

# Introduction to Matlab

The aim of this course is to introduce students to computer based modelling techniques. This course will enable you to perform simple mathematical tasks using a computer. Following this we introduce the concept of writing simple computer programs to perform repetitive tasks by using *for loops*, *while loops* and *if-else-end* constructions. The objective is to enable you to model engineering problems using the computer. As part of this course, you will use Matlab to perform these tasks.

## What is MATLAB?

The MATrix LABoratory program was initially designed to solve linear equations and eigenvalue problems. It enables scientists and engineers to use matrix based techniques to solve problems without having to write programs in traditional languages such as C, FORTRAN or BASIC. MATLAB is a commercial product and student edition is available at a discounted price in local bookstores.

## Suggested Reading

The main reference text which applies to this course is [1-4]. In addition there are many good web based resources which include Matlab tutorials. These can be found by searching the web such as <http://www.math.mcmaster.ca/~rogern4/2zz3/tutorial4.pdf>

## References

- [1] D. Hanselman and B. Littlefield, 2005. Mastering Matlab 7, Pearson Education Ltd
- [2] BR. Hunt, RL. Lipsman, J Rosenberg, 2001, A guide to MATLAB: for beginners and experienced users, Cambridge University Press, 2001
- [3] AJ Knight, 2000, Basics of MATLAB and beyond, Chapman & Hall/CRC, 2000
- [4] Austin, M and Chancogne, D 1999, Engineering Programming C, MATLAB, JAVA, John Wiley & Sons, Inc. Herniter, 2001, Programming in MATLAB, Brooks/Cole

# Computer-Based Modelling: Course Summary

The lab course is structured in the following manner.

**Part 1** will give an introduction to Computer-Based Modelling using Matlab. The first session will provide basic knowledge in the following areas:

1. Running Matlab,
2. Getting help,
3. Constants and variables,
4. Special symbols, Last line editing and Viewing variables
5. Accuracy,
6. Workspace and diary,
7. Scalars, arrays and matrices
8. Polynomials
9. Basic plotting using Matlab.
10. IF and FOR constructions
11. Programming a graphical user interface
12. Appendix

**Part 2** is a project that you should do by yourself in order to maximise your individual learning outcome.

The projects will be undertaken for four sessions. If you feel that you require working on your project outside lab hours, please feel free to do so. The project will be assessed after submission by the course supervisors. These projects make up 50% of the marks for this course.

# 1. Running Matlab

To start Matlab on the PCs at UWE/Frenchay, open the Start menu, select 'All Programs' and choose Matlab.

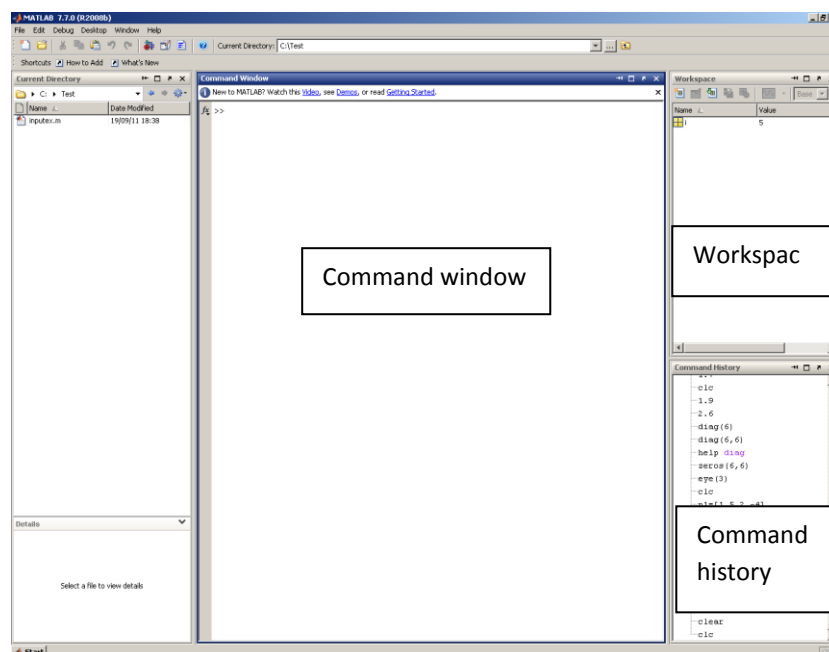
In the centre of the Matlab screen is the Matlab **Command window** including its prompt `>>`. Whenever Matlab is waiting for commands from the user, it will show this prompt, and the flashing vertical cursor bar.

When you want to exit from Matlab type:

```
>> quit
```

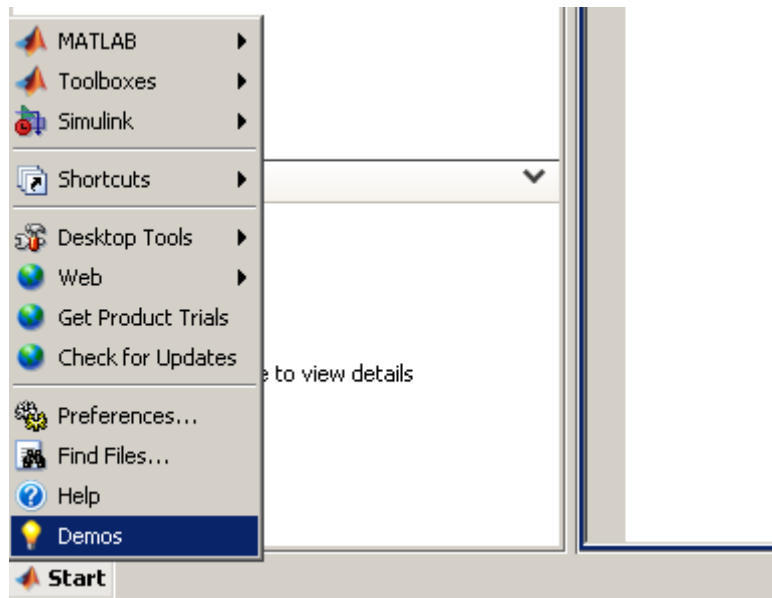
or select File/Exit MATLAB from the menus or type CTRL+Q.

On the left or right hand side of the screen, there should be the **Workspace window**. This window is extremely useful in assessing the value of variable that you declare in a program for example. The **Command History** 'window' shows your code history, i.e. your actions performed in the Command window.



## 2. Getting help

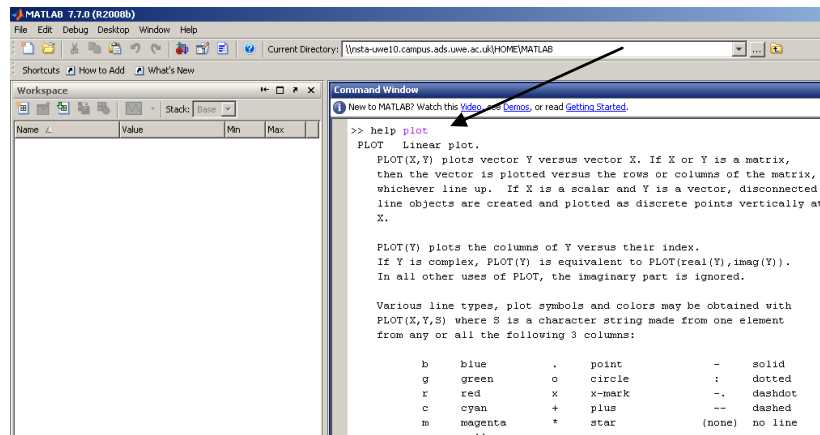
Matlab has an extensive help system, which you can access from the Help menu. Clicking on Demos gives you a window from which you can select a topic to view as a demonstration. To Start the Matlab Demos, open Start in the lower left corner of the Matlab screen, click on 'Demos' and then in the directory tree on 'Matlab'.



From within the help menu, clicking on MATLAB Help enables the user to browse through a list of available help topics. Help for individual commands can also be obtained by typing the help command at the prompt. For example, to get help on the command *plot* type

```
>> help plot (or doc plot)
```

which will print a description of what *plot* does. It also gives you a list of related commands. The *help* command requires that you know exactly what command you want help with. If you are not sure, you can use the command *lookfor* followed by a keyword. To interrupt the *lookfor* command type CTRL+C. In the lab classes try and use the Matlab help system as much as you can, this is especially relevant when demonstrators are busy.



## Exercise 1

Run some of the demos, for example **Writing a Matlab program (~5min)** or **Creating Graphical User Interfaces (~10min)**, under Demos within the demo help window. The demos will give you an idea of the capabilities of Matlab. Then use the Matlab help system to find information on the following: square root; trigonometric, exponential and logarithm functions.

### 3. Calculations, constants and variables

Matlab can be used as a calculator, by typing in an expression followed by the Enter/Return key. For example typing:

```
>> 2 + 3
```

gives

```
ans = 5
```

For most scientific computing packages the following standard operators are used: + add, - subtract, \* multiply, / divide, ^ power. Negative numbers can be written by prefixing with -. Careful use of brackets is often required in these types of calculations. For other useful commands regarding math operations please see the table below

Meaning	Function in MATLAB	Example
Sin of x (in radians)	sin(x)	sin (3.14)=0
Cosine of x (in radians)	cos(x)	cos (3.14)=-1
Tangent of x (in radians)	tan(x)	tan(3.14)=0
arcsine of x	asin(x)	asin (1)=1.57
$e^x$	exp(x)	exp(1)=2.718
$\log_e x$	log(x)	log(2.718)=1
$\log_{10} x$	log10(x)	log10(100)=2
Absolute value	abs(x)	abs(-2)=2
Keep integer part of x	fix (x)	fix(3.8)=3

For most mathematical calculations direct computation using expressions is too cumbersome, and it is more efficient to represent numbers algebraically. Matlab allows us to do this for any letters of the alphabet or other combinations of letters provided they contain no spaces. In general there are two classes of numbers we want to represent, those which have a constant value, and those which can vary. We refer to these as constants and variables, and both can be denoted by letters or names. For constants we often use  $a$ ,  $b$ ,  $c$  etc, for variables  $x$ ,  $y$ ,  $z$ .

Many problems in engineering applications are related to values which change over time, usually denoted by  $t$ . In mathematical notation the fact that the variables change with time is often denoted by

writing  $x(t)$ ,  $y(t)$  etc. In terms of scientific computing we have to declare all the constants, before we can use them.

## Exercise 2

Try the following commands and note the difference in using numbers and variables.

<i>Treat MATLAB as a calculator</i>	<i>Use variables</i>	<i>Comment</i>
<code>&gt;&gt;3+8</code> <code>ans=</code> <code>11</code>	<code>&gt;&gt;x=3+8</code> <code>x=</code> <code>11</code>	<i>There is no difference between the calculator mode and variable mode, except that the results are saved into different places.</i>
<code>&gt;&gt;(3+8)*(3+8)</code> <code>ans=</code> <code>121</code>	<code>&gt;&gt;y=x*x</code> <code>y=</code> <code>121</code>	<i>Since x contains the result of 3+8 we can use variable x to calculate y.</i>
<code>&gt;&gt;sin(3.1416)</code> <code>&gt;&gt;cos(3.1416)</code>	<code>&gt;&gt;x=3.1416;</code> <code>y1=sin(x); y2=cos(x)</code> <code>&gt;&gt;y1</code> <code>&gt;&gt;y2</code>	<i>Multiple lines can be typed in one line using 'append' (i.e. the semicolon ;). Notice how this suppresses the answer messages.</i>

## 4. Special symbols, Last line editing and Viewing variables

### Special symbols

Matlab has several predefined constants and symbols. For example  $\pi$  is given by `pi`. Another predefined symbol is `i` (and `j`) which represents  $\sqrt{-1}$  and is used for complex numbers. Matlab is not very strict about re-assigning variables, and will quite happily let you redefine `i` or `pi` or any other constant already declared. As a result care is required when writing complex expressions with many declarations.

Other symbols you may see when using Matlab are `Inf` which stands for infinity, and `NaN` which means ‘not a number’. This appears when a division by zero occurs.

### Exercise 3

Find out what the commands *ginput*, *date* and *clock* represent in Matlab.

### Last line editing

Matlab allows you to do last line editing, which means you can retrieve old command lines to save retyping things. This is done by using the *up* and *down* arrows on the computer keyboard in the Command window. If you want an old command beginning with a particular letter, type that letter and press the up arrow.

### Viewing variables

To view the variables that are being used in the current Matlab session, type `who`. Using `whos` gives a more detailed listing including variable size and type. It is also possible to use the command `ls` (short for ‘list’) to view all the files in the current directory. If you want to get rid of all variables in the Workspace and start again, use the command `clear`.



## 5. Accuracy and precision

All computations in Matlab are carried out in double precision. The *format* command can be used to change the way that numbers are displayed.

```
>> format long
```

```
>> format short
```

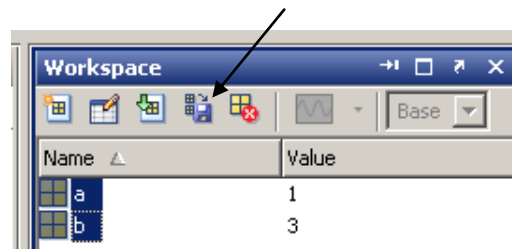
Occasionally the round-off error inherent in floating point computations may have an effect on the accuracy of your output. This is something to be aware of but rarely poses problems for the majority of engineering computations.

### Exercise 4

Experiment with the *format* command. How do you input numbers using the exponent notation?

## 6. Workspace and diary

The space where your variables are stored and computations carried out is the workspace (you see all variables stored in the 'base' workspace in the **workspace window**. Keep in mind that functions have a separate workspace that you don't see in the **workspace window**). This can be thought of as a temporary place to carry out calculations. The workspace has a browser which enables you to monitor the variables you have and their values. To open it, click Desktop on the menu bar and then on the Workspace option. You can save all variables saved in workspace in a \*.mat file use them later for further tasks or reference.



Calculations with Matlab can often be lengthy and involved. It is often possible that you wish to see exactly what you have done during a Matlab session. This can be achieved by using the *diary* command. The *diary* command creates a text file which contains all your actions generated in the command window. Before we can start using a diary, we must ensure that we are working in a directory (or folder) in which we are allowed to create a file. To start using the diary command to save to a file called 'mywork' for example, type

```
>> diary mywork
```

All the subsequent commands will then be written to the file mywork. To turn off the diary facility type:

```
>> diary off
```

During the course you may find it useful to use the diary facility to record your work. The contents of the diary file can be viewed in the Matlab Editor/debugger by typing

```
>> edit mywork
```

Note: to ensure that the most recent commands you have used have been recorded in the diary, turn off the diary facility before viewing the diary file.

## 7. Scalars, arrays and matrices

We discussed earlier of how to declare constants and variables. In fact all the examples we used were for scalars i.e. single numbers. In the majority of real engineering problems we will need to use vectors and matrices. These mathematical objects are represented in the computer as arrays. In fact, manipulating large arrays is one of the tasks for which computers are useful and Matlab especially suitable for.

### Declaring arrays

In general an array is a sequenced collection of variables. Vectors can be represented using a one dimensional array; Matrices, by using a two dimensional array. There are several ways of declaring arrays in Matlab, the simplest approach is to directly type the variables in the Command window. For example (please start your diary of this example)

```
>> a = [5 6 8 9 9]
```

will declare a one dimensional array,  $\mathbf{a}$ , having five elements. This array can be thought of as a row vector. Any element of an array can be addressed using the array index. For example to access the third element of the array  $\mathbf{a}$  we type

```
>> a(3)
```

which will return the numerical value assigned to the third element of the array, which in this case is 8. Numerical values can be assigned to the individual elements of  $\mathbf{a}$  using the assignment operator '='. In fact using the notation  $a(n)$ , where  $n$  is the index, we can treat the individual elements of the array as scalars. Try typing

```
>> n=4
```

and then

```
>> a(n)
```

to confirm that the answer is the fourth element of the array *a*. A commonly used array, is one with equally spaced increments. This can be created very efficiently in Matlab by stating the first element, the increment and the final element separated with colons. So for an array of integers between 10 and 20 we would type

```
>> b=10:1:20
```

```
b = 10 11 12 13 14 15 16 17 18 19 20
```

To create an array with unit increment in Matlab, we need only to type the first and last element i.e. `b=10:20`, unit spacing is assumed by default.) Often we wish to declare an array to represent the independent variable in mathematical problems. For example, in dynamics problems this would be the time-base. For example if we require a time-base from 0 to 5 seconds in steps of 0.1 seconds we could declare an array by typing

```
>> t=0:0.1:5
```

This array is 51 elements long, we will frequently need to use far longer arrays, to stop Matlab displaying the array in the command window, we can add a semi-colon to the end of the command

```
>> t=0:0.1:5;
```

This is very useful as displaying arrays that are created and the answers to every calculation that is performed can massively increase the time required to run a program. The arrays we have declared here are all in row format. We can transpose the array into a column using the transpose operator `'`. To see this, try typing

```
>> bt=b'
```

## Exercise 5

Create an array called time which goes from zero to 2 in steps of 0.1. Now switch off the diary and view it in the editor. Please also review the command *linspace* in this context.

## Vector calculations

As with scalars, multiplication of vectors uses the `*` command. However, there is now an additional function that can be used, the `.*` command. This command multiplies two vectors that are the same size element-by-element, i.e. if we define two equal length vectors `a` and `b` and then type

```
>> c=a*b
```

You probably received an error. Instead try

```
>> c=a.*b
```

then the first element in vector `c` will be the product of the first elements in vectors `a` and `b` and so on.

## Matrices

A matrix is a two dimensional array and can be declared in several ways, for example by typing

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

Here the semi-colons within the square brackets represent the end of each row. So the in mathematical terms, matrix `P` is

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

As with one-dimensional arrays, we can access individual elements of the matrix using indices. So, by typing

```
>> P(2,1)
```

we are asking to see the element of the array at the second row and third column. So we get the result

```
>> ans = 4
```

In a similar way we can use indices to assign numerical values to individual elements of the matrix. For example  $P(4,2)=67$  assigns the numerical value 67 to the matrix element at row four, column two. (Try this and then view matrix P). Please check also the meaning of commands *eye(3)* and *zeros(6,6)*

## Exercises 6

Run the Matlab demo on basic matrix operations. Then declare a matrix called Q, with 5 rows and 3 columns. Assign element  $Q(5,2)$  with the value 12, and assign all other elements equal to zero.

Use the Matlab help menu to find more about Matrix operations

## 8. Plotting graphs

An essential function in engineering computing is the ability to represent data graphically. In Matlab this is done using the plot command. Read the help given for the plot command. If we want to plot a sine function we can first create a time vector with suitable increment between the elements and then we can create an array of  $\sin(t)$  by typing

```
>> x=sin(t);
```

Now, we can plot the results by typing  $\text{plot}(t,x)$ . To get multiple plots we can use the hold command. Typing *hold on*, at the command prompt holds the current plot. Then if we create a second array  $y=\cos(t)$ ; and give the command  $\text{plot}(t,y)$ , then both plots appear together. To get different colour lines we can give the second plot command as  $\text{plot}(t,y,'g')$ , which plots the cos function in green.

Multiple plots can also be obtained by using the plot command directly i.e.  $\text{plot}(t,x,t,y)$ . To add a grid to the plot use the command *grid on*. Labels, titles and text can be added by using the commands *xlabel*, *ylabel*, *title*, *text* and *gtext*; for more than a single word use *quotes*. These can also be added using the menus on the plot window itself.

## 9. Polynomials

To represent a polynomial function in Matlab we define a coefficient array of the powers of the polynomial in descending order. For example, the polynomial  $x^3 + 5x^2 + 2x - 4$  can be represented as

```
>> p1=[1 5 2 -4]
```

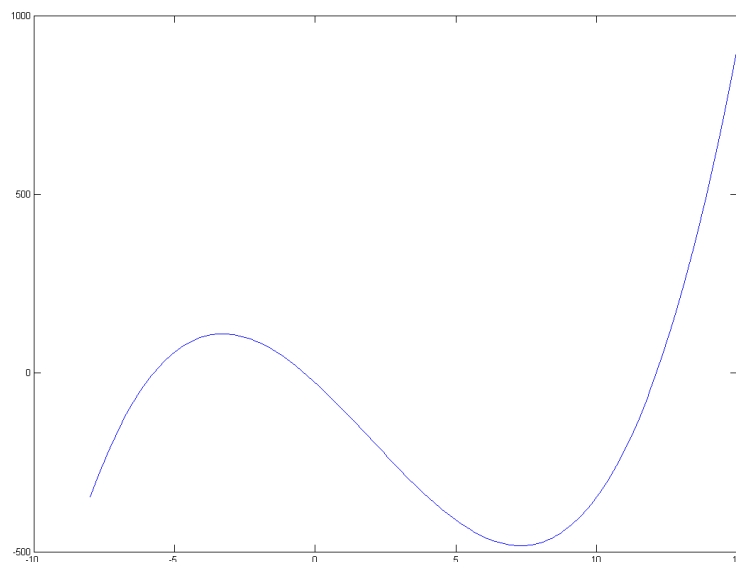
Polynomial functions can be evaluated for a certain range (or domain) of  $x$  values, by using the *polyval* command. For example if we define an array  $x = 0:0.1:10$ , then

```
>> y=polyval(p1,x);  
>> plot(x,y)
```

will plot the polynomial function defined by  $p1$  in the range  $0 < x < 10$ .

### Exercise 7


Use the command *roots* to find the roots of the equations polynomial  $x^3 - 6x^2 - 72x - 27$ . Plot the polynomial using an appropriate  $x$  interval. Use plot formatting options to complement the figure.

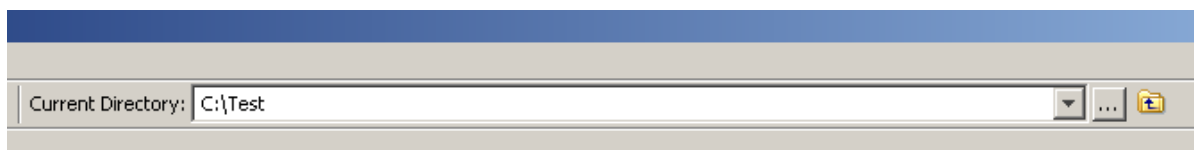




## 10. Write your first program

The command lines are only useful for small problems. In engineering practise, it is more efficient to write multiple commands into a single file (called M-file, i.e., macro file) and run these commands in one go. To start writing a program, open **File/New/Blank M-File**

To execute the M-File open *Debug/save and run*, click the icon  in the task menu or type the name of the file (without the extension .m) in the command window. Matlab looks for the m-file, and having found it compiles the code, and then executes it. Please make sure that your current directory path corresponds to the place where you save your M-files (e.g. C:/Test).



To save the M-File (in your directory) open File/Save as and name the program as *brlrobotics.m*

The following code will help you to understand the input from keyboard and output to screen. Add them into the *brlrobotics.m* M-file and see what is happening:

```
A=input ('Input a number=');  
  
fprintf ('The typed number as real %f\n',A);  
  
B=fix(A);  
  
fprintf ('The typed number as integer %d\n',B);
```

## 11.IF and FOR constructions

This section will help you to understand loop and if functions. For example you can use the For-loop to plot a 2D or 3D workspace of a robot by taking mechanism constraints (e.g. maximum joint positions) using IF functions into account (Example 2). Please add the following code to a m-file.

### Example 1:

#### **Loop control:**

```
for i=1:5
    fprintf('i=%d\n',i)
end
```

Then, add

```
n=10;
for i=1:n;
    j=i*2;
    fprintf('i=%d j=%d \n',i,j)
end;
```

Change n to 15 or any other values, try again.

Please try also the following

```
q=-100:10:100; %motion interval in y-direction
w=-100:10:100; %motion interval in x-direction

for i=1:length(q)
    for u=1:length(w)
        plot(q(i),w(u),'kx'),hold on %plot a 2D array
        xlabel('x-axis')
        ylabel('y-axis')
        zlabel('z-axis')
    end
end
```

#### **IF Control:**

This is useful for changing the flow of instructions (i.e. statements/lines) executed by the program.

```
a=1;
b=2;
if(a>b),
    fprintf('a is greater than b\n');
else
    fprintf('a is not greater than b\n');
end
```

then, change  $a=4$ , run it again.

## Example 2:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Calculation of 2D constant orientation workspace   %
%   for a Stewart Gough Platform (SGP)               %
%   using numerical method                             %
%   D.Raabe, 2006                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
clc;
deg2rad=pi/180;
close all;

%Define grid
q=-200:10:200; %motion interval in y-direction
w=-200:10:200; %motion interval in x-direction

for yy=q
    for xx=w
        %INVERSE KINEMATICS (IK)

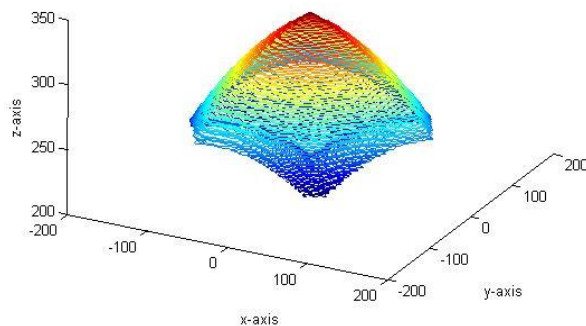
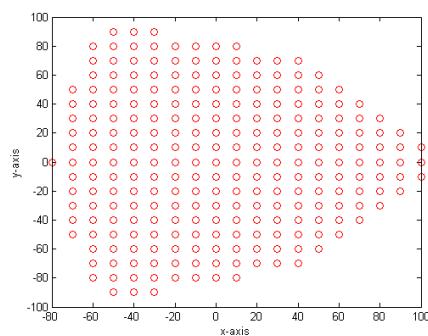
        %Define Cartesian IK Input (new end-effector position)
        %a=[x,y,z,yaw,pitch,roll];
        a=[x,y,z,(0*deg2rad),(0*deg2rad),(0*deg2rad)];

        %solve IK for SGP here
        %...
        %...
        %output l=[l1,l2,l3,l4,l5,l6];

        %Define Workspace constraints e.g. max. & min. actuator length
        lmax=355;
        lmin=255;

        %Evaluate and Plot workspace
        if l>lmin & l<lmax
            plot(a(1),a(2),'ro'),hold on
        end

        xlabel('x-axis')
        ylabel('y-axis')
        zlabel('z-axis')
    end
end
```



## 12. Programming a graphical user interface

GUIDE, the MATLAB Graphical User Interface development environment, provides a set of tools for creating GUIs. These tools greatly simplify the process of laying out and programming a user interface.

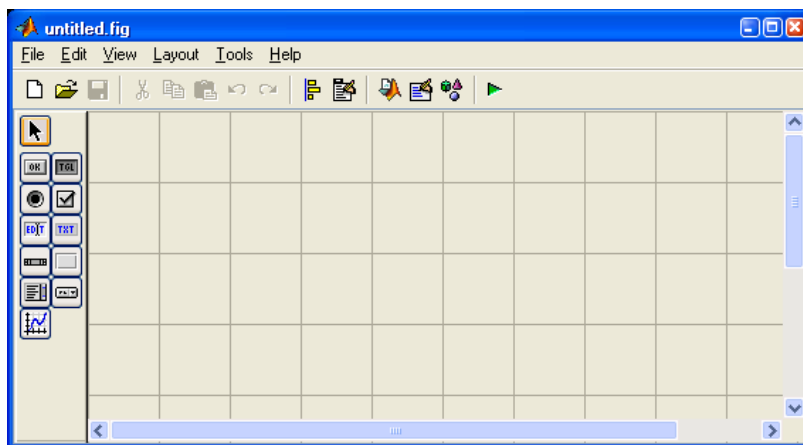
### Example

1) Open a blank sheet

```
>>guide
```

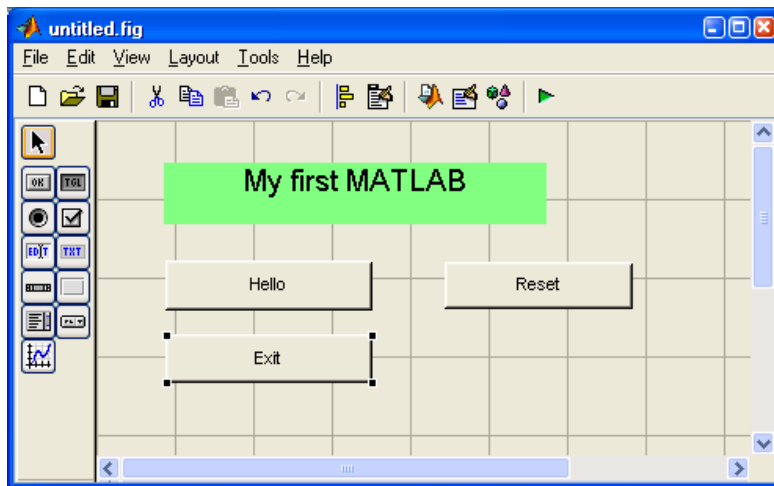
Choose blank GUI (Default)

A blank graphical user interface is shown below



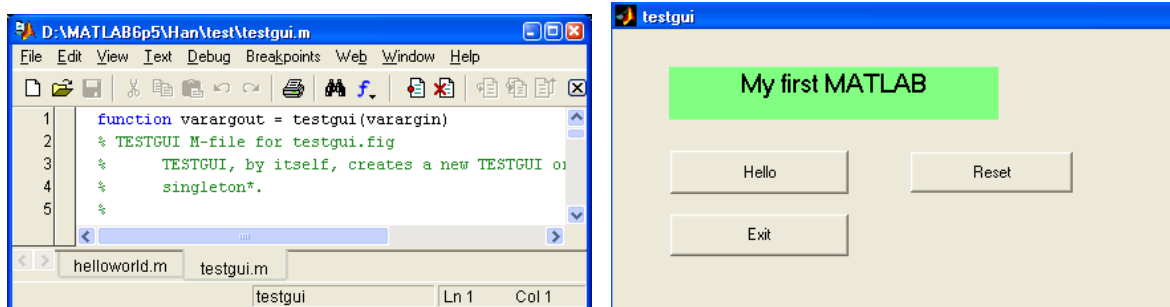
2) Drag a text label from the control box to the empty form, double click it, in the pop-up window, find 'String' item and change its content from 'Static text' to 'My first MATLAB', font size to 14, background colour to green

3) Drag three push buttons to the form and name them using 'String' as, "Hello", "Reset" and "Exit" and change their 'Tag' to pushbuttonHello, pushbuttonReset and pushbuttonExit respectively.

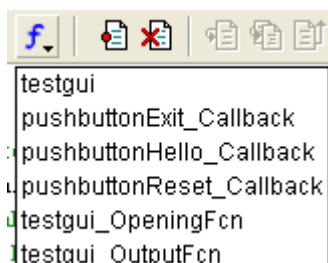


4) Tools-> run, name it as **testgui**

Two windows will pop up: one with matlab code and one with user interface window.



5) Goto the code window, click 'f' icon to pop up a list with function names



Choose 'pushbuttonHello\_Callback' to go to code

***function pushbuttonHello\_Callback(hObject, eventdata, handles)***

***% hObject*** handle to pushbuttonHello (see GCBO)

***% eventdata*** reserved - to be defined in a future version of MATLAB

***% handles*** structure with handles and user data (see GUIDATA)

This function is the code that is executed when the user clicks the Hello button of your gui with the mouse. Do not worry about the comments (stuff after the % characters). Handles are a computing concept – a way of ‘getting hold of/access to’ (hence using) particular data or functions using some form of ID for those data/functions.

Type in

***set(handles.text1,'string','hello');***

which changes the character string displayed in the text label called text1 to ‘hello’

Do the same for pushbuttonReset\_Callback but instead typing in

***set(handles.text1,'string','My first MATLAB');***

6) Click ‘f’ icon to pop up ‘pushbuttonExit\_Callback’

Type in

***delete(handles.figure1)***

7) Go to ***testgui.fig***, then tools->run. Now test all three buttons.

## 13. Appendix

### Input and output

Meaning	Function in MATLAB
input	A=input(' Type in the value of A:')
output	fprintf(format,variable) e.g., fprintf('variable A =%f',A)
format	\n newline; %f for real values %d, for integers

### Logical control

	MATLAB	Examples
<b>1</b>	if <condition>, <program 1> end	If a>1, b=2; end
<b>2</b>	if <condition>, <program 1> else <program 2> end	If a>1, b=2; else b=3; end
<b>3</b>	if <condition>, <program 1> elseif <condition 2> <program 2> elseif <condition 3> <program 3> .... end	If a==1, b=2; elseif a==2 b=3; elseif a==3 b=5; end

## Condition operator

Relational operator	MATLAB
Less than	<
Greater than	>
Less than or equal	<=
Greater than or equal	>=
Equal	==
No equal	~=
<b>Logical operator</b>	
Not	^
And	&
Or	!
<b>Example</b>	<i>If (x&gt;6) &amp; (x&lt;10) ,     x=x+6 end</i>

## Looping control

	MATLAB	Example	Example
<b>1</b>	for program 1 end	For i=1:2 C=2*i end	For i=1 to 2 c=2*i end
<b>2</b>	while (condition) program end	while (i<4) y=4*i i=i+1 end	While (i<4) y=4*i i=i+1 loop