|  |
| --- |
| School of Electronic & Electrical Engineering  FACULTY OF ENGINEERING |

ELEC5681M – Programming

UNIT 2

Introduction to MATLAB

LAB1 and LAB2 notes for practising for Test 2

**Introduction**

MATLAB (“Matrix Laboratory”) is a software package developed by The MathWorks. It was originally developed to carry out matrix algebra but is also extremely useful for numerical analysis, plotting functions and data and solving algorithms. It is widely used to solve problems in engineering, scientific, computing, and mathematical disciplines. As well as being useful in ELEC5681M, you will also find that knowledge of MATLAB and its capabilities will come in useful in other modules (especially in control theory and digital signal processing) and for your final project – you are encouraged to use it as much as you can in other modules.

MATLAB is proprietary software and thus requires a (expensive!) license.

University of Leeds offers a campus-wide license to MATLAB, Simulink, and companion toolboxes. Students are covered by the campus-wide license and can install software on their home or laptop computers.

Access instructions (also explained at University of Leeds IT link <https://it.leeds.ac.uk/it?id=kb_article&sysparm_article=KB0011947>):

1. Go to [University of Leeds’ MATLAB Portal](https://www.mathworks.com/academia/tah-portal/university-of-leeds-40586183.html) ( <https://uk.mathworks.com/academia/tah-portal/university-of-leeds-40586183.html> ) to download the software.
2. Click “Get Started Now” in the Download MATLAB box.
3. You will be asked to create a MathWorks Account. Once you do that, you will be associated to our MATLAB license and will be able to:

* Download and activate software on your personal computer
* Start using MATLAB Online from a web browser (on the link: <https://uk.mathworks.com/products/matlab-online.html>)

We strongly suggest you complete the 2 hours MATLAB Onramp tutorial that can be found on our [MATLAB Portal](https://www.mathworks.com/academia/tah-portal/university-of-leeds-40586183.html) (<https://uk.mathworks.com/learn/tutorials/matlab-onramp.html?s_tid=tah_po_mlonramp>).

If you have trouble installing MATLAB, go to the [MATLAB Portal](https://www.mathworks.com/academia/tah-portal/university-of-leeds-40586183.html) and click “Customer Support” to get help.

**MATLAB Test format**

* Test 2, like test 1, is a 1-hour Mobius test that will be held on Thursday 14 December at 11:00 am (UK time).
* The test is automatically graded.
* The questions focus on the introductory principles of MATLAB use
* Please remember: **you only have one submission attempt**, you may alter your answers at any time before the deadline, however you may submit your answers for assessment only once.
* This document will provide very useful examples for practising and some essential guidelines to answer all the assignment questions and you may use the lab sessions to seek help.

**Contents**

[1. MATLAB – an expensive calculator 4](#_Toc52215771)

[2. Using variables 5](#_Toc52215772)

[3. Functions 7](#_Toc52215773)

[4. Arrays 7](#_Toc52215774)

[5. Plotting *data* 10](#_Toc52215775)

[6. M-files and constructing programs in MATLAB 12](#_Toc52215776)

[ if statement 13](#_Toc52215777)

[ The for loop 14](#_Toc52215778)

[ The while loop 15](#_Toc52215779)

[ Data type protection 16](#_Toc52215780)

[ Code testing 18](#_Toc52215781)

# MATLAB – an expensive calculator

* Load up MATLAB
* For simple problems MATLAB command can be typed line by line but in the most realistic cases constructing MATLAB script so-called M-file would be much better approach.

MATLAB can be used to carry out basic arithmetic operations such as addition, subtraction, multiplication, division and powers as well as the more complicated operators. Type in the following in the MATLAB command line:

>> 1.76+8.35

>> 12.43-5.86

>> 34.23\*6.12

>> 76.92/3.01

>> 5^2

>> 16^0.5

Note that powering to 0.5 is same as seeking square root.

MATLAB is easy to follow when doing simple operations using a single operator. When using multiple operators it is important to understand the *precedence* of the operators and brackets. The order is as follows (listed in order of precedence):

1. Brackets
2. Powers
3. Multiplication and division (left to right)
4. Addition and subtraction (left to right)

Example:

>> (54-50\*3)/(62-5/4\*3)

Pay very close attention to use of brackets. You should also develop a sense of seeing entry (54-50\*3)/(62-5/4\*3) and writing the corresponding mathematical form.

If you have an expression where commands are nested without any use of brackets, the commands are then read from left to right, in the example above 5/4\*3 is not 5 divided by 4\*3, but rather 5 gets divided by 4 first, and then everything is multiplied by 3.

>> 2^5^4^6/3/5/6\*3

Just follow the expression from left to right and add brackets where necessary, even though the original expression has no brackets, the actual mathematical expression does need them.

Calculate the following mathematical expressions with MATLAB – input expression exactly – do not simplify them (do not replace (11-9) with 2).

Input the following using MATLAB and answer which of the two offered expressions (**a)** or **b)** MATLAB calculated (show both your input and the MATLAB output).

* + - (2-3)^4/7 is it **a)** **b)**
    - 2/3/4/5 is it **a)** **b)**
    - 2^((3\*4)^2) is it **a)** **b)**

# Using variables

Do the following operation in MATLAB

>> 4+4

ans =

8

Now type in ans. What happens?

MATLAB has stored the result of the operation in a variable called ans (obviously short for answer). Now, type in ans\*2. What happens? After a variable has been declared, it is replaced everywhere it appears by the value stored in it. This is extremely useful and easy to do in MATLAB.

For instance we can type

>> x = 4

and then type the equation

x^2

which will give ans = 16. This may seem trivial but comes in very handy when we want to carry out a calculation repeatedly using different values i.e. suppose we want to do 4\*x^3+2\*x^2+5\*x+6 for different values of x (say 1,2,3,4 and 5). We could do:

>> 4\*2^3+2\*2^2+5\*2+6

etc. for each one or we could set x to which value we want, type out the equation, this time keeping the x variable. We can then set x to a different value and then scroll back to the equation using the up arrow cursor key to cycle back through previously typed lines.

We can name variables (almost) anything. They must be one word and cannot contain spaces. Spaces are normally identified using an underscore (\_) or sometimes by a capital letter at the beginning of the second word i.e. two\_words or twoWords. Note that variables are case-sensitive; voltage, VOLTAGE and Voltage are all different variables!

In MATLAB, numbers can be displayed in ‘e’ notation i.e.

1e3 = 1 x 103 = 1000

1e-6 = 1 x 10-6 = 0.000001

* + - Suppose we have a resistor in a simple circuit. If the voltage across the resistor is 5 V and the current through it is 0.25 mA what is its resistance? Do this on MATLAB, using appropriate variable names. Can MATLAB understand physical units?

MATLAB also has some well known constants:

* + - Try typing in pi. What happens? Try typing in i or j. What happens?

These are examples of built-in variables. MATLAB has variables with these names already so you should avoid using them to name your own variables.

* + - Type z=pi+4i. What happens?

Type pi=7 and i=5, then type z=pi+4\*i. What happens? Clear memory to reset the values to default.

# Functions

MATLAB has a large range of built-in functions (e.g. trigonometric, exponential, logarithmic etc.). There are too many to list here but MATLAB includes an extensive help menu which contains details of all functions available (Help, Function browser, Mathematics, Elementary Math/Polynomial/Constants etc.). The best way to really get to know how to use MATLAB is to experiment.

Some functions may have different names than the ones you are used to. For example tangent of x is not **,** but rather , similarly some functions may have completely different use, natural logarithm in MATLAB is , while log for base of 10 is .

Answer the following using MATLAB

* + - ; Is this what you expected? Have you cleared memory after previous task?
    - ; Do you notice anything strange? Why is this? Is MATLAB wrong?

# Arrays

So far we have been dealing with operations containing single variables. The power of MATLAB really becomes apparent when we begin to use *arrays*.

Consider the array

>> a = [1 2 3 4 5 6 7 8 9 10]

a =

1 2 3 4 5 6 7 8 9 10

This creates a one-dimensional array of numbers. Try repeating the above command but add a semi-colon ; to the end of the line. This semi-colon suppresses the output which is sometimes useful, especially when you may have any array containing thousands of points!

Suppose we want to make a longer array, we could step though it one by one, adding each required number, or we could the following shortcut

>> a = [1:1:10]

a =

1 2 3 4 5 6 7 8 9 10

The first number is the first element of the array, the second the increment step, and the third is the final value.

It is possible to carry out arithmetic operations on each element of the array at once i.e.

>> a+10

ans =

11 12 13 14 15 16 17 18 19 20

Suppose we want to make an array of numbers

>> a = [24 18 37 11 15 30]

a =

24 18 37 11 15 30

A specific element of array *a* can be selected by writing

>> a(3)

ans = 37

MATLAB has a built-in command, which can provide a total number of elements in an array (quite useful if array is long and it is not easy to count elements or if length of array is not known in advance i.e. can be different form case to case. Command is length*.* Therefore for above example:

>> length(a)

ans = 6

MATLAB does not know how to multiply two arrays, because from algebraic sense this can have multiple meanings, mainly because MATLAB is designed for working with matrices. In most practical cases, you will consider array multiplication to be product between two arrays where elements in the same position will multiply each other. Naturally both arrays would need to have same number of elements, and in any programming language you can use a for loop to conduct such product as:

**for** i=1:1:length(a)

c(i)=a(i)\*b(i)

end

For loop simply repeats sets of commands the number of times specified by the loop counter.

The notation **i=1:1:length(a)** means construct iterator called **i**, increment in steps of 1 until the value of **i** becomes **length(a).** Note that if step is 1, you may achieve same effect by writing **for i=1:length(a).**

This is how you can iterate through any array, **i** is simply the counter that refers to the array position where some value is stored.

The command **c(i)=a(i)\*b(i)** means in array **c,** in **i-th** position, store the value of product of **a(i)\*b(i)**

Fortunately, you do not need to write these for loops every time you operate with some arrays. You can instead use *overloaded* (in programming this means giving additional meaning) operators by simply placing a dot before the desired operator. For instance, you can multiply arrays **a** and **b** by simply writing **c=a.\*b**

The dot overloads operators \*, / and ^

Pay attention however that if you ever need to calculate something complicated with arrays as it is correct to write **x.\*y./z.^5** however every time you use dot operator you are forcing MATLAB to write a for loop for you. It is much more numerically efficient to manually write a for loop as

**for** i=1:length(x)

results(i)=x(i)\*y(i)/(z(i))^5

end

For most beginner level uses using dot operator will not cause significant numerical issues and you will get your results quickly. However, if you are solving some complicated problem, you will need to think about numerical efficiency.

Very important note is that a\*b returns error in MATLAB as it is algebraically undefined operation, however a/b does not result in error as algebraically this is inner product of two arrays and result is sum of products a(i)\*b(i) which is just a number. MATLAB will not warn you of this mistake, so if you have an expression as

a) sin(x)/x .\* exp(x) – this just exp(x) multiplied by some number as sin(x)/x is missing a dot

b) sin(x)./x .\* exp(x) – this correctly implements the above expression

**This is one of the hardest mistakes to catch in your code, so always double check your expressions if you are writing some formulas with arrays.**

* + - Create an array called odd1 which contains all the odd numbers from 1 to 20 and an array called even1 which contains all the even numbers from 1 to 20
    - Try adding the arrays together.
    - Now try multiplying the arrays together. What happens and why? Try using the .\* operator instead. What does this do?

# Plotting **data**

MATLAB is also extremely useful for plotting functions and data. This is achieved by combining what you have learned about functions with what you have learned about arrays.

Say we want to plot the function over the range . First we create an array to store the x values

>> x=[0:1:6]

x =

0 1 2 3 4 5 6

We then calculate y

>> y=x.^2

y =

0 1 4 9 16 25 36

Remember to use the dot . when carrying out arithmetic on arrays. We then use the plot command

>> plot(x,y)

Notice the order of the arrays. Suppose we want to add a title.

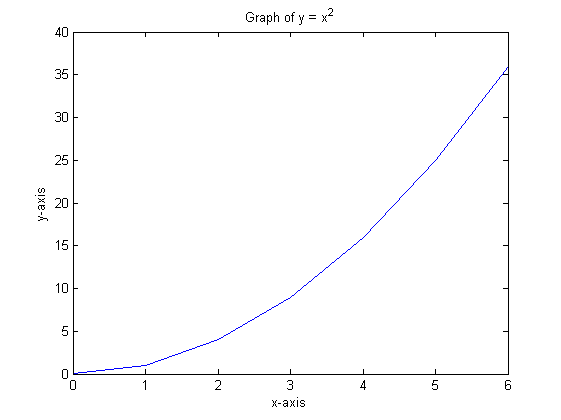
>> title('Graph of y = x^2')

and label the axes

>> xlabel('x-axis')

>> ylabel('y-axis')

You should end up with something that looks like this:



You can save the figure by going to File, Save As. PNG is a good format to use as it is small in size and compatible with Microsoft Word.

We can customise the plot by using the following command

plot(x,y,'r-')

The r makes the data be plotted in red and the – makes the data appear as a solid line. These can be modified as follows: *y yellow, m magenta, c cyan, r red, g green, b blue, w white and k black* and *. point, o circle, x x-mark, + plus, - solid line, \* star, : dotted, -. dash-dot and – dashed.*

We can plot multiple functions on the same figure using

>> plot(x,sin(x),x,cos(x))

We can then add a legend

>> legend('sin(x)','cos(x)')

You should have noticed before that when you plot a function, any previous figure that is open is over-written. If we want to plot multiple functions on the same figure we can also use the hold command

>> hold

To turn this off we use the hold off command.

* + - Create an array for x from 0 to in steps of 0.1 ; Look at the last value in the array. Is this ? What has happened?

In these types of cases it is better to use the linspace command. linspace(a,b,c) creates an array of c points in between a and b inclusively.

* + - Create a new array of 10 elements called x1 from 0 to and array of 100 elements called x2 from 0 to .Make a plot of showing two graphs, one with 10 points and the other with 100 points, the plot should include a title, axes labels and a legend. The 10-points line should be a solid green line and the 100-points line a dashed red line.

# M-files and constructing programs in MATLAB

We will now digress slightly to cover M-files. These will help you to cover the tasks more quickly. We open a new M-file by going *File/New/Blank M-file* or click the *New Script* button one the top left.

This should bring up the Editor window in which we can type our commands.

Type the following into the Editor window and click run button (green triangle). What happened?

x=linspace(0,2\*pi,1000);

y=sin(x);

plot(x,y)

You may be asked to save it before you run it.

M-files represent your code. In previous task you saw that you can do a lot in MATLAB from a command line, but in most cases this is really impractical.

In M-file you type a set of commands which are executed only when you click the run button, nothing is performed when you type commands in the editor. In MATLAB you will use .m files for main codes and functions, you will learn functions in the next assignment.

It is highly recommended that in **your main .m file code (not the .m file you use for functions)** type the following lines:

clear all;

close all;

clc;

First command clear all is the most important – it clears your memory (which you can view in the workspace) before executing the program – sometimes your code may remember something from previous execution and then you may get illogical results even if there is nothing wrong with your code. close all command closes all the pictures, if you had many plots in your code, and clc just clears your command line view. First command is crucial, other two are used for neatness.

Your .m file can contain any number of lines and commands depending on what you need to do, you can use a large verity of built-in MATLAB function.

MATLAB has very efficient help section, help pages for every function in MATLAB look similar, they always start with the name of the function, short description, syntax, explanation of the syntax and examples. At the end of the help page there is a section ‘*See also*’ where MATLAB recommends you functions which are closely related to the one you’re viewing.

Every time you violate rules (mathematical or coding) MATLAB would output an error message and direct you to the line where the error happened, however latest version of MATLAB has a small trick for dot . operation, which can completely give you illogical results, without MATLAB warning you about the error! This will be discussed in the FOR loop section.

**To construct MATLAB program:**

* Start the MATLAB program by double clicking on the *MATLAB icon* bringing up the Command Window
* For simple problems MATLAB command can be typed line by line but in the most realistic cases constructing MATLAB script so-called M-file would be much better approach
* Click on *File-New-M-file*
* Type program in the editor window.
* Save the program in a directory of your own choosing. It is best that this directory contain only your own MATLAB programs.
* To run the program, return to the command Window and in the *current directory slot* select the directory in which the program has been saved. Then in Command Window type in program name without the .m extension. (For example: Suppose the program has been saved as *example1.m* then after >> character type in *example1* )

As explained, you can find a lot of on-line information on MATLAB and number of student textbooks with useful introduction on MATLAB structure as well as in well documented MATLAB *help and in Mathworks web page*. Only a few useful hints on characteristic MATLAB procedures are given below:

## if statement

Syntax:

if logical expression

statement;

statement;

else

statement;

statement;

end

If the logical expression is true, the upper set of statements is executed. If the logical expression is false, the bottom set of statements is executed.

* Logical expressions are of the form

*a==b; a<=b;*

*a<b; a>=b;*

*a>b; a~=b;*

* The *if-elseif* ladder

Syntax:

if (logical expression 1)

statements;

elseif (logical expression 2)

statements 2

else

statements;

end

## The for loop

Itprovides the means to carry out series of statements with just a few lines of codes.

Syntax:

for m=1:20

statement;

statement;

--------;

end

The computer sets the index *m* to 1, carries out the statements between the *for* and *end* statements, then returns to the top of the loop, changes *m* to 2, and repeats the process. This continues until m I set to 21, in which case the computers leaves the loop.

In section 1.4 we shortly discussed how MATLAB deals with arrays and how using the dot . before an arithmetic operator can simplify your code. However, existence of the . is built-in MATLAB property, any other programming language would require you to make up your own operators (i.e operator overloading in C++), however, in every programming language you can deal with arrays by iterating through them. In MATLAB, array of *n* elements, stores the first variable in *a(1),* the second in *a(2), …,* the last in *a*(*n*), you can then smartly access each element by making a for loop for *iterator=*1 : *n* (or *length(a)*) andaccess elements in *a* as *a(iterator).* For example product of *a* and *b* can be performed as

for i=1:length(a)

c(i)=a(i)\*b(i);

end

This is valid only if *a* and *b* have the same length! Doing *c=a.\*b* literary implements the code above, and this works for any arithmetic operator (+,-,\*,/,^).

It is really important to understand that MATLAB is made to deal with matrices (therefore MATrix LABoratory) and it considers every variable or array as matrix. Constant is a *1 x 1* matrix, and array of *n* elements is a *1 x n* matrix.

Operators (+,-,\*,/,^) are designed to work with matrices, and follow the corresponding mathematical definitions. In most cases MATLAB will warn you that dimensions of your arrays are not compatible for general definition of these operators, and you will know that you forgot the dot, however, in the latest MATLAB version, operator / was upgraded and may cause severe consequence in your code if you wanted to do ./ operation. Type the following in the MATLAB command line:

x=linspace(0,2\*pi,10);

y1=cos(x)/x;

y2=cos(x)./x;

Both of these expressions are valid, and MATLAB will never know what you need in your code. Take a good look in your workspace and you will notice that *y1* is a number (check help page on operator / to see why), while *y2* is an array of 10 elements that divided each element of array *cos(x)* with array *x* and ./ literary replaced the code similar to the one above with a for loop. You can also notice that the first element in *y2* is *Inf.* This happened because the first element of *x* is 0, and division by zero is not defined and tends to infinity. MATLAB has another variable *NaN* (not a number) for completely illogical operation, and if you are writing a big code, MATLAB will not warn you about *Inf*s and *NaN*s in your code, it will fill the arrays with them and give you the end result (which may be very confusing), and it is up to you to check for consistency of your data.

Remember, iterating through arrays with for loops is always safe, dot . is very powerful and saves you from typing lots of code, but you always need to double (and triple) check your code.

## The while loop

Syntax:

n=0;

while logical expression

n=n+1;

y(n)=n^2;

end

In the while loop, the computer will carry out the statements between while and end statements as long as the condition in the while statement is satisfied. You may interpret while loop as **for loop+if**, the main difference between forand whileis that in forloop you always determine how many times you want to repeat your commands, however whileloop will repeat commands as long as the condition is true. This means that number of iterations in the while loop is generally undetermined, and this is of great use for infinite loops and GUI (graphic user interface) applications, however number of iterations is usually very important when you wish to optimize your code. To count number of iterations in the while loop, you need to set counter=0 before the loop and within the loop just increment it (in the example above, variable n behaves as a counter!).

In the example above variable nis used as a counter, logical expressions are explained in ifcommand, for example n<10 would be a good condition, then the code above would repeat 10 times and the command y(n)=n^2 would create array ywith 10 elements, where elements are squares of n=1 2 3 … 10. Keep in mind that if you delete the line n=n+1 in the code above you have an infinite loop, because n=0and 0 is always <10. **To terminate code execution in MATLAB hold *ctrl* and press *C*.**

In programming it is important that your codes are neat and fast, you can measure time of your codes by using tic, tocaround the commands, if you do:

a = linspace(1,50,50);

tic

sum(a)

toc

MATLAB will output the time needed for the command sum (built-in MATLAB’s function).

Most of the time in coding you will spend in fixing the code and handling errors and illogical results, when writing a code you should also test it for anything that user may do, and compare your test with something that works (analytical result or built-in function).

## Data type protection

Every variable in programming must have a data type. You know many of them already: whole numbers (integer), decimal numbers (float and double), arrays, matrices etc. In more advanced languages you may create custom data types and pay attention to behaviour of your variables. Consider this example:

In MATLAB

**function** y=myfunction(x,a,b) )

y=a \* x.^2 + b;

end

In C/C++

**int[]** myfunction(**int[]** x, **double** a, **double** b)

**{**

**int[]** y**;**

**for** (**int** i=0;i<x.size(); i++)

**{**

y[i]=a\*x[i]\*x[i]+b;

**}**

**return** y;

**}**

Clearly there is a lot of difference between these two languages. A very simple code that multiplies square of values in array x by a and adds b looks much more intimidating in C/C++. Notice that in the example above, function return type (array of integers int[]) needs to be specified before the function name, and every argument in the function needed exact data type specification. Note that in MATLAB squaring x can be done by placing a **dot** before the power operator, while in C/C++ for loop needed to do this manually.

Contrary to other programming languages, you do not need to specify exactly the data type of the variables you are using. MATLAB, by default, treats all numbers as double floating point values which is usually very useful in mathematical problems. At the beginning stage, this looks very pleasant, however it is incredibly challenging and most of the debugging of your codes in the future will fall in chasing mistakes caused by defaulting data types.

MATLAB’s functions allow array and matrix input and sometimes this can create undesired consequences in your code without getting any warning from MATLAB. The code above could easily accept numbers a or b as matrices!

The neglect of data type in MATLAB runs in every expression you write, not just functions.

For instance, if *x* was an array, and you typed for i=1:x,the loop will work only for i=1 and you will get an answer at the output, but loop will not iterate any further. Similarly if you put array into while loop condition as while x<5 you may get trapped in an infinite loop or be considering only the best/worst case scenario. Similar issue arises if you put array/matrix into if statement (unless you wanted to do so).

Note that it is programmer’s responsibility to protect the code from illogical use, for example, proper implementation of a code that performs *factorial* calculation would be:

n = input('Provide positive integer number');

if (n<0)

error('Only positive input is accepted!')

end

if floor(n)~=n

error('Only integer values are accepted')

end

if numel(n)>1

error('Only single integer values are accepted, array/matrix input is prohibited')

end

myfact=1;

for i=1:n

myfact=myfact\*i;

end

The first two if statements protect the function from negative and decimal input, and the third if statement protects the function from an array/matrix input. Additional protection may be added to this code if there is a danger from data overflow (trying to represent number that is larger/smaller then data type limit).

Since MATLAB does not need data type specifier declaration, unskilled user may attempt sending illogical input to the function. Note that this can create a lot of execution problems, for instance, **for** (or **while**) loop would not run correctly if n was an array or a matrix, however you will not get an error message! In a for loop the first iteration when i=1 will execute as myfact=myfact\*1*;* and output of this function will be myfact=1 with no error messages from MATLAB! As a creator of the code you need to be very careful and predict when you actually want your input to be an array and when not. MATLAB has “dot” operators as *./* and .\* to deal with the arrays which is useful when dealing with mathematical expressions, however if you forget the dot, illogical output may occur with no errors on the screen! Generally “dot” operations simply save you from writing a **for** loop that iterates through the array and performs \* or / operation on each element so this is reason as they are called element-by-element operations.

Make sure you fully understand the usage of these operators.

## Code testing

It is approximated that up to 70% of time for code development is used on removing logical and technical issues (commonly referred as bugs). What can save a lot of time is high quality flow chart and algorithm for the problem and incremental development (writing small pieces of code and testing them as you develop). Modern software development deals with very complicated codes that may consist of several hundred thousands of lines, thus the projects must be split between different teams and code testing is the most important part of the development.

The most complicated issues are those that process the input illogically without causing built–in functions to fail, which due to high flexibility with MATLAB in terms of data types can happen frequently (as described previously).

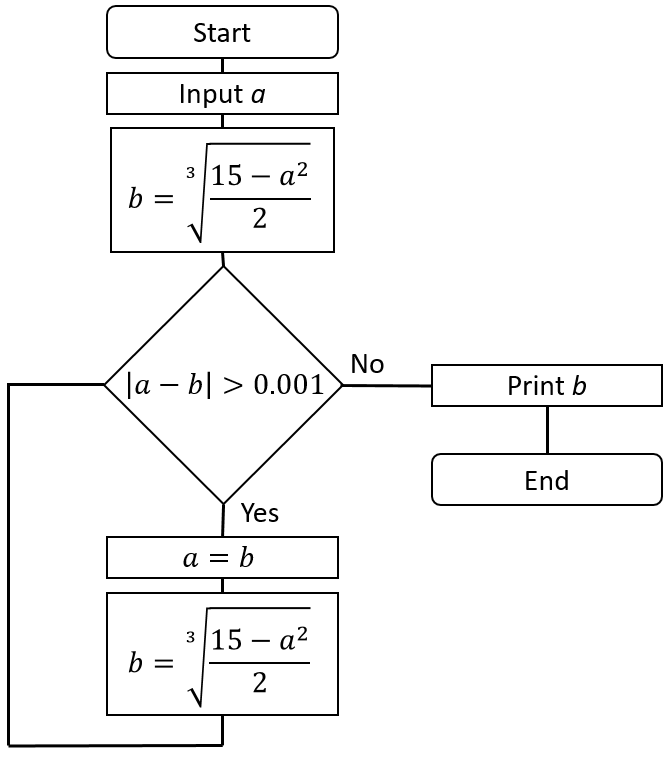
Every code therefore needs to be tested. Complex and extensive programmes in many companies have entire teams of developers dedicated only to the verification of the code.

The simplest tests would require you to compare the output of your code with a more reliable function. The more advanced tests would test the code on a set of expected data input and in many text books about programming you will find problems that specify the input, desired output and several tests samples. Every time your code fails a test, you must determine the source of the error and find a way to fix it.

* + Write a simple MATLAB program which will calculate sum of the first *n* integer numbers.**Do not use neither the well-known analytical formula nor the built-in MATLAB sum() function for solving this task.** Test your code for n=100and compare with the formula and MATLAB built-in **sum()** function (figure out the syntax in MATL0AB help page for the usage). Test your code for different values of n and compare it with formula (if you wish to be fully certain that your code works). Make sure you protect your code from illogical use (n needs to be a positive integer).
  + Similarly, write a MATLAB program which calculates sum of all elements in array *b* which needs to be provided by user (use ***input()*** command). **Do not use built-in MATLAB** sum()**function for solving this task.** Test the program with array [5 12 33 -45 22 -28 125 -14 35] and compare the result with MATLAB’s **sum()** function. *Advanced: Test your code for random array with 10, 50 and 500 elements and compare it with MATLABs* **sum()** *function. (Hint: Explore MATLAB help to figure out how to make random array of* n *elements).*

**Example of an iterative algorithm**

The equation *2x3+x2-15=0* may be solvedby iteration  using the flow chart given below:

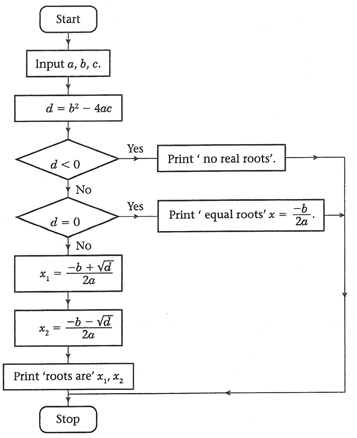


* Write MATLAB program which implements the algorithm above, use ^(1/3) for powering the expression and answer following tasks:
  + Use a=2 to find a root of the equation
* Use a=20 to find the root of the equation. What do you notice? (Double c*lick on variable b in the MATLAB workspace*)
* Use **nthroot(expression,3*)*** function (look MATLAB help) instead of **expression^(1/3**) and repeat above given tasks. What do you notice now? Why did this happen? Can you count the number of iterations?

**Example on roots of quadratic equation**

This flow chart can be used to find the real roots of a quadratic equation of the form

*ax2+bx+c=0.*



Construct MATLAB program which will solve roots *x*1 and *x*2 for any set of input variables *a, b* and *c* and demonstrate how your program works for different values of variables *a, b* and *c*. Test specific examples when you have two different roots *x1* and *x2* , when you have equal roots *x1*=*x2*, and when there are no real roots. For given examples plot the function *f(x)=ax2+bx+c* and check approximately whether position (crossing of the function with *x-axis)* of calculated roots are accurate. Finally, insert your calculated roots (if any) back into quadratic equation and check that condition *f(x)=0* is satisfied (with possibly small numeric errors).