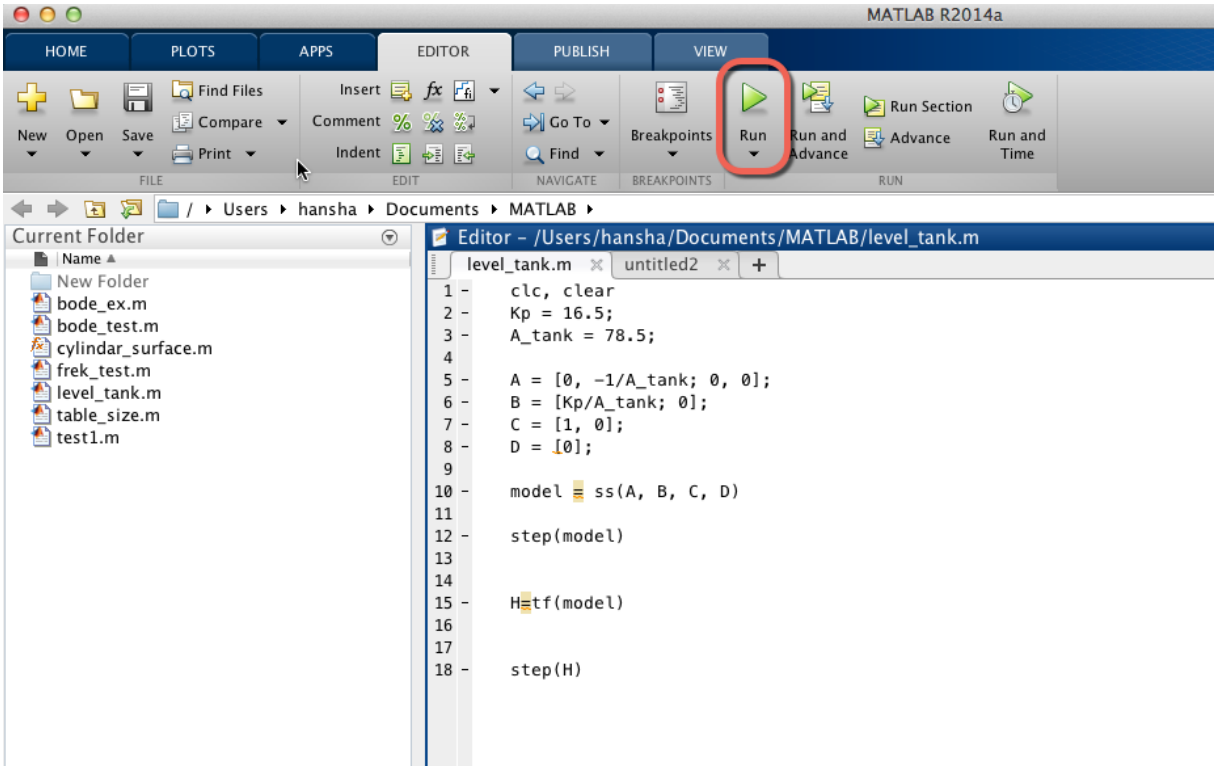
Scripts

A Script is a collection of MATLAB commands and functions that is bundled together in a m-file. When you run the Script, all the commands are executed sequentially.

The built-in **Editor** for creating and modifying m-files are shown below:

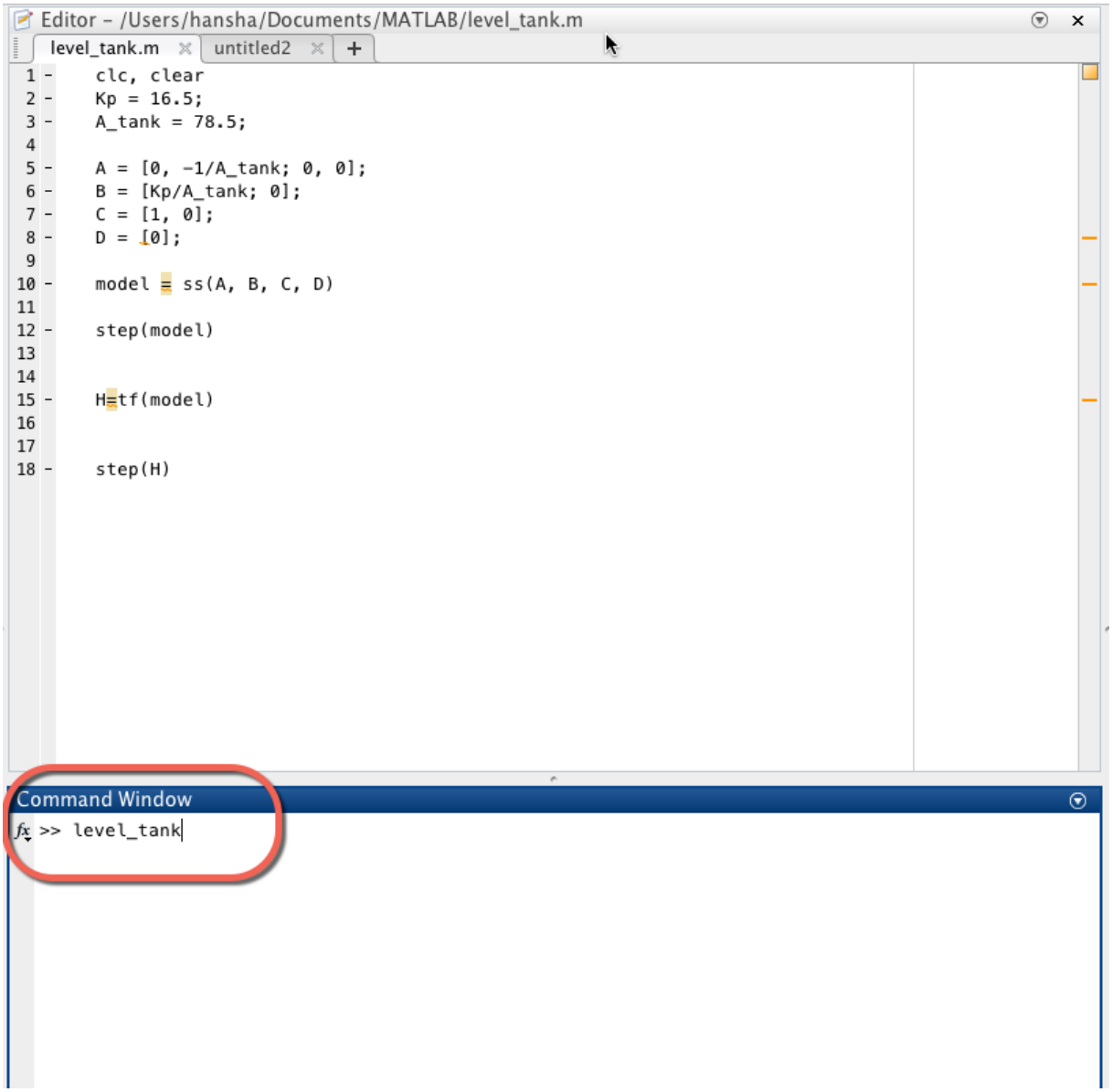


In the Editor you create a sequence of MATLAB commands that you save as a m-file (the file extension ends with .m). Push the “Run” button when you want to run your program.



If the code contains errors or warning the MATLAB compiler will let you know by displaying some colors symbols to the right in the Editor, as shown on the Figure above.

Running a m-file in the Command window (just type the name of the m-file and hit Enter to run the m-file):



You may open or edit a m-file using the open button in the toolbar.

An alternative is to type "**Edit** <name of m-file>" from the Command window.

Task 6: Script

Create a Script (M-file) where you create a vector with random data and find the average and the standard deviation

Run the Script from the Command window.

[End of Task]

Functions

MATLAB includes more than 1000 built-in functions that you can use, but sometimes you need to create your own functions.

To define your own function in MATLAB, use the following syntax:

```
function outputs = function_name(inputs)
% documentation
…
```

Or in more detail:

```
function [x, y] = myfun(a, b, c)   % Function definition line
% H1 line -- A one-line summary of the function's purpose.
% Help text -- One or more lines of help text that explain
%   how to use the function. This text is displayed when
%   the user types "help functionname".

% The Function body normally starts after the first blank line.
% Comments -- Description (for internal use) of what the
%   function does, what inputs are expected, what outputs
%   are generated. Typing "help functionname" does not display
%   this text.

x = prod(a, b);                    % Start of Function code
```

The first line of a function M-file starts with the keyword function. It gives the function name and order of arguments. In example above, we have 3 input arguments (i.e, a, b, c) and 2 output arguments (i.e, x, y).

The first line of the help text is the H1 line, which MATLAB displays when you use the **lookfor** command or the **help** command.

Note! It is recommended that you use lowercase in the function name. You should neither use spaces; use an underscore "_" if you need to separate words.

A Function can have one or more inputs and one or more outputs.

Below we see how to declare a function with one input and one output:

```
function output = functionName(input)
```

Below we see how to declare a function with multiple inputs and multiple outputs:

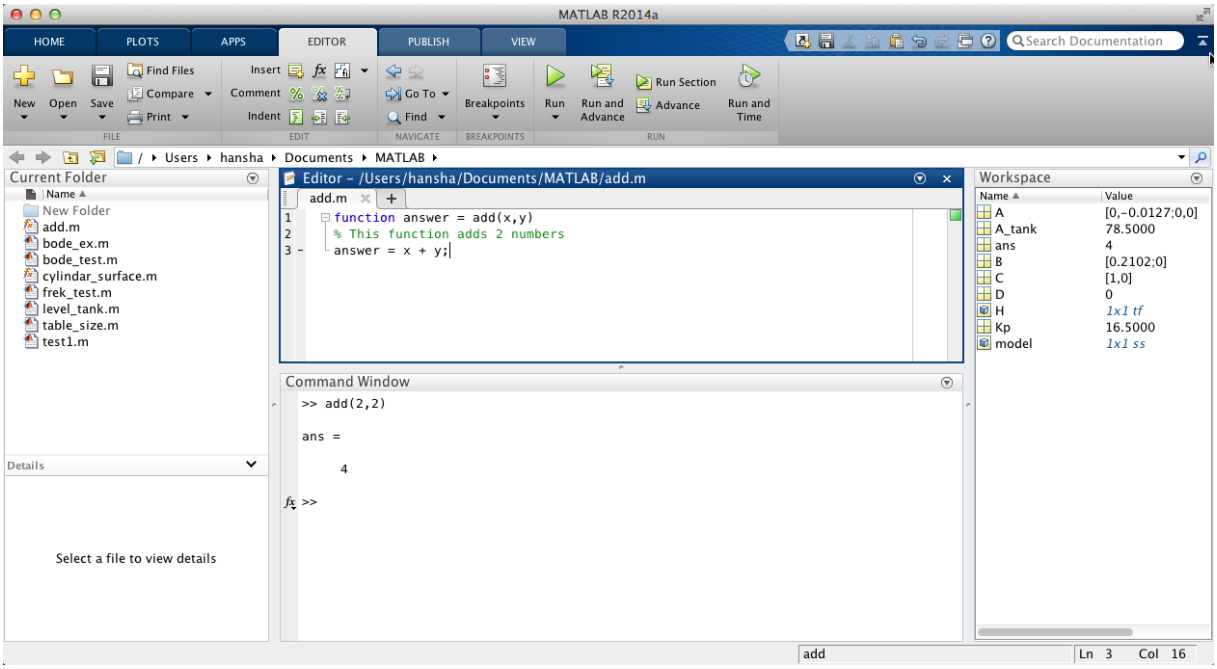
Example:

Here is a simple Example:

```
function answer = add(x,y)
% this function adds 2 numbers

answer = x + y;
```

Note! The function name (add) and the name of the file ("add.m") need to be identical.



You may use the function like this:

```
% Example 1:
add(2,3)


% Example 2:
a = 4;
b = 6;
add(a,b);


% Example 3:
answer = add(a,b)
```

[End of Example]

You may create your own functions and save them as an m-file. Functions are M-files that can accept input arguments and return output arguments. Functions operate on

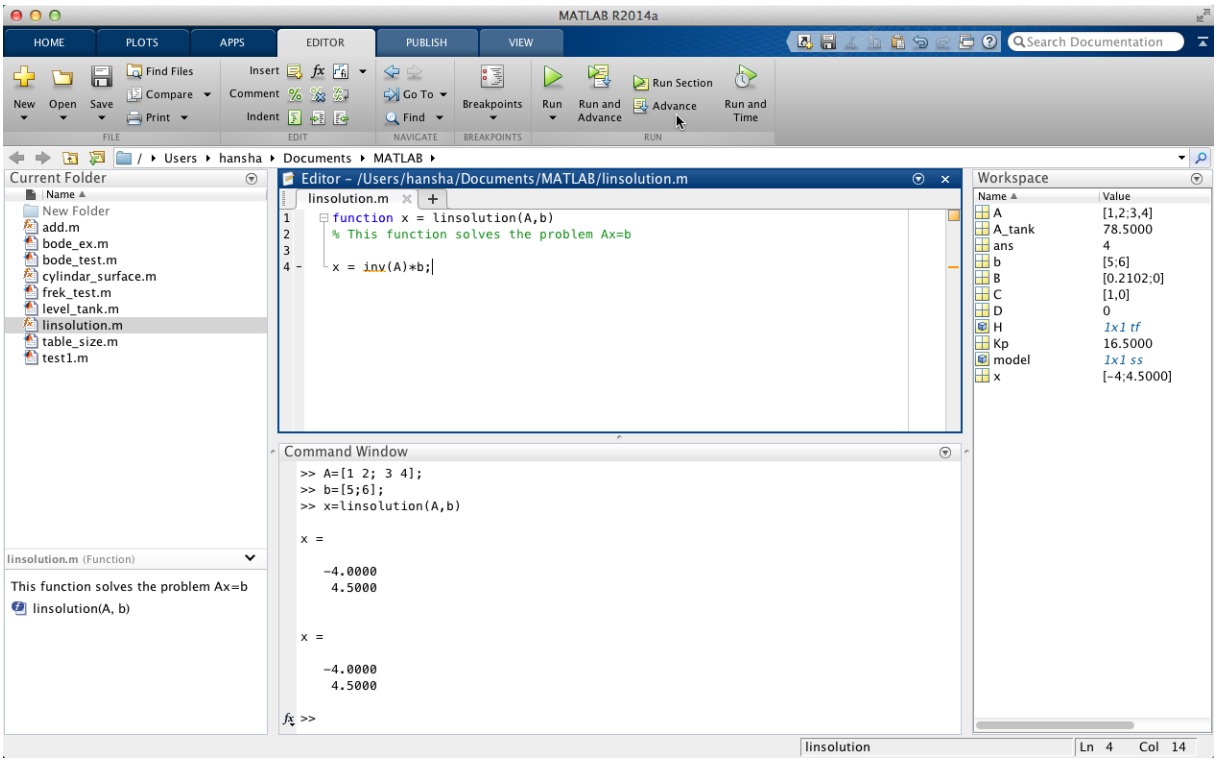
variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

Note! The name of the M-file and of the function should be the same!

Example:

Create a function called "linsolution" which solve $Ax = b \rightarrow x = A^{-1}b$

Below we see how the m-file for this function looks like:



You may define A and b in the Command window and the use the function on order to find x:

```
>> A=[1 2;3 4];
>> b=[5;6];
>> x = linsolution(A,b)
x =
   -4.0000
    4.5000
```

After the function declaration (`function [x] = linsolution(A,b)`) in the m.file, you may write a description of the function. This is done with the Comment sign "%" before each line.

From the Command window you can then type "`help <function name>`" in order to read this information:

```
>> help linsolution
   Solves the problem Ax=b using x=inv(A)*b
   Created By Hans-Petter Halvorsen
```

[End of Example]

Naming a Function Uniquely:

To avoid choosing a name for a new function that might conflict with a name already in use, check for any occurrences of the name using this command:

```
which -all functionname
```

Task 7: User-defined function

Create a function **calc_average** that finds the average of two numbers.

Test the function afterwards as follows:

```
>> x = 2;
>> y = 4;
>> z = calc_average(x,y)
```

[End of Task]

Task 8: User-defined function

Create a function **circle** that finds the area in a circle based on the input parameter r (radius).

Run and test the function in the Command window.

[End of Task]

Example: Multiple Return Values

Assume the following algebraic equation:

$$x^2 + 4x + 3 = 0$$

We can find the solution x by using the following general term:

Given the second order algebraic equation:

$$ax^2 + bx + c = 0$$

The solution (roots) is as follows ($a \neq 0$):

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Then we can create a function that finds the solution for x based on different input values for a, b and c, e.g.,

```
function [x1,x2] = solveeq(a,b,c)
    x1 = (-b+sqrt(b^2-4*a*c))/(2*a);
    x2 = (-b-sqrt(b^2-4*a*c))/(2*a);
```

Note! This function has 3 input arguments and 2 return values.

We can define values for a, b and c:

```
>> a = 1;
>> b = 4;
>> c = 3,
```

Then we can use the function like this:

```
>> [x1, x2] = solveeq(a,b,c)
```

Then MATLAB responds with the following results:

```
x1 =
    -1
x2 =
    -3
```

Of course, a better solution in this case may be to use vectors instead (because MATLAB has built-in features for vectors and matrices), like this:

```
function x = solveeq(a,b,c)
    x(1,1) = (-b+sqrt(b^2-4*a*c))/(2*a);
    x(2,1) = (-b-sqrt(b^2-4*a*c))/(2*a);
```

We define values for a, b and c:

```
>> a = 1;
>> b = 4;
>> c = 3,
```

Then we can use the function like this:

```
>> x = solveeq(a,b,c)
```

Then MATLAB responds with the following results:

```
x =
    -1
    -3
```

[End of Example]

You may use different loops in MATLAB

- For loop
- While loop

If you want to control the flow in your program, you may want to use one of the following:

- If-else statement
- Switch and case statement

It is assumed you know about For Loops, While Loops, If-Else and Switch statements from other programming languages, so we will briefly show the syntax used in MATLAB and go through some simple examples.

If-else Statement

The “if” statement evaluates a logical expression and executes a group of statements when the expression is true. The optional “elseif” and else keywords provide for the execution of alternate groups of statements. An “end” keyword, which matches the “if”, terminates the last group of statements. The groups of statements are delineated by the four keywords—no braces or brackets are involved.

The general syntax is as follows:

```
if expression1
   statements1
elseif expression2
   statements2
else
   statements3
end
```

Example:

Here are some simple snippets using the if sentence:

```
n = 5
if n > 2
    M = eye(n)
elseif n < 2
    M = zeros(n)
else
    M = ones(n)
end
```

or:

```
n = 5
if n == 5
    M = eye(n)
else
    M = ones(n)
end
```

Note! You have to use "`if n == 5`" – not "`if n = 5`"

[End of Example]

Example:

```
if A == B, ...
```

Note! If A and B are scalars this works – but If A and B are matrices this might not work as expected!

→ Try it!

Use instead:

```
if isequal(A, B), ...
```

→ Try it!

[End of Example]

Operators:

You may use the following operators in MATLAB:

Mathematical Operator	Description	MATLAB Operator
$<$	Less Than	<
$\leq$	Less Than or Equal To	<=
$>$	Greater Than	>
$\geq$	Greater Than or Equal To	>=
$=$	Equal To	==
$\neq$	Not Equal To	~=

Logical Operators:

You may use the following logical operators in MATLAB:

Logical Operator	MATLAB Operator
AND	&
OR	|

Task 13: If-else Statements

Given the second order algebraic equation:

$$ax^2 + bx + c = 0$$

The solution (roots) is as follows:

$$x = \begin{cases} \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}, & a \neq 0 \\ -\dfrac{c}{b}, & a = 0, b \neq 0 \\ \emptyset, & a = 0, b = 0, c \neq 0 \\ \mathbb{C}, & a = 0, b = 0, c = 0 \end{cases}$$

where $\emptyset$ - there is no solution, $\mathbb{C}$ - any complex number is a solution

→ Create a function that finds the solution for x based on different input values for a, b and c, e.g.,

```
function x = solveeq(a,b,c)
…
```

→ Use if-else statements to solve the problems

→ Test the function from the Command window to make sure it works as expected, e.g.,

```
>> a = 0, b = 2, c = 1
>> solveeq(a,b,c)
```

Compare the results using the built-in function **roots**.

Tip! For $\emptyset$, you can just type disp('there is no solution') and for $\mathbb{C}$ you can type disp('any complex number is a solution') – or something like that.

[End of Task]