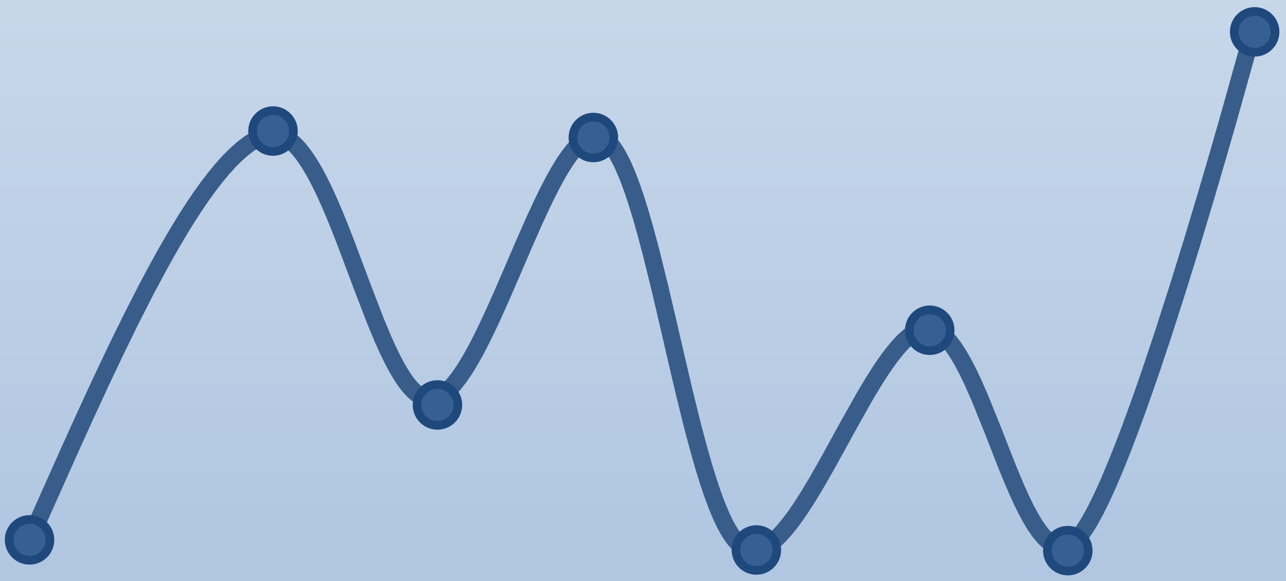
Introduction to MATLAB

Hans-Petter Halvorsen



https://www.halvorsen.blog

Introduction to MATLAB

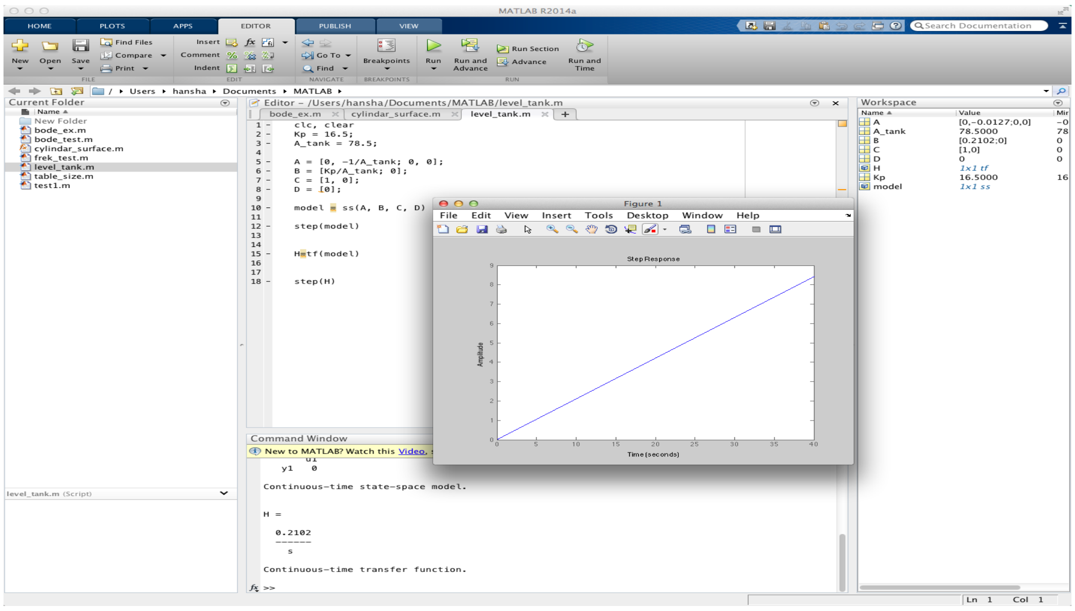
University of South-Eastern Norway

MATLAB for Students

MATLAB

Introduction to MATLAB

Hans-Petter Halvorsen, 2022.08.17



```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

```
       u1
   y1   0

Continuous-time state-space model.


H =

  0.2102
  ------
    s

Continuous-time transfer function.

>>
```

http://www.halvorsen.blog

MATLAB
The Language of Technical Computing

Preface

In this MATLAB Course, you will learn basic MATLAB and how to use MATLAB in Control and Simulation applications. An introduction to Simulink and other Tools will also be given.

MATLAB is a tool for technical computing, computation and visualization in an integrated environment. MATLAB is an abbreviation for MATrix LABoratory, so it is well suited for matrix manipulation and problem solving related to Linear Algebra, Modelling, Simulation and Control applications.

This is a self-paced course based on this document and some short videos on the way. This document contains lots of examples and self-paced tasks that the users will go through and solve on their own. The user may go through the tasks in this document in their own pace and the instructor will be available for guidance throughout the course.

The MATLAB Course consists of 3 parts:

1. Introduction to MATLAB
2. Modelling, Simulation and Control
3. Simulink and Advanced Topics

In part 1 you will be familiar with the MATLAB environment and learn basic MATLAB programming.

The course consists of lots of Tasks you should solve while reading this course manual and watching the videos referred to in the text.

Make sure to bring your **headphones** for the videos in this course. The course consists of several short videos that will give you an introduction to the different topics in the course.

Prerequisites

You should be familiar with undergraduate-level mathematics and have experience with basic computer operations.

What is MATLAB? MATLAB is a tool for technical computing, computation and visualization in an integrated environment. MATLAB is an abbreviation for MATrix LABoratory, so it is well suited for matrix manipulation and problem solving related to Linear Algebra.

MATLAB is developed by The MathWorks. MATLAB is a short-term for MATrix LABoratory. MATLAB is in use world-wide by researchers and universities. For more information, see www.mathworks.com

For more information about MATLAB, etc., please visit http://www.halvorsen.blog

Online MATLAB Resources:

MATLAB:

http://www.halvorsen.blog/documents/programming/matlab/

MATLAB Basics:

http://www.halvorsen.blog/documents/programming/matlab/matlab_basics.php

Modelling, Simulation and Control with MATLAB:

http://www.halvorsen.blog/documents/programming/matlab/matlab_mic.php

MATLAB Videos:

http://www.halvorsen.blog/documents/video/matlab_basics_videos.php

MATLAB for Students:

http://www.halvorsen.blog/documents/teaching/courses/matlab.php

On these web pages you find video solutions, complete step by step solutions, downloadable MATLAB code, additional resources, etc.

Table of Contents

1 Introduction

Additional Resources, Videos, etc. are available from:

http://www.halvorsen.blog/documents/programming/matlab

Part I: Introduction to MATLAB consists of the following topics:

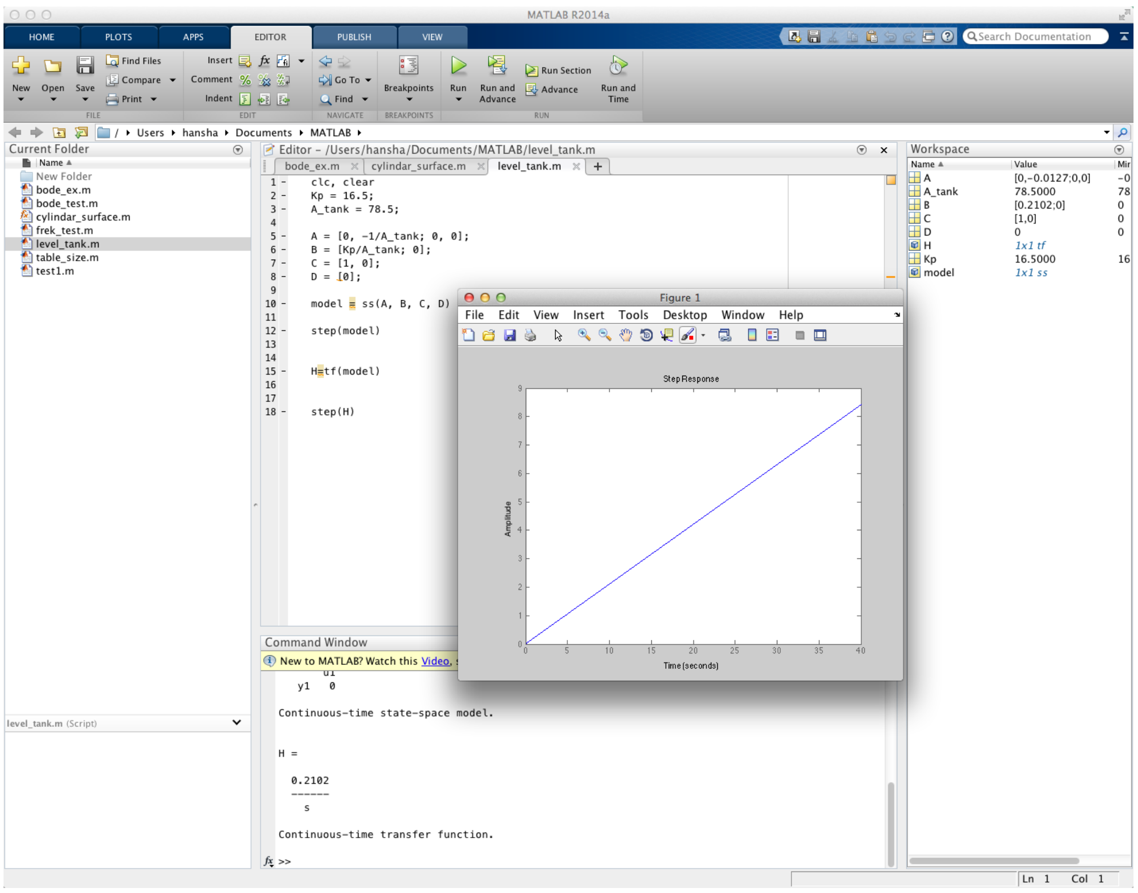
- The MATLAB Environment
- Using the Help System in MATLAB
- MATLAB Basics
- Linear Algebra; Vectors and Matrices
- M files; Scripts and User-defined functions
- Plotting
- Flow Control and Loops; For and While Loops, If and Case statements
- Mathematics
- Additional Tasks

2 The MATLAB Environment

The MATLAB Environment consists of the following main parts:

- Command Window
- Command History
- Workspace
- Current Folder
- Editor

Below we see the MATLAB environment:



```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

Name ▲	Value
A	[0,-0.0127;0,0]
A_tank	78.5000
B	[0.2102;0]
C	[1,0]
D	0
H	*1x1 tf*
Kp	16.5000
model	*1x1 ss*

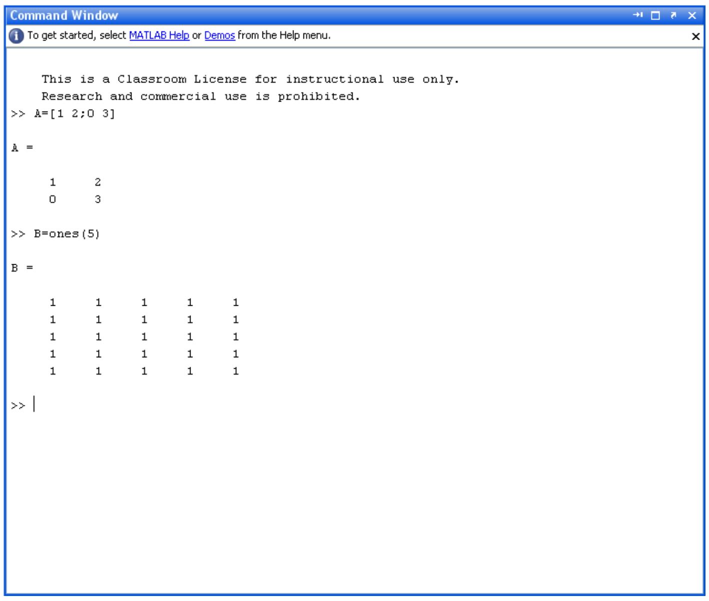
```
       u1
   y1   0

Continuous-time state-space model.


H =

  0.2102
  ------
    s

Continuous-time transfer function.

>>
```

Before you start, you should watch the video “**Working in the Development Environment**”.

The video is available from:
https://www.halvorsen.blog/documents/teaching/courses/matlab/matlab1.php

2.1 Command Window

The **Command Window** is the main window in MATLAB. Use the Command Window to enter variables and to run functions and M-files scripts (more about m-files later).


Command Window
To get started, select MATLAB Help or Demos from the Help menu.

```
  This is a Classroom License for instructional use only.
  Research and commercial use is prohibited.
>> A=[1 2;0 3]

A =

     1     2
     0     3

>> B=ones(5)

B =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1

>> |
```

You type all your commands after the command Prompt "**>>**", e.g., defining the following matrix:

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$$

The MATLAB syntax is as follows:

```
>> A = [1 2;0 3]
```

Or

```
>> A = [1,2;0,3]
```

If you, for an example, want to find the answer to

$$a + b, where\ a = 4, b = 3$$

Type like this:

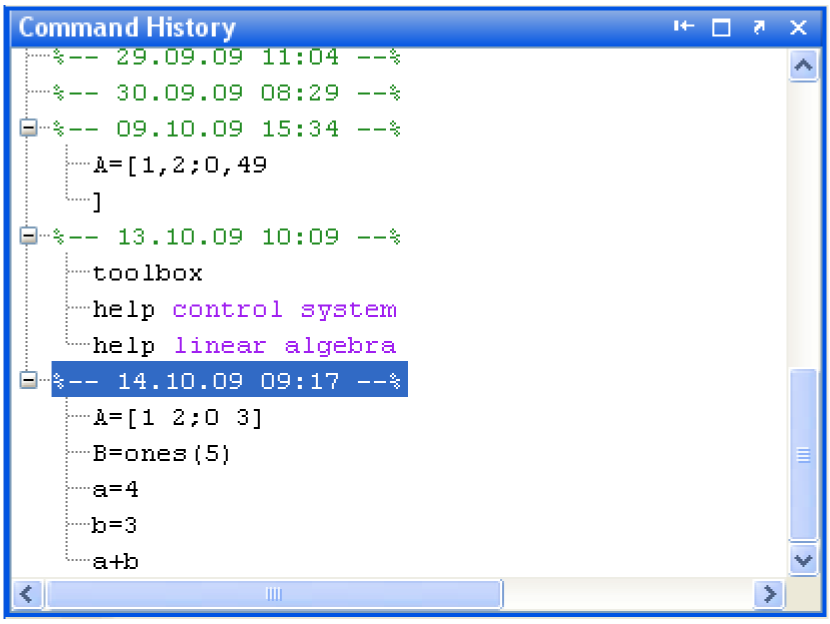
```
>>a = 4
>>b = 3
>>a + b
```

MATLAB then responds:

```
ans =
     7
```

2.2 Command History

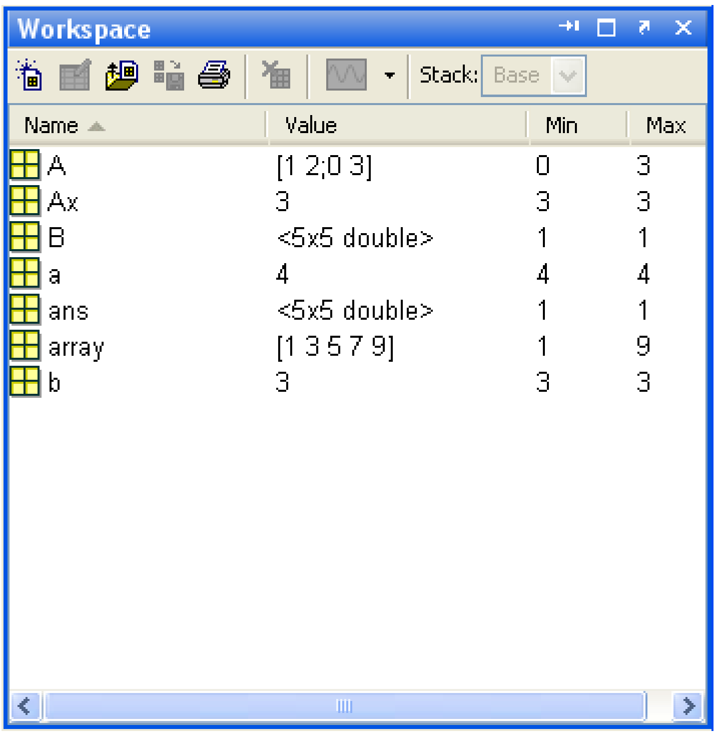
Statements you enter in the Command Window are logged in the **Command History**. From the Command History, you can view and search for previously run statements, as well as copy and execute selected statements. You can also create an M-file from selected statements.



```
%-- 29.09.09 11:04 --%
%-- 30.09.09 08:29 --%
%-- 09.10.09 15:34 --%
  A=[1,2;0,49
  ]
%-- 13.10.09 10:09 --%
  toolbox
  help control system
  help linear algebra
%-- 14.10.09 09:17 --%
  A=[1 2;0 3]
  B=ones(5)
  a=4
  b=3
  a+b
```

2.3 Workspace

The **Workspace** window list all your variables used as long you have MATLAB opened.



Name	Value	Min	Max
A	[1 2;0 3]	0	3
Ax	3	3	3
B	<5x5 double>	1	1
a	4	4	4
ans	<5x5 double>	1	1
array	[1 3 5 7 9]	1	9
b	3	3	3

You could also use the following command

```
>>who
```

This command list all the commands used

or

```
>>whos
```

This command lists all the command with the current values, dimensions, etc.

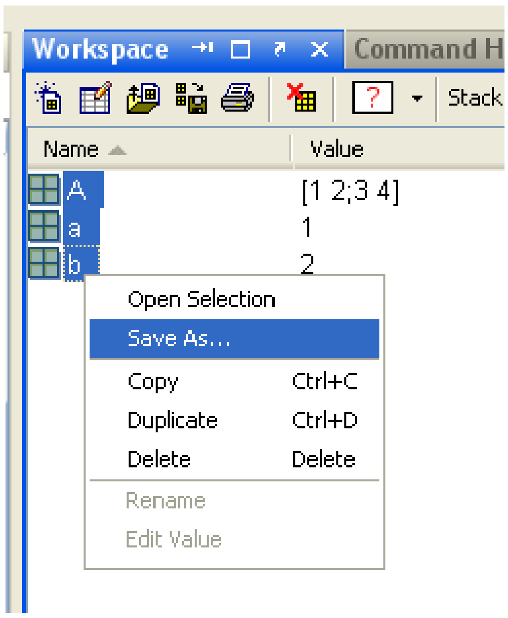
The command `clear`, will clear all the variables in your workplace.

```
>>clear
```

Save your data:

You may also save all your variables and data to a text file (**.mat** file), this is useful if you want to save your data and use it for later.

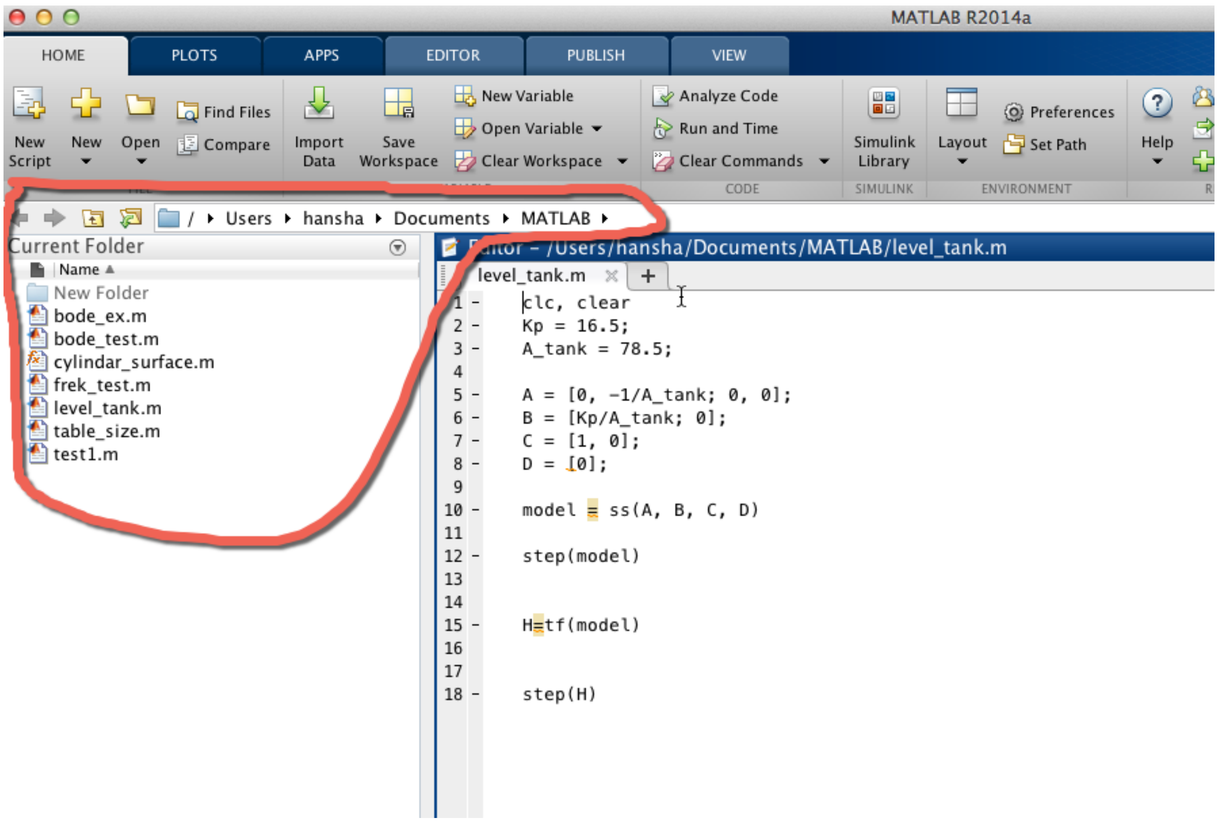
Select the variables you want to save and right-click and select "Save As…":



MATLAB also have commands for this: **save**/**load** and **diary**.

2.4 Current Folder

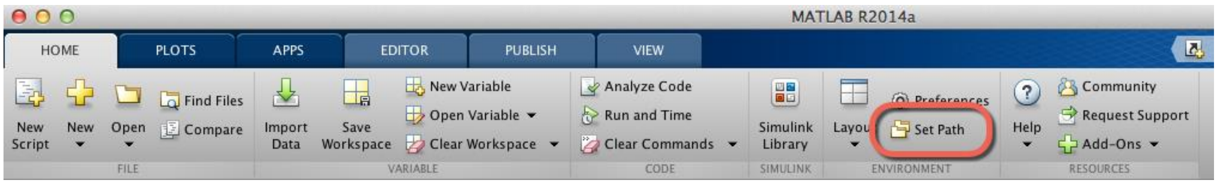
The “**Current Folder**” window lists all m files, etc. available in the current directory.



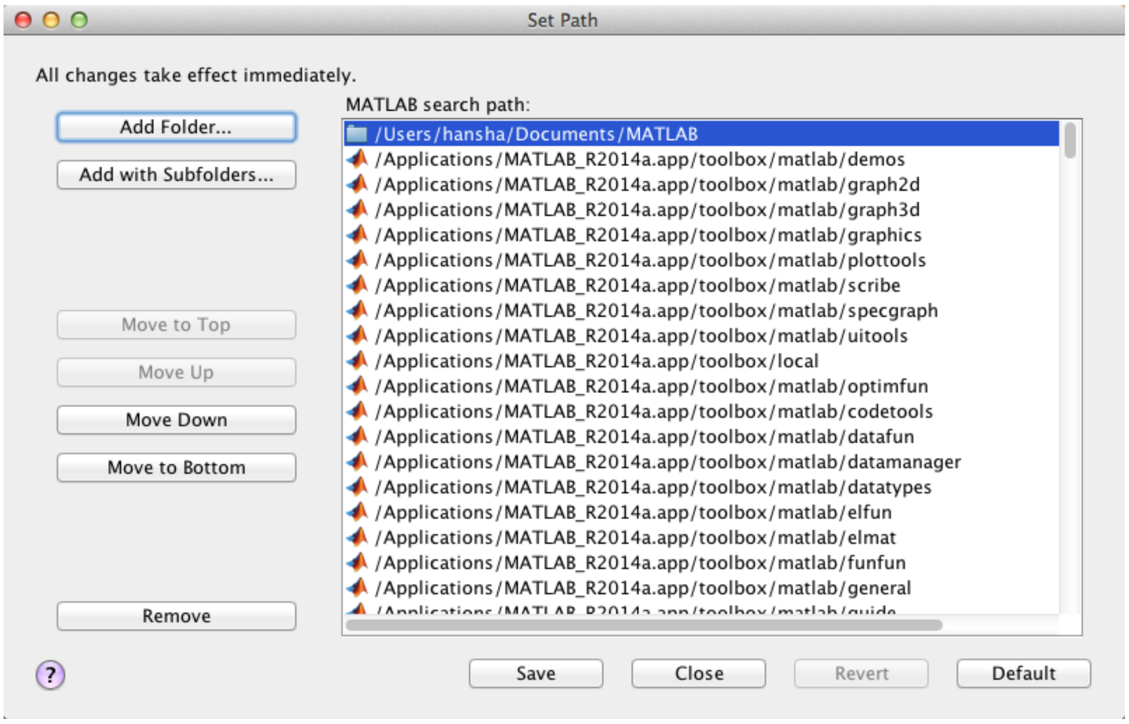
```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

You should set your working folder as the Current Directory or set your working folder as part of the search path, if you don't MATLAB will not find your files.

Search Path:



You need to use this if you want MATLAB to find your scripts and functions you want to use.



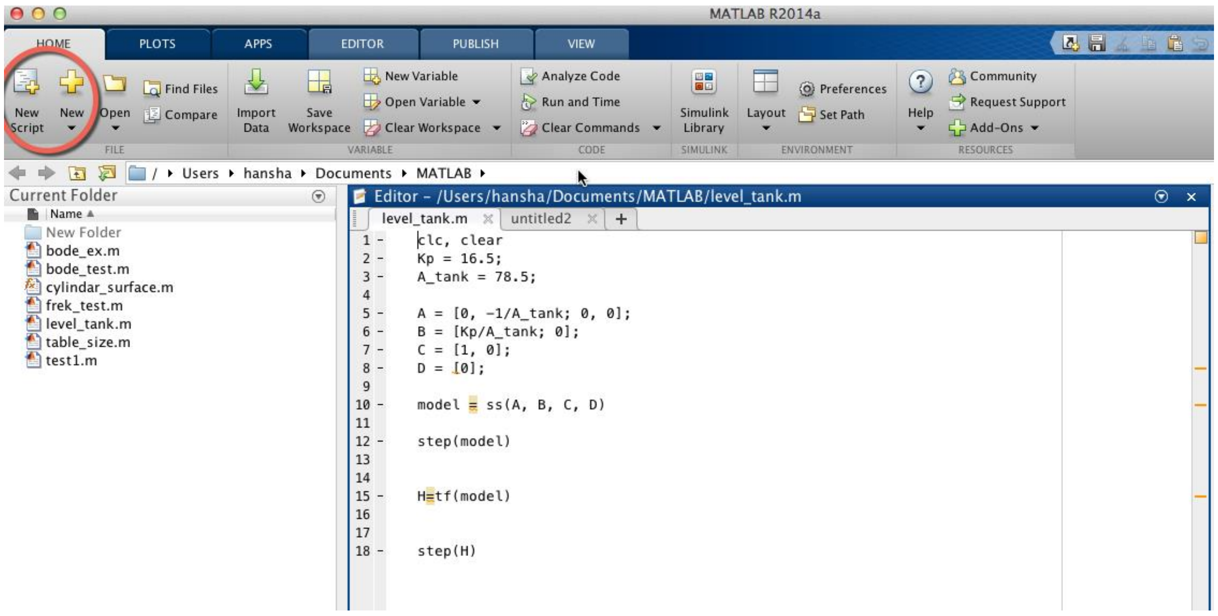
All changes take effect immediately.

MATLAB search path:

```
/Users/hansha/Documents/MATLAB
/Applications/MATLAB_R2014a.app/toolbox/matlab/demos
/Applications/MATLAB_R2014a.app/toolbox/matlab/graph2d
/Applications/MATLAB_R2014a.app/toolbox/matlab/graph3d
/Applications/MATLAB_R2014a.app/toolbox/matlab/graphics
/Applications/MATLAB_R2014a.app/toolbox/matlab/plottools
/Applications/MATLAB_R2014a.app/toolbox/matlab/scribe
/Applications/MATLAB_R2014a.app/toolbox/matlab/specgraph
/Applications/MATLAB_R2014a.app/toolbox/matlab/uitools
/Applications/MATLAB_R2014a.app/toolbox/local
/Applications/MATLAB_R2014a.app/toolbox/matlab/optimfun
/Applications/MATLAB_R2014a.app/toolbox/matlab/codetools
/Applications/MATLAB_R2014a.app/toolbox/matlab/datafun
/Applications/MATLAB_R2014a.app/toolbox/matlab/datamanager
/Applications/MATLAB_R2014a.app/toolbox/matlab/datatypes
/Applications/MATLAB_R2014a.app/toolbox/matlab/elfun
/Applications/MATLAB_R2014a.app/toolbox/matlab/elmat
/Applications/MATLAB_R2014a.app/toolbox/matlab/funfun
/Applications/MATLAB_R2014a.app/toolbox/matlab/general
/Applications/MATLAB_R2014a.app/toolbox/matlab/guide
```

2.5 Editor

The **Editor** is used to create scripts and m-files. Click the "New Script" button in the Toolbar



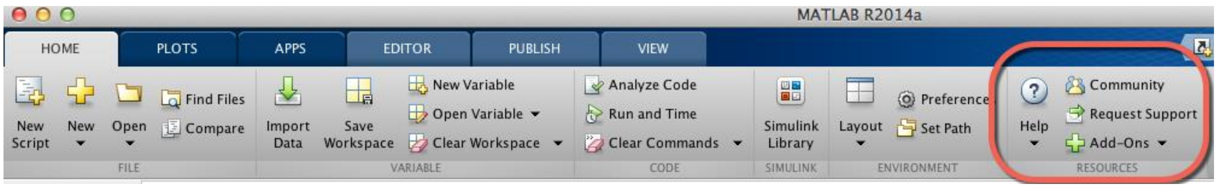
```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

When you learn about m-files (scripts and functions) in a later chapter you will be using this editor to enter your commands and save them.

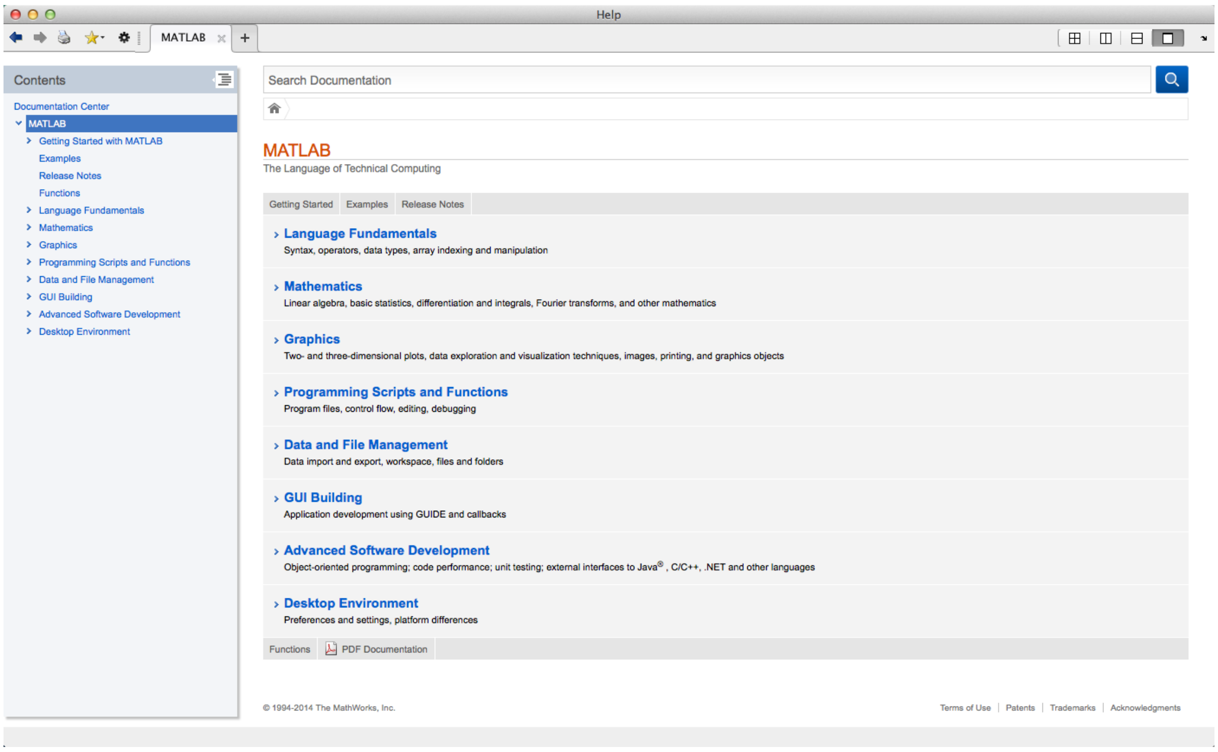
Note! In the beginning of the course (chapter 1-5) we will only use the Command Window. In chapter 6 we will start using the Editor.

3 Using the Help System in MATLAB

The Help system in MATLAB is quite comprehensive, so make sure you are familiar with how the help system works.



when clicking the "Help" button, the following window appears:



MATLAB
The Language of Technical Computing

Getting Started | Examples | Release Notes

› **Language Fundamentals**
Syntax, operators, data types, array indexing and manipulation

› **Mathematics**
Linear algebra, basic statistics, differentiation and integrals, Fourier transforms, and other mathematics

› **Graphics**
Two- and three-dimensional plots, data exploration and visualization techniques, images, printing, and graphics objects

› **Programming Scripts and Functions**
Program files, control flow, editing, debugging

› **Data and File Management**
Data import and export, workspace, files and folders

› **GUI Building**
Application development using GUIDE and callbacks

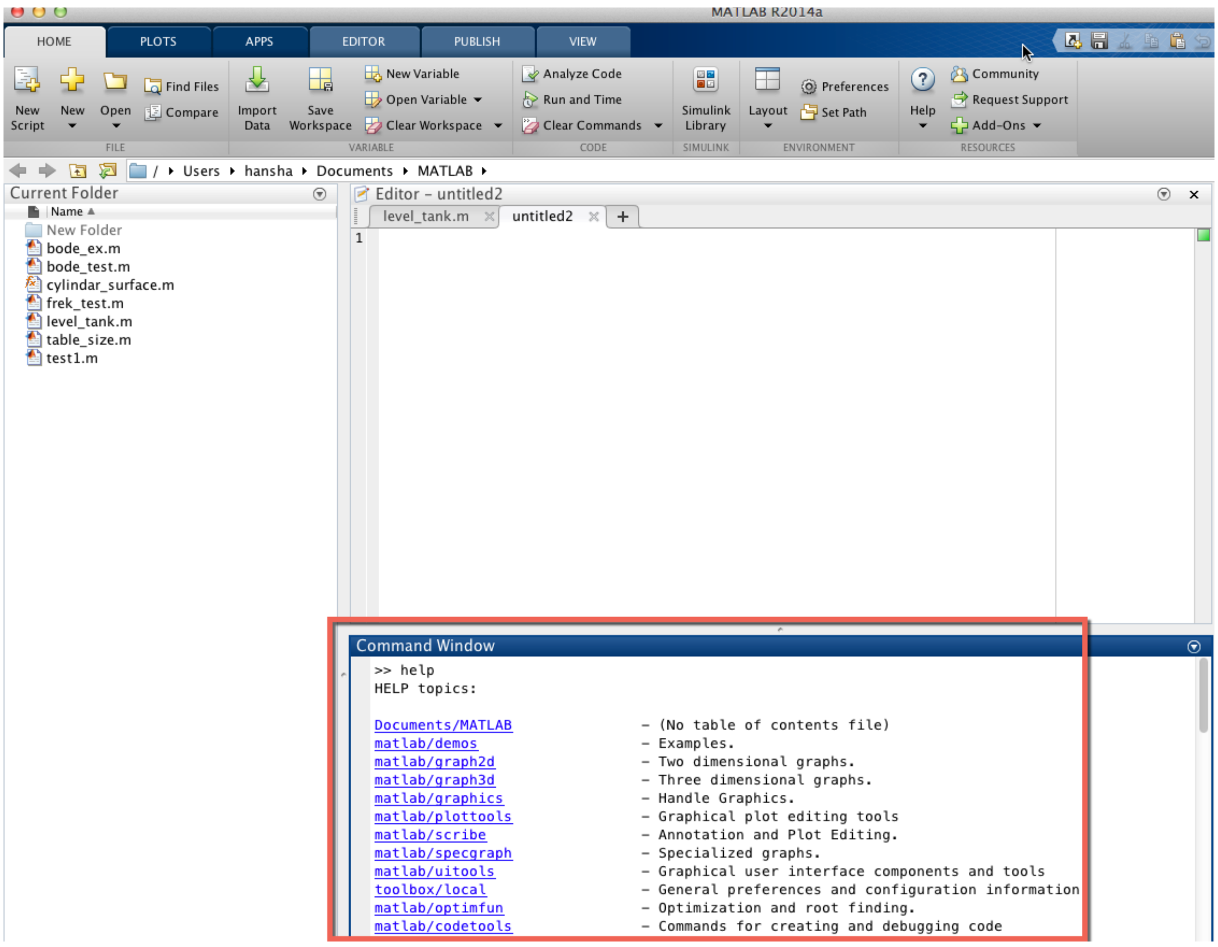
› **Advanced Software Development**
Object-oriented programming; code performance; unit testing; external interfaces to Java®, C/C++, .NET and other languages

› **Desktop Environment**
Preferences and settings, platform differences

Functions | PDF Documentation

© 1994-2014 The MathWorks, Inc.   Terms of Use | Patents | Trademarks | Acknowledgments

You may also type "**Help**" in the Command window:



```
>> help
HELP topics:

Documents/MATLAB              - (No table of contents file)
matlab/demos                  - Examples.
matlab/graph2d                - Two dimensional graphs.
matlab/graph3d                - Three dimensional graphs.
matlab/graphics               - Handle Graphics.
matlab/plottools              - Graphical plot editing tools
matlab/scribe                 - Annotation and Plot Editing.
matlab/specgraph              - Specialized graphs.
matlab/uitools                - Graphical user interface components and tools
toolbox/local                 - General preferences and configuration information
matlab/optimfun               - Optimization and root finding.
matlab/codetools              - Commands for creating and debugging code
```

MATLAB answers with links to lots of Help topics. You may also type more specific, e.g., "**Help elfun**" (Elementary Math Functions), and MATLAB will list all functions according to the specific category.

If you type "**help <functionname>**" you will get specific help about this function.

You may also type "**doc <topic>**" to open the Help window on the specific topic of interest.

Searching:

We can use the **help** keyword when we want to get help for a specific function, but if we want to search for all functions, etc. with a specific keyword you may use the **lookfor** command.

Example:

```
lookfor plot
```

[End of Example]

4 MATLAB Basics

Before you start, you should watch the video "**Getting Started with MATLAB**"

The video is available from:
https://www.halvorsen.blog/documents/teaching/courses/matlab/matlab1.php

4.1 Basic Operations

Variables:

Variables are defined with the assignment operator, "**=**". MATLAB is dynamically typed, meaning that variables can be assigned without declaring their type, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

Example:

```
>> x = 17
x =
 17
>> x = 'hat'
x =
hat
>> x = [3*4, pi/2]
x =
   12.0000    1.5708
>> y = 3*sin(x)
y =
   -1.6097    3.0000
```

[End of Example]

Note! MATLAB is case sensitive! The variables x and X are not the same.

Note! Unlike many other languages, where the semicolon is used to terminate commands, in MATLAB the **semicolon** serves to suppress the output of the line that it concludes.

```
>> a=5
a =
     5
>> a=6;
>>
```

As you see, when you type a semicolon (;) after the command, MATLAB will not respond. This is very useful because sometimes you want MATLAB to respond, while in other situations that is not necessary.

Built-in constants:

MATLAB have several built-in constants. Some of them are explained here:

Name	Description
i, j	Used for complex numbers, e.g., z=2+4i
pi	π
inf	∞, Infinity
NaN	Not A Number. If you, e.g., divide by zero, you get NaN

Naming a Variable Uniquely:

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

```
which  -all  variablename
```

Example:

```
>> which -all pi

built-in (C:\Matlab\R2007a\toolbox\matlab\elmat\pi)
```

You may also use the **iskeyword** command. This command causes MATLAB to list all reserved names.

```
>> iskeyword
ans =
    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'persistent'
    'return'
    'switch'
    'try'
    'while'
```

Note! You cannot assign these reserved names as your variable names.

Note! MATLAB allows you to reassign built-in function names as variable names, but that is not recommended! – so be carefully when you select the name of your variables!

Example:

```
>> sin=4
sin =
     4
>> sin(3)
??? Index exceeds matrix dimensions.
```

In this example you have defined a variable "sin" – but "sin" is also a built-in function – and this function will no longer work!

If you accidently do so, use the **clear** command to reset it back to normal.

[End of Example]

Task 1: Basic Operations

Type the following in the Command window:

```
>>y=16;
>>z=3;
>>y+z
```

Note! When you use a semicolon, no output will be displayed. Try the code above with and without semicolon.

Note! Some functions display output even if you use semicolon, like ***plot***, etc.

Other basic operations are:

```
>>16-3
>>16/3
>>16*3
```

→ Try them.

[End of Task]

Built-in Functions:

Here are some descriptions for the most used basic built-in MATLAB functions.

Function	Description	Example
help	MATLAB displays the help information available	`>>help`
help <function>	Display help about a specific function	`>>help plot`

who, whos	who lists in alphabetical order all variables in the currently active workspace.	`>>who`
		`>>whos`

clear	Clear variables and functions from memory.	`>>clear`
		`>>clear x`

size	Size of arrays, matrices	`>>x=[1 2 ; 3 4];`
		`>>size(A)`

length	Length of a vector	`>>x=[1:1:10];`
		`>>length(x)`

format	Set output format	

disp	Display text or array	`>>A=[1 2;3 4];`
		`>>disp(A)`

plot	This function is used to create a plot	`>>x=[1:1:10];`
		`>>plot(x)`
		`>>y=sin(x);`
		`>>plot(x,y)`

clc	Clear the Command window	`>>cls`

rand	Creates a random number, vector or matrix	`>>rand`
		`>>rand(2,1)`

max	Find the largest number in a vector	`>>x=[1:1:10]`
		`>>max(x)`

min	Find the smallest number in a vector	`>>x=[1:1:10]`
		`>>min(x)`

mean	Average or mean value	`>>x=[1:1:10]`
		`>>mean(x)`

std	Standard deviation	`>>x=[1:1:10]`
		`>>std(x)`

Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

Task 2: Statistics functions

Create a random vector with 100 random numbers between 0 and 100. Find the minimum value, the maximum value, the mean and the standard deviation using some of the built-in functions in MATLAB listed above.

[End of Task]

4.2 Arrays; Vectors and Matrices

Before you start, you should watch the video "**Working with Arrays**".

The video is available from:
https://www.halvorsen.blog/documents/teaching/courses/matlab/matlab1.php

Matrices and vectors (Linear Algebra) are the basic elements in MATLAB and also the basic elements in control design theory. So, it is important you know how to handle vectors and matrices in MATLAB.

A general matrix A may be written like this:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in R^{nxm}$$

In MATLAB we type vectors and matrices like this:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
>> A = [1 2; 3 4]
A =  1     2

    3     4
```

or:

```
>> A = [1, 2; 3, 4]
A =  1     2

    3     4
```

→ To separate rows, we use a semicolon ";"

→ To separate columns, we use a comma "," or a space " ".

To get a specific part of a matrix, we can type like this:

```
>> A(2,1)
ans =
    3
```

or:

```
>> A(:,1)
ans =
     1
     3
```

or:

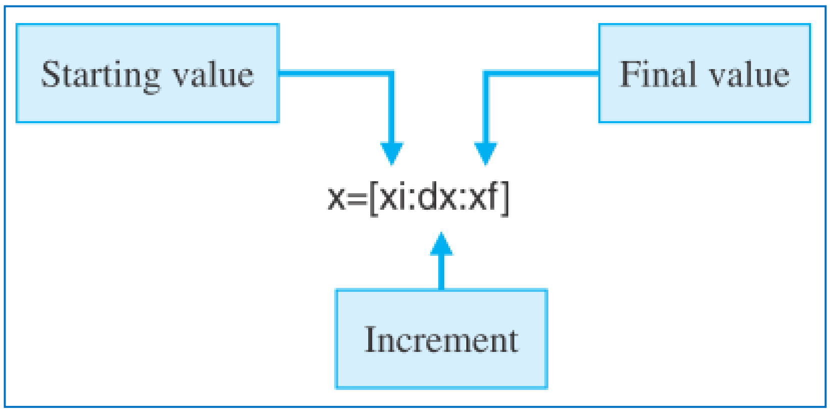
```
>> A(2,:)
ans =
     3     4
```

From 2 vectors x and y we can create a matrix like this:

```
>> x = [1; 2; 3];
>> y = [4; 5; 6];
>> B = [x y]
B =   1     4
     2     5
     3     6
```

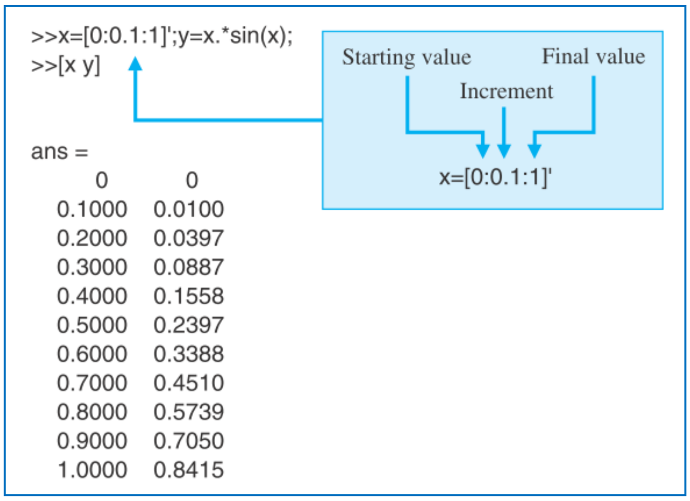
4.2.1 Colon Notation

The "**colon notation**" is very useful for creating vectors:



Example:

This example shows how to use the colon notation creating a vector and do some calculations.



```
>>x=[0:0.1:1]';y=x.*sin(x);
>>[x y]

ans =
        0         0
   0.1000    0.0100
   0.2000    0.0397
   0.3000    0.0887
   0.4000    0.1558
   0.5000    0.2397
   0.6000    0.3388
   0.7000    0.4510
   0.8000    0.5739
   0.9000    0.7050
   1.0000    0.8415
```

[End of Example]

Task 3: Vectors and Matrices

Type the following vector in the Command window:

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Type the following matrix in the Command window:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

Type the following matrix in the Command window:

$$C = \begin{bmatrix} -1 & 2 & 0 \\ 4 & 10 & -2 \\ 1 & 0 & 6 \end{bmatrix}$$

→ Use Use MATLAB to find the value in the second row and the third column of matrix C.

→ Use MATLAB to find the second row of matrix C.

→ Use MATLAB to find the third column of matrix C.

[End of Task]

Deleting Rows and Columns:

You can delete rows and columns from a matrix using just a pair of square brackets [].

Example:

Given:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

To delete the second column of a matrix A, use:

```
>>A=[0 1; -2 -3];
>>A(:,2) = []
A =
     0
    -2
```

[End of Example]

4.3 Tips and Tricks

Naming conversions:

When creating variables and constants, make sure you create a name that is not already exists in MATLAB. Note also that MATLAB is case sensitive! The variables x and X are not the same.

Use the which command to check if the name already exists: `which -all <your name>`

Example:

```
>> which -all sin
built-in (C:\Matlab\R2007a\toolbox\matlab\elfun\@double\sin)  % double method
built-in (C:\Matlab\R2007a\toolbox\matlab\elfun\@single\sin)  % single method
```

Large or small numbers:

If you need to write large or small numbers, like $2\,x\,10^5$, $7.5\,x\,10^{-8}$you can use the "e" notation, e.g.:

```
>> 2e5
ans =
     200000
>> 7.5e-8
ans =
  7.5000e-008
```

Line Continuation:

For large arrays, it may be difficult to fit one row on one command line. We may then split the row across several command lines by using the line continuation operator "…".

Example:

```
>> x=[1 2 3 4 5 ...

6 7 8 9 10]

x =

    1     2     3     4     5     6     7     8     9    10
```

Multiple commands on same line:

It is possible to type several commands on the same line. In some cases this is a good idea to save space.

Example:

```
>> x=1,y=2,z=3
x =
    1
y =
    2
z =
    3
```

4.3.1 Array Operations

We have the following basic matrix operations:

+	Addition
−	Subtraction
*	Multiplication
/	Division
^	Power

The basic matrix operations can be modified for element-by-element operations by preceding the operator with a period. The modified operations are known as **array operations**.

Given

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Then

$$A.* B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

The elements of A.*B are the products of the corresponding elements of A and B.

We have the following array operators:

+	Addition
−	Subtraction
.*	Multiplication
./	Division
.^	Power

Example:

```
>> A = [1; 2; 3]
A =
     1
     2
     3
>> B = [-6; 7; 10]
B =
    -6
     7
    10
>> A*B
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> A.*B
```

```
ans =
   -6
   14
   30
```

[End of Example]

5 Linear Algebra; Vectors and Matrices

Linear Algebra is a branch of mathematics concerned with the study of matrices, vectors, vector spaces (also called linear spaces), linear maps (also called linear transformations), and systems of linear equations.

MATLAB are well suited for Linear Algebra. This chapter assumes you have some basic understanding of Linear Algebra and matrices and vectors.

Here are some useful functions for Linear Algebra in MATLAB:

Function	Description	Example
rank	Find the rank of a matrix. Provides an estimate of the number of linearly independent rows or columns of a matrix A.	`>>A=[1 2; 3 4]` `>>rank(A)`
det	Find the determinant of a square matrix	`>>A=[1 2; 3 4]` `>>det(A)`
inv	Find the inverse of a square matrix	`>>A=[1 2; 3 4]` `>>inv(A)`
eig	Find the eigenvalues of a square matrix	`>>A=[1 2; 3 4]` `>>eig(A)`
ones	Creates an array or matrix with only ones	`>>ones(2)` `>>ones(2,1)`
eye	Creates an identity matrix	`>>eye(2)`
diag	Find the diagonal elements in a matrix	`>>A=[1 2; 3 4]` `>>diag(A)`

Type "**help matfun**" (Matrix functions - numerical linear algebra) in the Command Window for more information, or type "**help elmat**" (Elementary matrices and matrix manipulation).

You may also type "**help <functionname>**" for help about a specific function.

Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

5.1 Vectors

Given a vector x:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in R^n$$

Example:

Given:

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
>> x=[1; 2; 3]
x =
     1

     2

     3
```

The **Transpose** of vector *x*:

$$x^T = [x_1 \quad x_2 \quad \cdots \quad x_n] \in R^{1xn}$$

```
>> x'
ans =
     1     2     3
```

[End of Example]

The **Length** of vector *x*:

$$\|x\| = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Example:

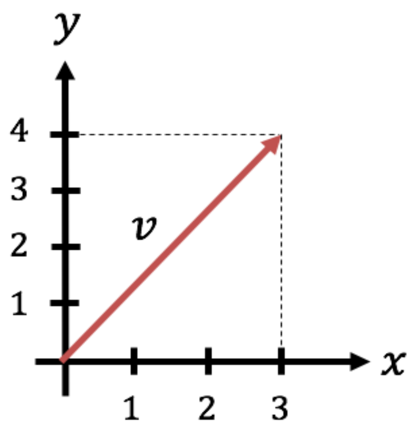
The length of a vector most makes sense for 2 or 3 dimensional vectors.

Given the following vector:

$$v = [3, 4]$$

Note! Sometimes you also see it like this: $\vec{v}$

It can be visualized like this:



In order to find the length of v we use Pythagoras like this:

$$|v| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

MATLAB:

```
>> v = [3,4]'
>> l = sqrt(3^2 + 4^2)
 l =
      5
```

Or using the general formula shown above (which works for any dimensions):

```
>> l = sqrt(v'*v)
l =
     5
```

Note!

```
>> length(v)
ans =
     2
```

The built-in function length() don’t give the actual length of the vector but finds number of elements in the vector or array, i.e., the size of the array.

[End of Example]

Orthogonality:

$$x^T y = 0$$

5.2 Matrices

Given a matrix A:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in R^{nxm}$$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

```
>> A=[0 1;-2 -3]
A =
     0     1

    -2    -3
```

[End of Example]

5.2.1 Transpose

The **Transpose** of matrix A:

$$A^T = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \in R^{mxn}$$

Example:

$$A^T = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}^T = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}$$

```
>> A'
ans =
     0    -2

     1    -3
```

[End of Example]

5.2.2 Diagonal

The **Diagonal** elements of matrix A is the vector

$$diag(A) = \begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{pp} \end{bmatrix} \in R^{p=\min(x,m)}$$

Example:

```
>> diag(A)
ans =
     0

    -3
```

[End of Example]

The **Diagonal** matrix *Λ* is given by:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \in R^{nxn}$$

Given the **Identity** matrix I:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \in R^{nxm}$$

Example:

```
>> eye(3)
ans =

     1     0     0

     0     1     0

     0     0     1
```

[End of Example]

5.2.3 Triangular

Lower Triangular matrix *L*:

$$L = \begin{bmatrix} . & 0 & 0 \\ \vdots & \ddots & 0 \\ . & \cdots & . \end{bmatrix}$$

Upper Triangular matrix *U*:

$$U = \begin{bmatrix} . & \cdots & . \\ 0 & \ddots & \vdots \\ 0 & 0 & . \end{bmatrix}$$

5.2.4 Matrix Multiplication

Given the matrices $A \in R^{nxm}$ and $B \in R^{mxp}$, then

$$C = AB \in R^{nxp}$$

where

$$c_{jk} = \sum_{l=1}^{n} a_{jl} b_{lk}$$

Example:

```
>> A = [0 1; -2 -3]
A =
     0     1

    -2    -3
>> B = [1 0;3 -2]
B =
     1     0

     3    -2
>> A*B
ans =
     3    -2

   -11     6
```

→ Check the answer by manually calculating using pen & paper.

[End of Example]

Note!

$$n\begin{bmatrix} \overset{m}{A} \end{bmatrix} m \begin{bmatrix} \overset{p}{B} \end{bmatrix} = n \begin{bmatrix} \overset{p}{C} \end{bmatrix}$$

Note!

$$AB \neq BA$$

$$A(BC) = (AB)C$$

$$(A + B)C = AC + BC$$

$$C(A + B) = CA + CB$$

5.2.5 Matrix Addition

Given the matrices $A \in R^{nxm}$ and $B \in R^{nxm}$, then

$$C = A + B \in R^{nxm}$$

Example:

```
>> A = [0 1; -2 -3]
>> B = [1 0; 3 -2]
>> A + B
ans =
     1     1

     1    -5
```

→ Check the answer by manually calculating using pen & paper.

[End of Example]

5.2.6 Determinant

Given a matrix $A \in R^{nxn}$, then the **Determinant** is given by:

$$det(A) = |A|$$

Given a $2x2$ matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

Then

$$det(A) = |A| = a_{11}a_{22} - a_{21}a_{12}$$

Example:

```
A =
     0     1

    -2    -3

>> det(A)
ans =
     2
```

→ Check the answer by manually calculating using pen & paper.

[End of Example]

Notice that

$$\det(AB) = \det(A)\det(B)$$

and

$$\det(A^T) = \det\,(A)$$

Example:

```
>> det(A*B)
ans =
    -4
>> det(A)*det(B)
ans =
    -4
>> det(A')
ans =
     2
>> det(A)
ans =
     2
```

[End of Example]

5.2.7 Inverse Matrices

The **inverse** of a quadratic matrix $A \in R^{nxn}$ is defined by:

$$A^{-1}$$

if

$$AA^{-1} = A^{-1}A = I$$

For a $2x2$ matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

The inverse A^{-1} is then given by

$$A^{-1} = \frac{1}{det\,(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \in R^{2x2}$$

Example:

```
A =
     0     1

    -2    -3

>> inv(A)

ans =

   -1.5000   -0.5000

    1.0000         0
```

→ Check the answer by manually calculating using pen & paper.

Notice that:

$$AA^{-1} = A^{-1}A = I$$

[End of Example]

5.3 Eigenvalues

Given $A \in R^{nxn}$, then the Eigenvalues is defined as:

$$det(\lambda I - A) = 0$$

Example:

```
A =
     0     1

    -2    -3

>> eig(A)
ans =
    -1

    -2
```

→ Check the answer by manually calculating using pen & paper.

[End of Example]

Task 4: Matrix manipulation

In this task we will practice on entering matrices and perform basic matrix operations.

Given the matrices A, B and C:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & -1 \\ -2 & 2 \end{bmatrix}$$

→ Solve the following basic matrix operations using MATLAB:

- $A + B$
- $A - B$
- A^T
- A^{-1}
- $diag(A), diag(B)$
- $det(A), det(B)$
- $det(AB)$
- $eig(A)$

where eig = Eigenvalues, diag = Diagonal, det = Determinant

→ Use MATLAB to "prove" the following:

- $AB \neq BA$
- $A(BC) = (AB)C$
- $(A + B)C = AC + BC$
- $C(A + B) = CA + CB$
- $\det(AB) = \det(A)\det(B)$
- $\det(A^T) = \det(A)$
- $AA^{-1} = A^{-1}A = I$

where I is the unit matrix

By "proving", I mean enter the left side in MATLAB, then enter the right side, then check if you get the same results or not. In that way you have "proved" it.

[End of Task]

5.4 Solving Linear Equations

MATLAB can easily be used to solve a large amount of linear equations using built-in functions.

When dealing with large matrices (finding inverse of A is time-consuming) or the inverse doesn't exist other methods are used to find the solution, such as:

- Least Square method
- LU factorization
- Singular value Decomposition
- Etc.

In MATLAB we can also simply use the backslash operator "\" in order to find the solution like this:

```
x = A\b
```

Example:

Given the following equations:

$$x_1 + 2x_2 = 5$$

$$3x_1 + 4x_2 = 6$$

$$7x_1 + 8x_2 = 9$$

We can set in on matrix form:

$$Ax = b$$

But as you know, it only works when A is a square matrix.

From the equations we find:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 7 & 8 \end{bmatrix}$$

$$b = \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix}$$

Normally we can find the solution by taking the inverse of the A matrix:

$$x = A^{-1}b$$

As you can see, the A matrix is not a quadratic matrix, meaning we cannot find the inverse of A, thus $x = A^{-1}b$ will not work (try it in MATLAB and see what happens).

Let’s try (just to verify that it is not working):

```
>> A = [1 2; 3 4; 7 8];
>> b = [5;6;9];
>> x = inv(A)*b
Error using inv
Matrix must be square.
```

As expected, MATLAB cannot solve this because the A matrix is not square.

So, we can solve it using the backslash operator “\”:

```
A = [1 2; 3 4; 7 8];
b = [5;6;9];
x = A\b
```

This gives the answer:

```
x =

   -3.5000

    4.1786
```

Meaning $x_1 = -3.5$ and $x_2 = 4.1786$

Actually, when using the backslash operator “\” in MATLAB it uses the LU factorization as part of the algorithm to find the solution.

We could have used the known Least Square Method formula as well.

Given:

$$Ax = b$$

Then the Least Square Method formula is given by (how we can derive this equation will not be shown here):

$$x_{LS} = (A^T A)^{-1} A^T b$$

Let’s try:

```
A = [1 2; 3 4; 7 8];
b = [5;6;9];
x_ls = inv(A'*A)*A'*b
```

This gives the same answer:

```
x =

   -3.5000

    4.1786
```

This means using the Least Square formula also works fine in this case. It gives same results as using the backslash. "\" operator in this case. In general, the backslash operator is better to use because it finds the best way to solve the equations. MATLAB does the "dirty work" for you.

Task 5: Solving Linear Equations

Given the equations:

$$x_1 + 2x_2 = 5$$
$$3x_1 + 4x_2 = 6$$

Set the equations on the following form:

$$Ax = b$$

→ Find A and b and define them in MATLAB.

Solve the equations, i.e., find $x_{1,}\, x_2$, using MATLAB. It can be solved like this:

$$Ax = b \rightarrow x = A^{-1}b$$

[End of Task]

6 M-files; Scripts and user-define functions

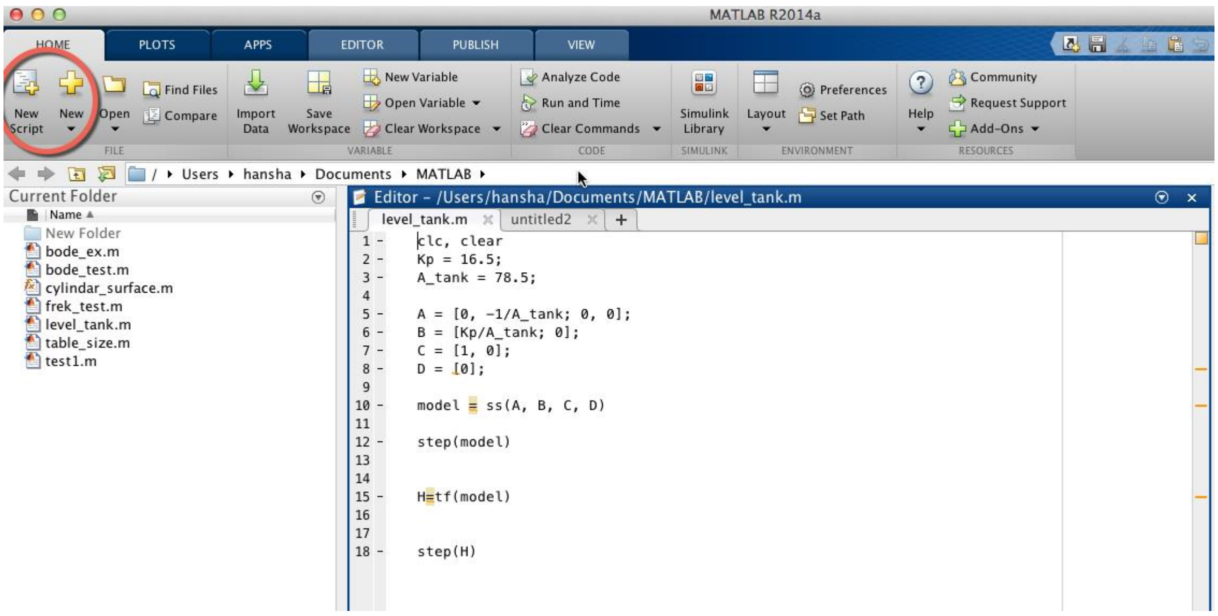
Scripts or m-files are text files containing MATLAB code. Use the MATLAB Editor or another text editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends with ".m".

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.

Before you start, you should watch the video "**Writing a MATLAB Program**".

The video is available from:
https://www.halvorsen.blog/documents/teaching/courses/matlab/matlab1.php

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



```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

6.1 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.

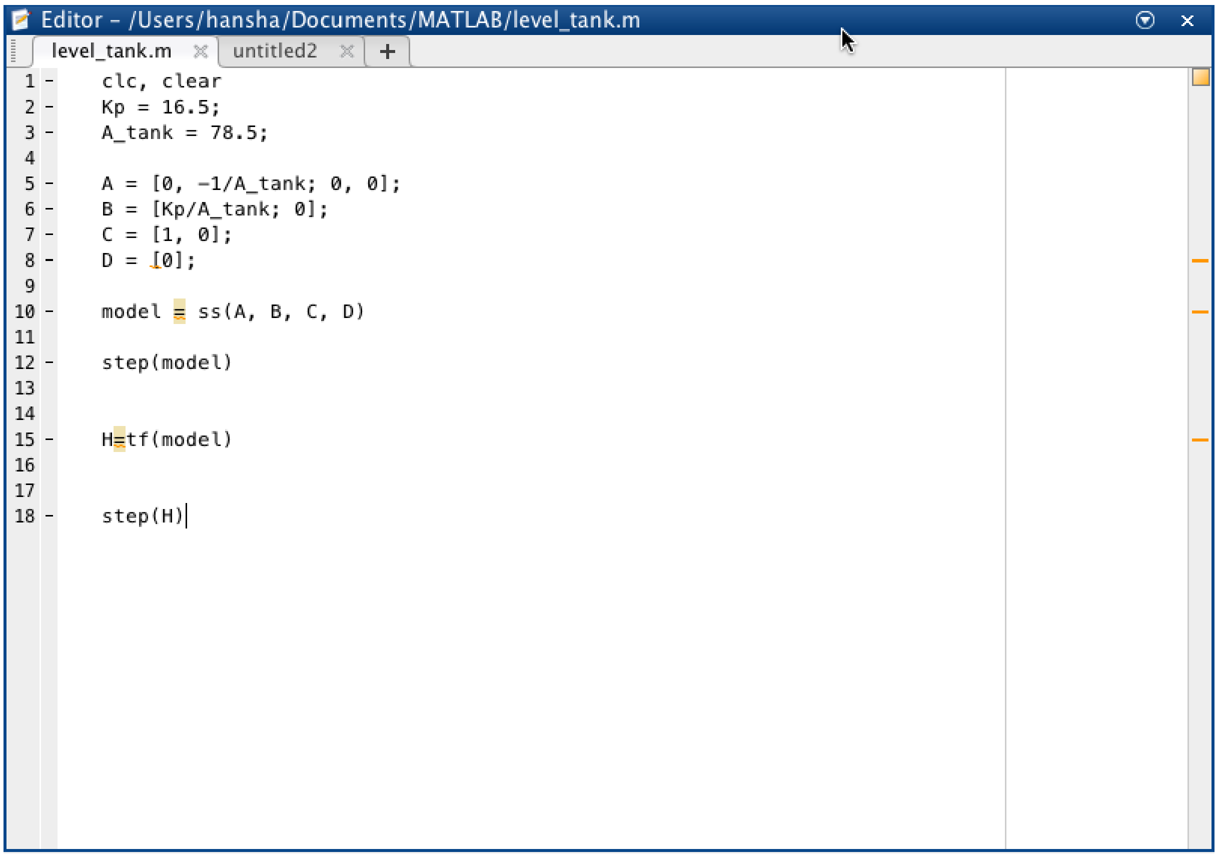
MATLAB have lots of built-in functions, but very often we need to create our own functions (these are called user-defined functions)

Below we will learn more about Scripts and Functions.

6.2 Scripts

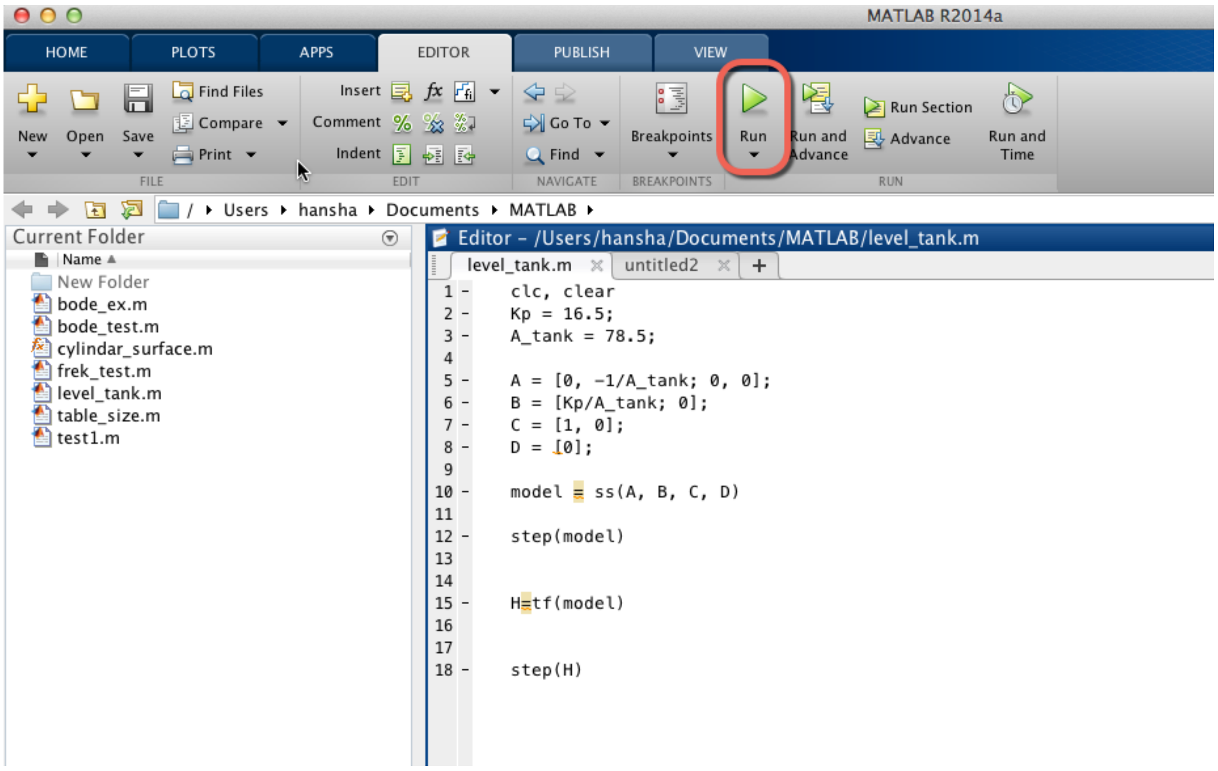
A Script is a collection of MATLAB commands and functions that is bundled together in a m-file. When you run the Script, all the commands are executed sequentially.

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



```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

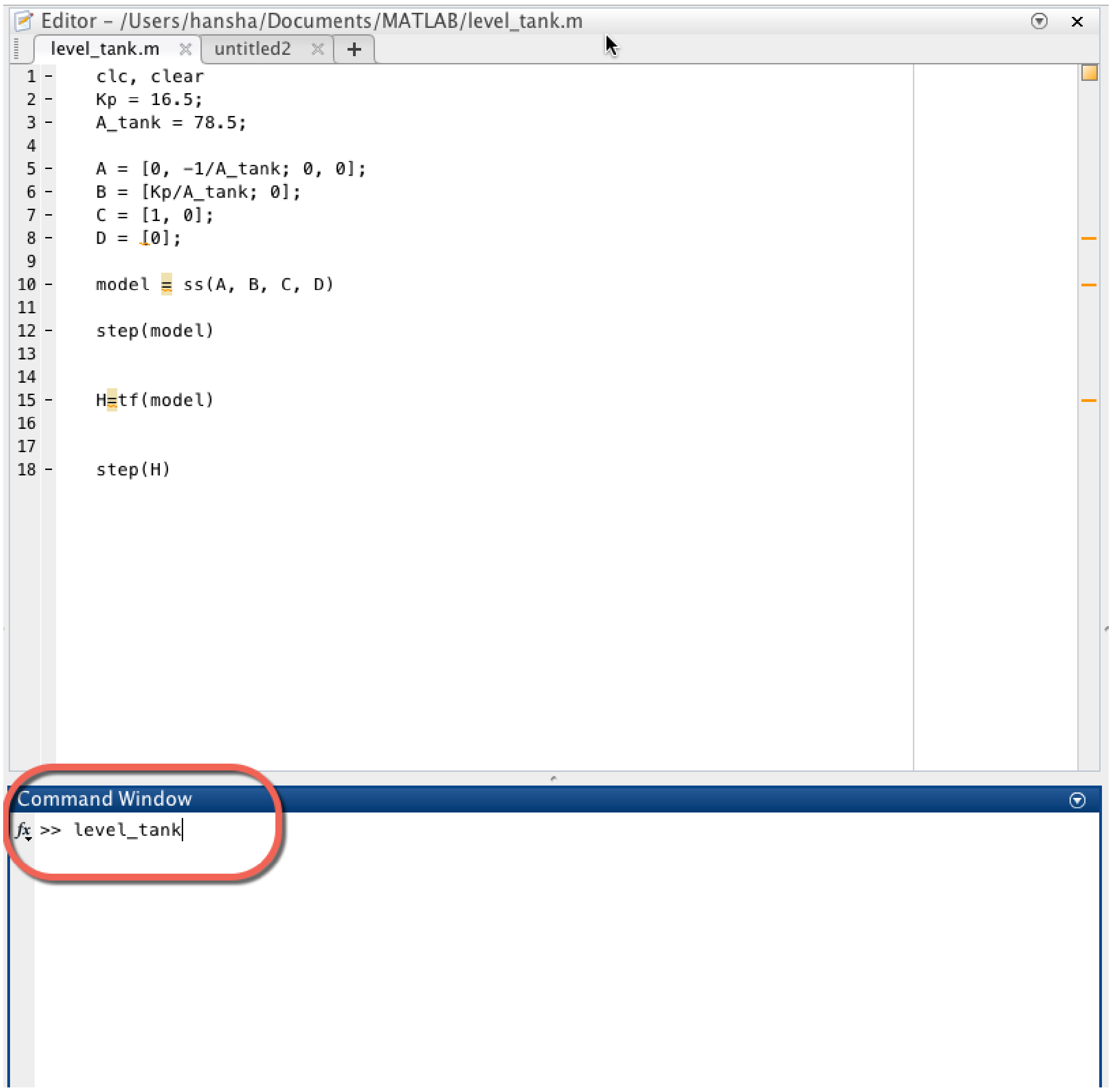
In the Editor you create a sequence of MATLAB commands that you save as a m-file (the file extension ends with .m). Push the “Run” button when you want to run your program.



```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

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):



```
clc, clear
Kp = 16.5;
A_tank = 78.5;

A = [0, -1/A_tank; 0, 0];
B = [Kp/A_tank; 0];
C = [1, 0];
D = [0];

model = ss(A, B, C, D)

step(model)


H=tf(model)


step(H)
```

```
>> level_tank
```

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]

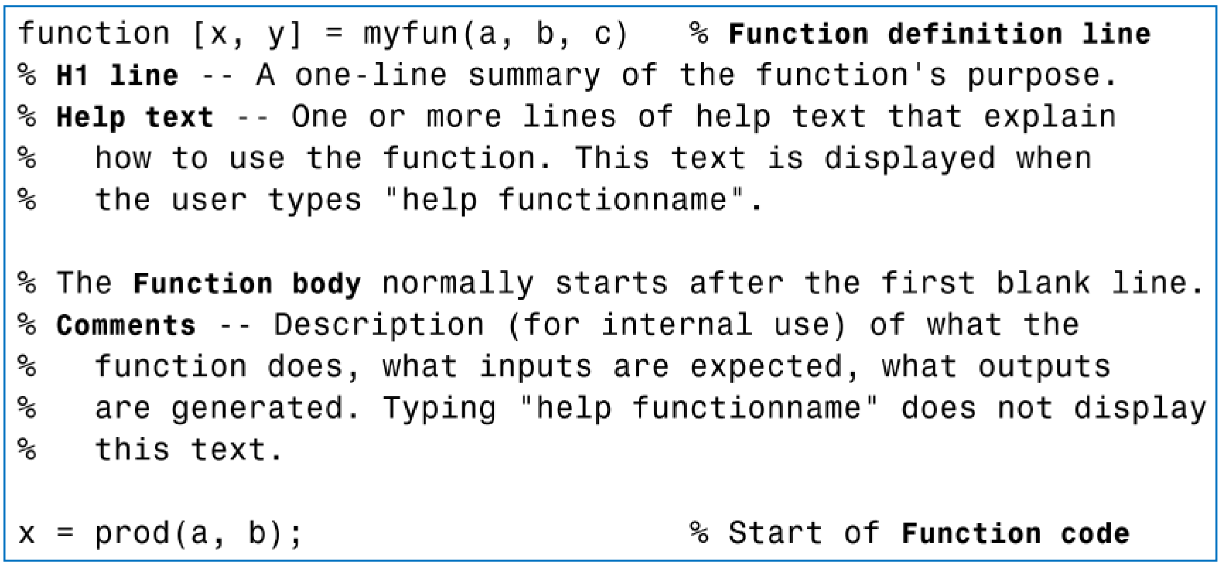
6.3 Functions

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:

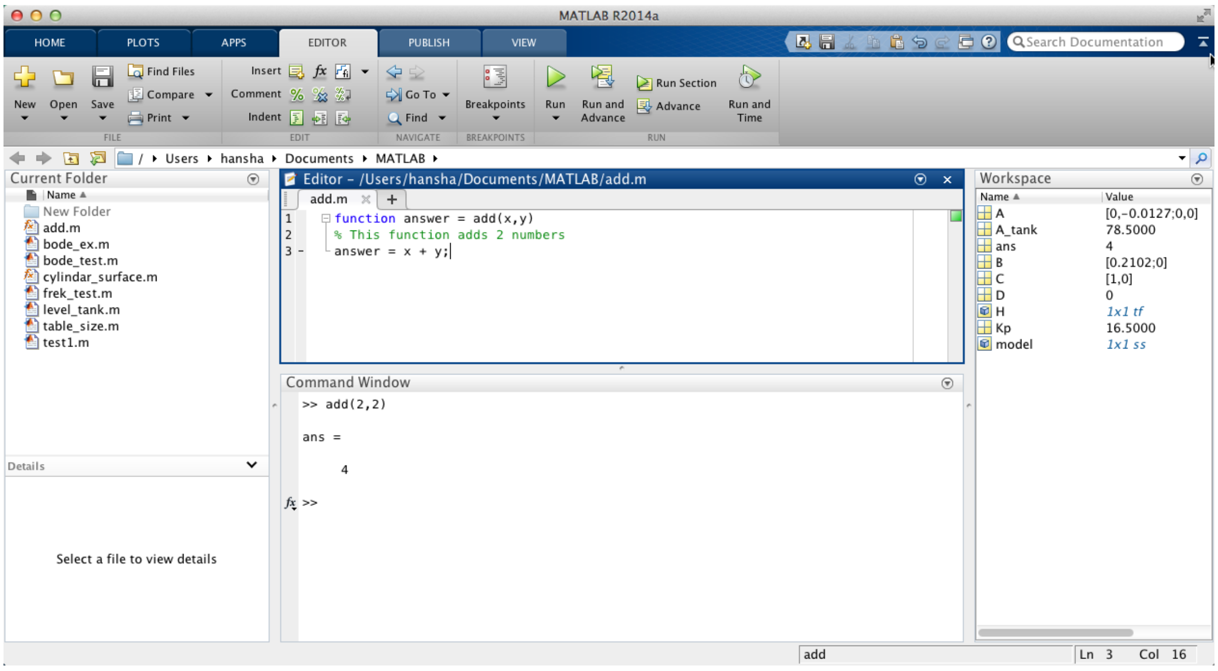
```
function [out1,out2] = functionName(in1,in2,…)
```

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.



Current Folder:

New Folder
add.m
bode_ex.m
bode_test.m
cylindar_surface.m
frek_test.m
level_tank.m
table_size.m
test1.m

```
function answer = add(x,y)
% This function adds 2 numbers
answer = x + y;
```

```
>> add(2,2)

ans =

     4

>>
```

Name	Value
A	[0,-0.0127;0,0]
A_tank	78.5000
ans	4
B	[0.2102;0]
C	[1,0]
D	0
H	1x1 tf
Kp	16.5000
model	1x1 ss

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]

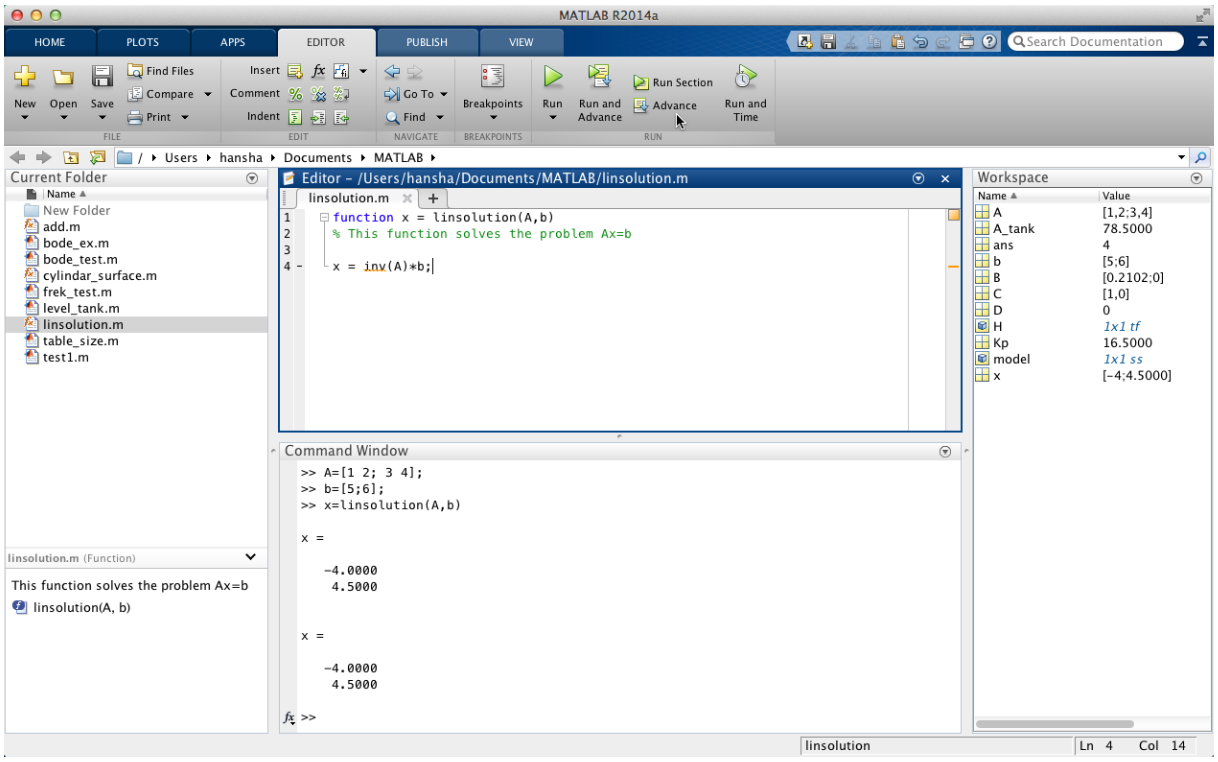
You may create your own functions and save them as a m-file. Functions are M-files that can accept input arguments and return output arguments. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

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:



```
function x = linsolution(A,b)
% This function solves the problem Ax=b

x = inv(A)*b;
```

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

x =

   -4.0000
    4.5000


x =

   -4.0000
    4.5000

>>
```

Name	Value
A	[1,2;3,4]
A_tank	78.5000
ans	4
b	[5;6]
B	[0.2102;0]
C	[1,0]
D	0
H	*1x1 tf*
Kp	16.5000
model	*1x1 ss*
x	[-4;4.5000]

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]

<u>Naming a Function Uniquely:</u>

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]

7 Plotting

Plotting is a very important and powerful feature in MATLAB. In this chapter we will learn the basic plotting functionality in MATLAB.

Plots functions: Here are some useful functions for creating plots:

Function	Description	Example

plot	Generates a plot. plot(y) plots the columns of y against the indexes of the columns.	`>X = [0:0.01:1];`
		`>Y = X.*X;`
		`>plot(X, Y)`

figure	Create a new figure window	`>>figure`
		`>>figure(1)`

subplot	Create subplots in a Figure. subplot(m,n,p) or subplot(mnp), breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot. The axes are counted along the top row of the Figure window, then the second row, etc.	`>>subplot(2,2,1)`

grid	Creates grid lines in a plot.
	"grid on" adds major grid lines to the current plot.
	"grid off" removes major and minor grid lines from the current plot.
		`>>grid`
		`>>grid on`
		`>>grid off`

axis	Control axis scaling and appearance. "axis([xmin xmax ymin ymax])" sets the limits for the x- and y-axis of the current axes.	`>>axis([xmin xmax ymin ymax])`
		`>>axis off`
		`>>axis on`

title	Add title to current plot title('string')	`>>title('this is a title')`
xlabel	Add xlabel to current plot xlabel('string')	`>> xlabel('time')`
ylabel	Add ylabel to current plot ylabel('string')	`>> ylabel('temperature')`
legend	Creates a legend in the corner (or at a specified position) of the plot	`>> legend('temperature')`

hold	Freezes the current plot, so that additional plots can be overlaid	`>>hold on`
		`>>hold off`

Type "**help graphics**" in the Command Window for more information, or type "**help <functionname>**" for help about a specific function.

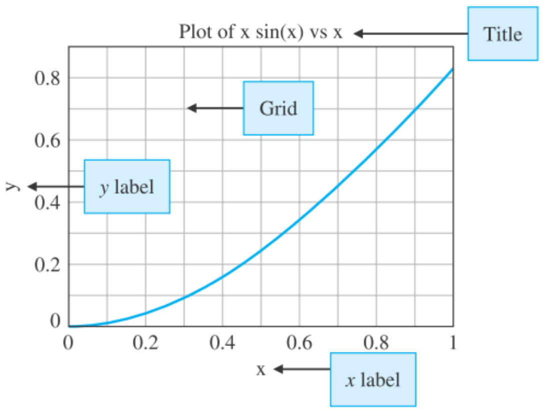
Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

Example:

Here we see some examples of how to use the different plot functions:

```
>>x=[0:0.1:1]';
>>y=x.*sin(x);
>>plot(x,y)
>>title('Plot of x sin(x) vs x ')
>>xlabel('x')
>>ylabel('y')
>>grid on
```

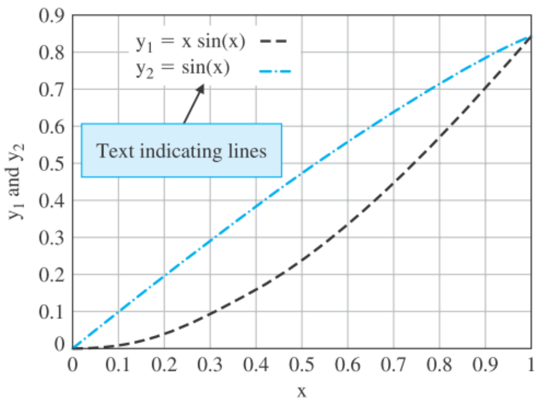
(a)



```
>>  x=[0:0.1:1]';
>>  y1=x.*sin(x); y2=sin(x);
>> plot(x,y1,'--',x,y2,'-.')
>> text(0.1,0.85,'y_1 = x sin(x) ---')
>> text(0.1,0.80,'y_2 = sin(x) .\_.\_')
>> xlabel('x'), ylabel('y_1 and y_2'), grid on
```

Dashed line for y1
Dashed-dot line for y2

(a)



[End of Example]

Before you start using these functions, you should watch the video "**Using Basic Plotting Functions**".

The video is available from:
https://www.halvorsen.blog/documents/teaching/courses/matlab/matlab1.php

Task 9: Plotting

In the Command window (or use the Script Editor) in MATLAB window input the time from $t = 0$ seconds to $t = 10$ seconds in increments of 0.1 seconds as follows:

```
>>t = [0:0.1:10];
```

Then, compute the output y as follows:

```
>>y = cos(t);
```

Use the Plot command:

```
>>plot(t,y)
```

[End of Task]

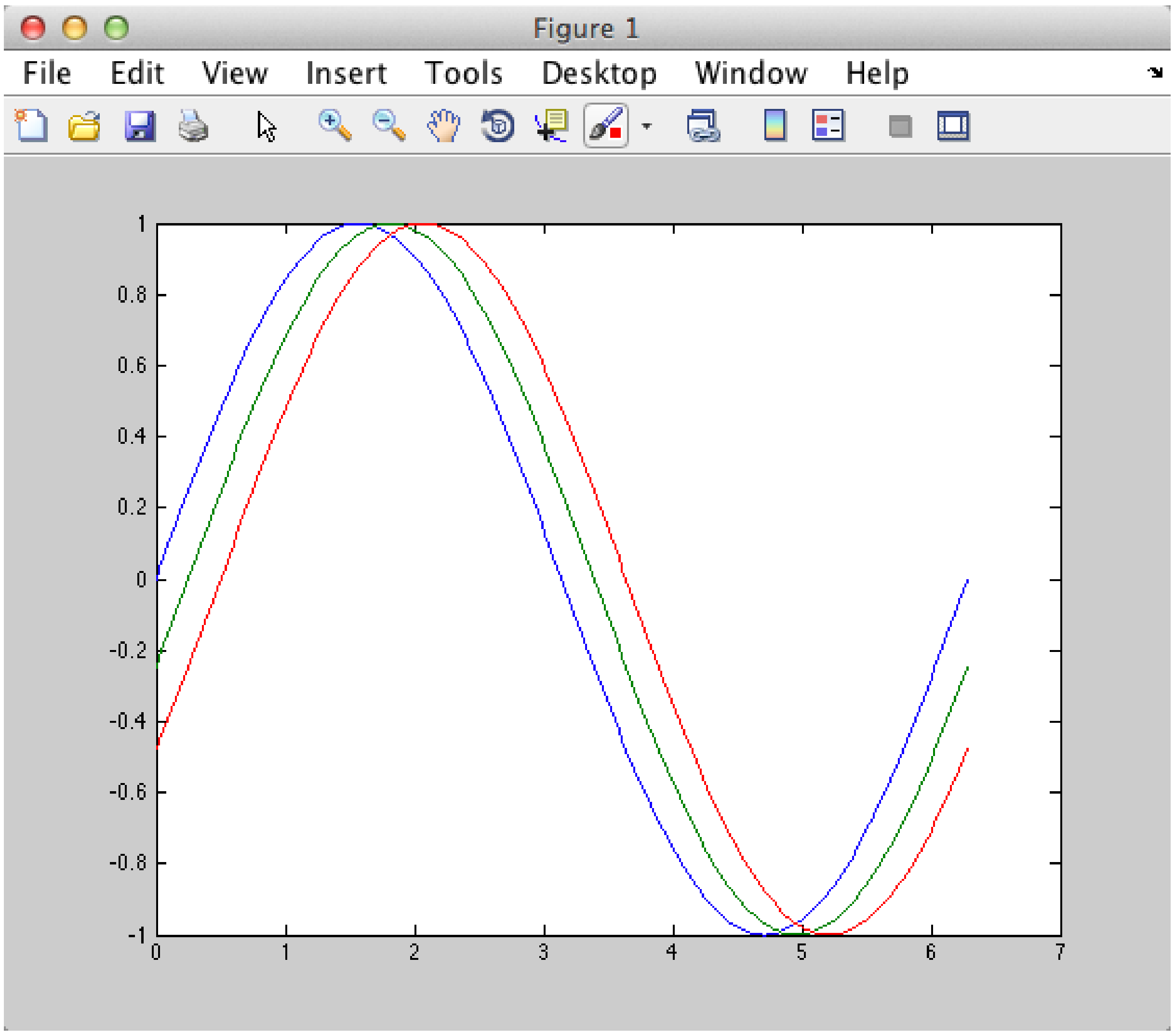
7.1 Plotting Multiple Data Sets in One Graph

In MATLAB it is easy to plot multiple data set in one graph.

Example:

```
x = 0:pi/100:2*pi;
y = sin(x);
y2 = sin(x-.25);
y3 = sin(x-.5);
plot(x,y, x,y2, x,y3)
```

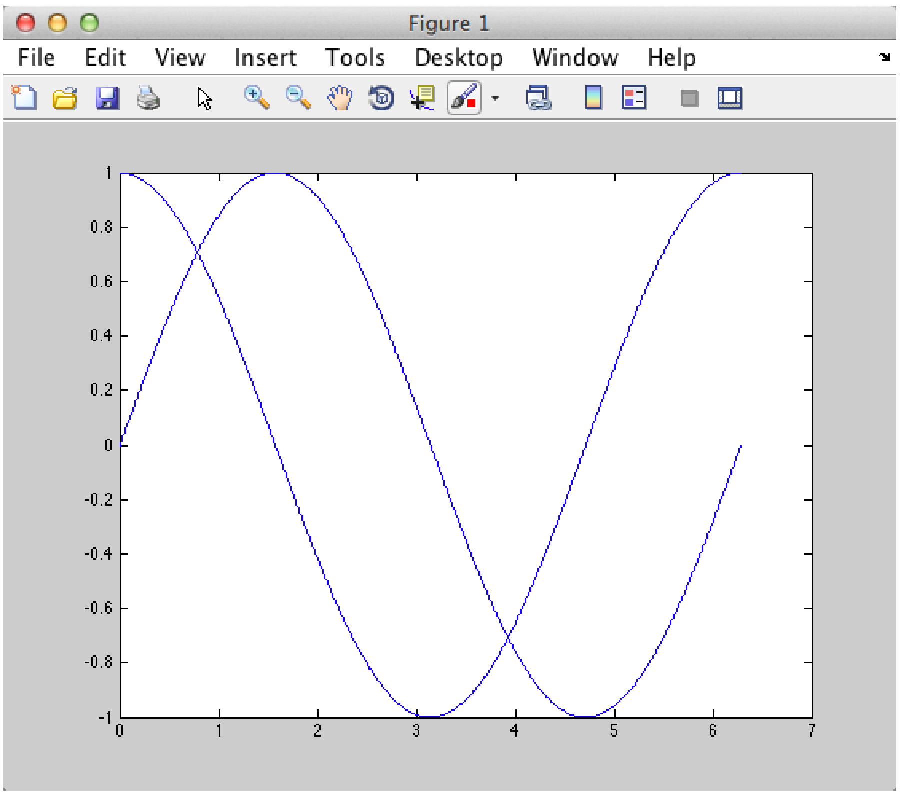
This gives the following plot:



Another approach is to use the **hold** command:

```
x=0:0.01:2*pi;

plot(x, sin(x))
hold on

plot(x, cos(x))
hold off
```

This gives the following plot:



[End of Example]

You can also do the plotting in different plots using the **figure()** command.

Example:

```
x=0:0.01:2*pi;
figure(1)
plot(x, sin(x))
figure(2)
plot(x, cos(x))
```

The results will be like this:

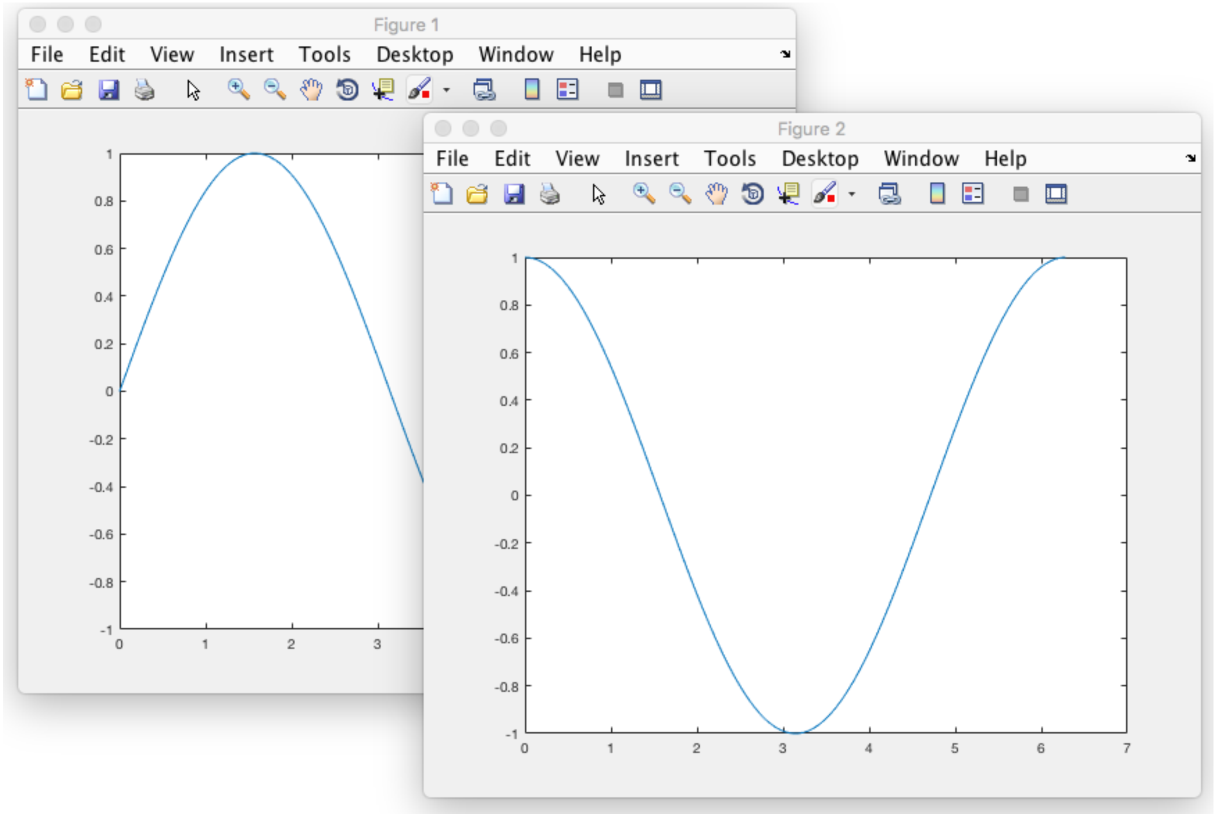


Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 2
File Edit View Insert Tools Desktop Window Help

[End of Example]

Task 10: Plot of dynamic system

Given the following differential equation:

$$\dot{x} = ax$$

where $a = -\frac{1}{T}$,where T is the time constant

The solution for the differential equation is:

$$x(t) = e^{at}x_0$$

Set $T = 5$ and the initial condition $x(0) = 1$

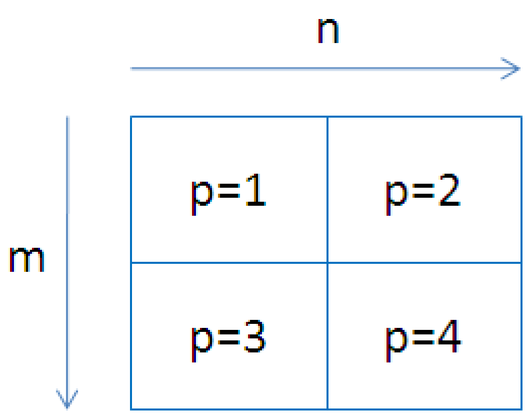
→ Create a Script in MATLAB (.m file) where you plot the solution $x(t)$ in the time interval $0 \le t \le 25$

→ Add Grid, and proper Title and Axis Labels to the plot.

[End of Task]

7.2 Displaying Multiple Plots in one Figure – Sub-Plots

The subplot command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing "subplot(m,n,p)" partitions the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot. The plots are numbered along the first row of the figure window, then the second row, and so on.



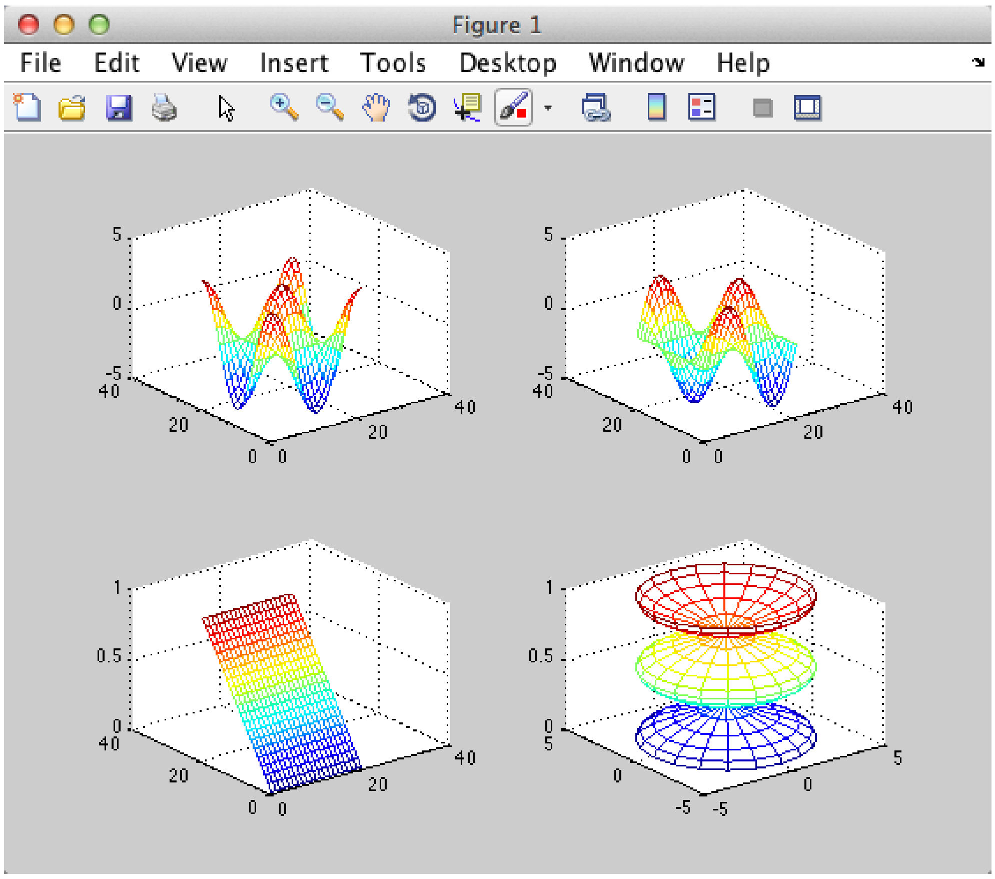
The syntax is as follows:

```
subplot(m,n,p)
```

Example:

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X)
subplot(2,2,2); mesh(Y)
subplot(2,2,3); mesh(Z)
subplot(2,2,4); mesh(X,Y,Z)
```

This gives:



[End of Example]

Task 11: Sub-plots

Plot Sin(x) and Cos(x) in 2 different subplots.

Add Titles and Labels.

[End of Task]

7.3 Custimizing

There is lots of customizing you can do with plots, e.g., you can add a title, x- and y-axis labels, add a legend and customize line colors and line-styles.

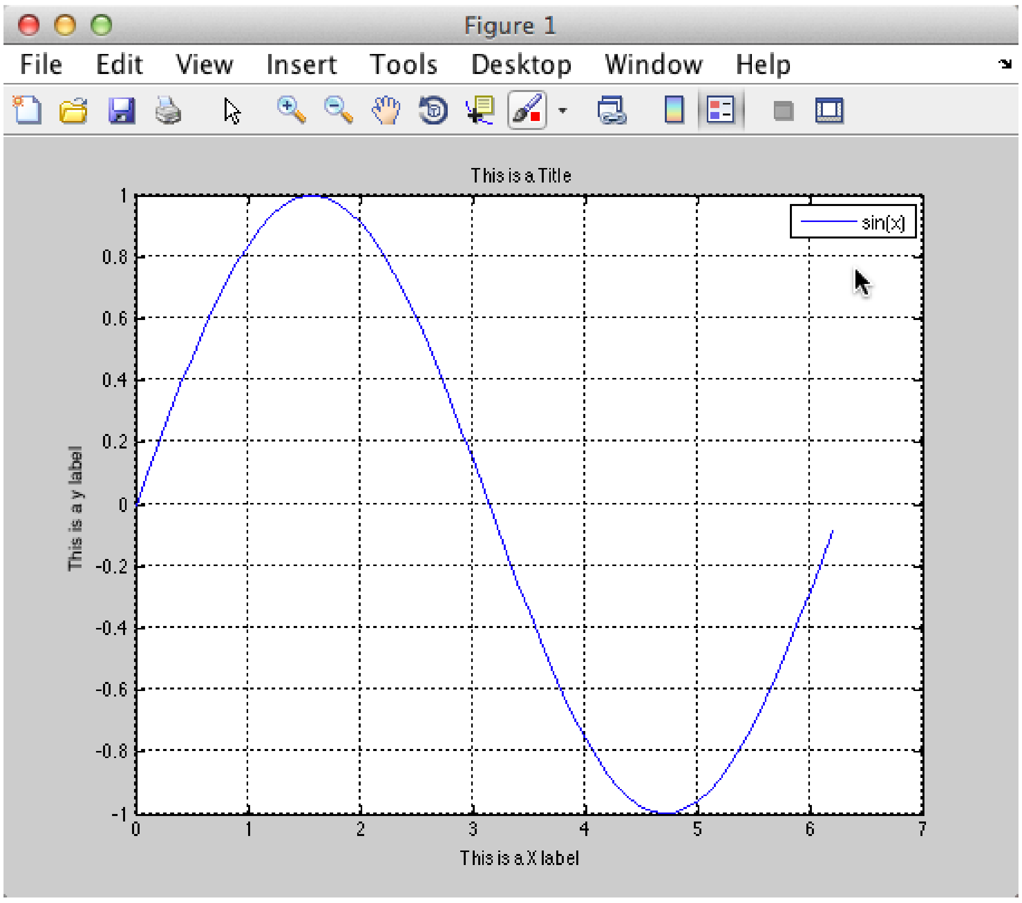
The functions for doing this is; **title**, **xlabel**, **ylabel**, **legend**, etc.

Example:

```
x=0:0.1:2*pi;
```

```
plot(x, sin(x))
%Customize the Plot:
title('This is a Title')
xlabel('This is a X label')
ylabel('This is a y label')
legend('sin(x)')
grid on
```

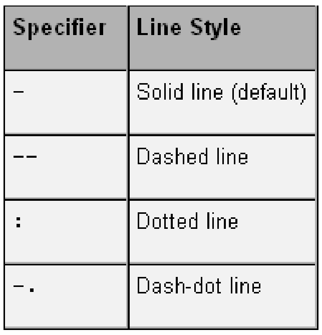
This gives the following plot:



[End of Example]

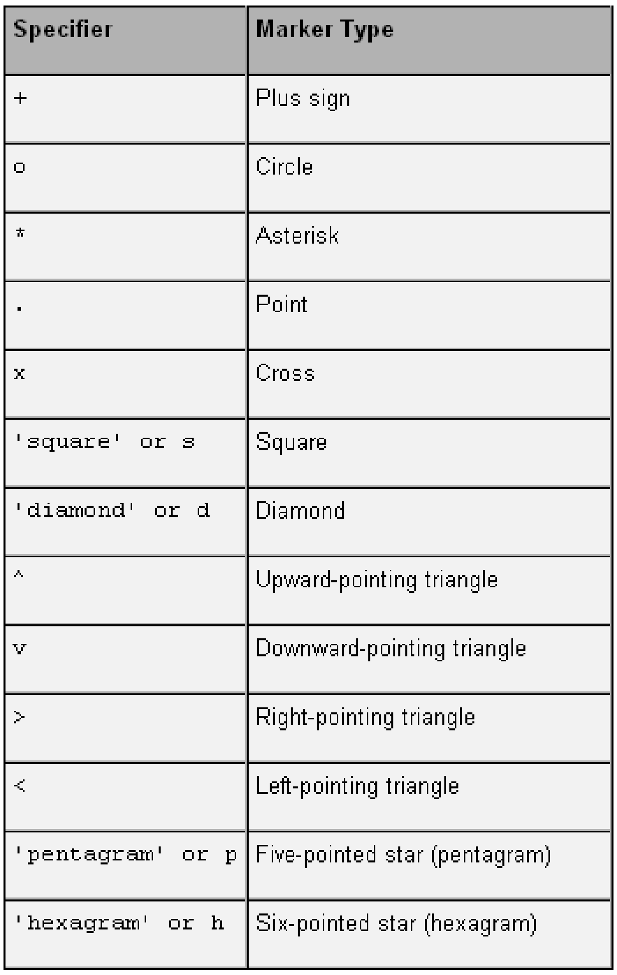
For line colors and line-styles we have the following properties we can use for the **plot** function:

Line Styles:



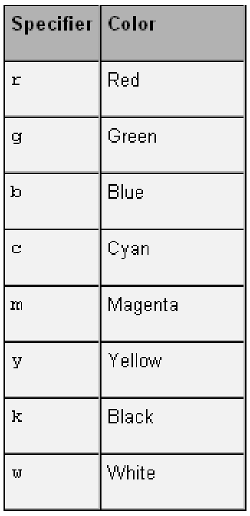
Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Marker specifiers:



Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or s	Square
'diamond' or d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram' or p	Five-pointed star (pentagram)
'hexagram' or h	Six-pointed star (hexagram)

Colors:

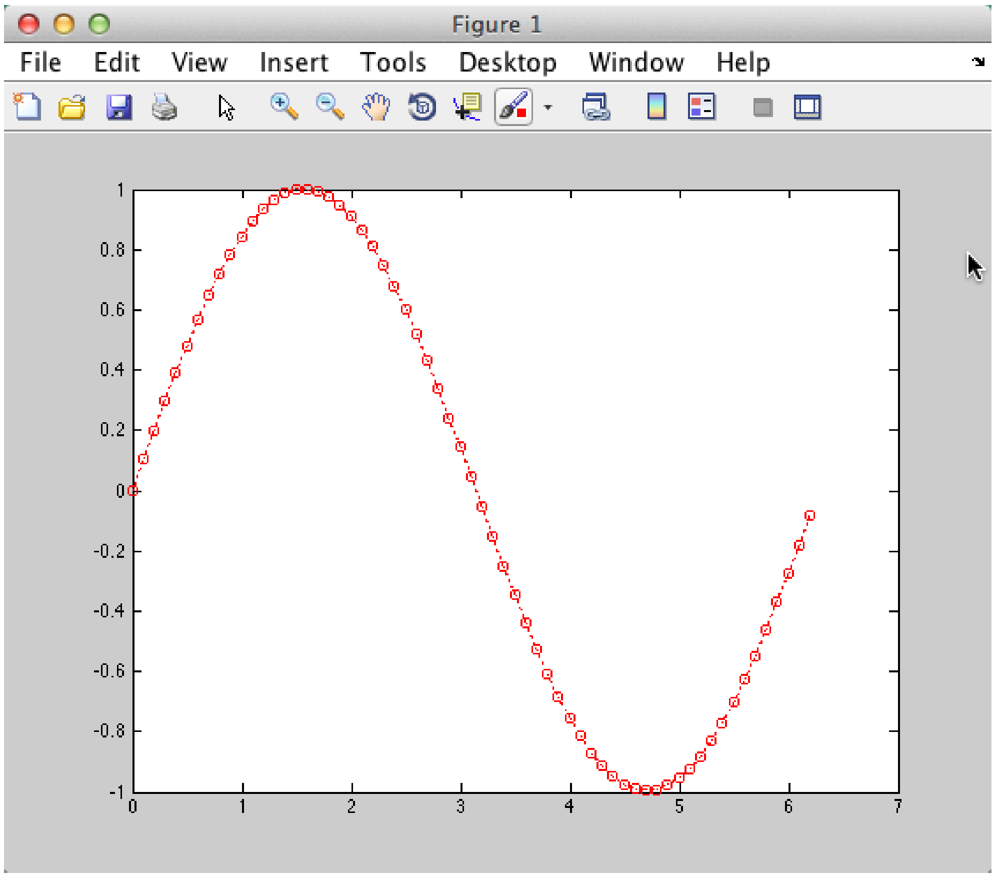


Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Example:

```
>> x=0:0.1:2*pi;
>> plot(x, sin(x), 'r:o')
```

This gives the following plot:



[End of Example]

7.4 Other Plots

MATLAB offers lots of different plots.

Task 12: Other Plots

Check out the help for the following 2D functions in MATLAB: loglog, semilogx, semilogy, plotyy, polar, fplot, fill, area, bar, barh, hist, pie, errorbar, scatter.

→ Try some of them, e.g., bar, hist and pie.

[End of Task]

8 Flow Control and Loops

8.1 Introduction

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.

8.2 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 code 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	<
≤	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 ∅, you can just type disp('there is no solution') and for ℂ you can type disp('any complex number is a solution') – or something like that.

[End of Task]

8.3 Switch and Case Statement

The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups. Only the first matching case is executed. There must always be an end to match the switch.

The general syntax is as follows:

```
switch variable
 case case_value1
   statements1
  case case_value2
   statements2
   …
  otherwise
   statements
end
```

Example:

```
n=2
switch(n)
   case 1
      M = eye(n)
   case 2
      M = zeros(n)
   case 3
      M = ones(n)
end
```

[End of Example]

Task 14: Switch-Case Statements

Create a function that finds either the Area or the circumference of a circle using a Switch-Case statement

You can, e.g., call the function like this:

```
>> r=2;
>> calccircl(r,1) % 1 means area
>> calccircl(r,2) % 2 means circumference
```

[End of Task]

8.4 For loop

The For loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements.

The general syntax is as follows:

```
for variable = initval:endval
    statement
    ...
    statement
end
```

Example:

```
m=5
for n = 1:m
    r(n) = rank(magic(n));
end
r
```

[End of Example]

Task 15: Fibonacci Numbers

In mathematics, Fibonacci numbers are the numbers in the following sequence:

0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, …

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two. Some sources omit the initial 0, instead beginning the sequence with two 1s.

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation:

$$f_n = f_{n-1} + f_{n-2}$$

with seed values:

$$f_0 = 0, f_1 = 1$$

→ Write a function in MATLAB that calculates the N first Fibonacci numbers, e.g.,

```
>> N=10;
>> fibonacci(N)
ans =
     0
     1
     1
     2
     3
     5
     8
    13
    21
    34
```

→ Use a For loop to solve the problem.

Fibonacci numbers are used in the analysis of financial markets, in strategies such as Fibonacci retracement, and are used in computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure. They also appear in biological settings, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

[End of Task]

8.5 While loop

The while loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching end delineates the statements.

The general syntax is as follows:

```
while expression
```

```
    statements
end
```

Example:

```
m=5;
while m > 1
    m = m - 1;
    zeros(m)
end
```

[End of Example]

Task 16: While Loop

Create a Script or Function that creates Fibonacci Numbers up to a given number, e.g.,

```
>> maxnumber=2000;
>> fibonacci(maxnumber)
```

Use a While Loop to solve the problem.

[End of Task]

8.6 Additional Tasks

Here are some additional tasks about Loops and Flow control.

Task 17: For Loops

Extend your **calc_average** function from a previous task so it can calculate the average of a vector with random elements. Use a For loop to iterate through the values in the vector and find sum in each iteration:

```
mysum = mysum + x(i);
```

Test the function in the Command window

[End of Task]

Task 18: If-else Statement

Create a function where you use the "**if-else**" statement to find elements larger than a specific value in the task above. If this is the case, discard these values from the calculated average.

Example discarding numbers larger than 10 gives:

```
x =
     4     6    12
>> calc_average(x)
ans =
     5
```

[End of Task]

9 Mathematics

MATLAB is a powerful tool for mathematical calculations.

Type “help elfun” (elementary functions) in the Command window for more information about basic mathematical functions.

9.1 Basic Math Functions

Some Basic Math functions in MATLAB: **exp**, **sqrt**, **log**, etc.→ Look up these functions in the Help system in MATLAB.

Task 19: Basic Math function

Create a function that calculates the following mathematical expression:

$$z = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)}$$

Test with different values for x and y.

[End of Task]

9.2 Statistics

Some Statistics functions in MATLAB: **mean**, **max**, **min**, **std**, etc.

→ Look up these functions in the Help system in MATLAB.

Task 20: Statistics

Create a vector with random numbers between 0 and 100. Find the following statistics: mean, median, standard deviation, minimum, maximum and the variance.

[End of Task]

9.3 Trigonometric Functions

MATLAB offers lots of Trigonometric functions, e.g., **sin**, **cos**, **tan**, etc. → Look up these functions in the Help system in MATLAB.

Note! Most of the trigonometric functions require that the angle is expressed in radians.

Example:

```
>> sin(pi/4)
ans =
    0.7071
```

[End of Example]

Task 21: Conversion

Since most of the trigonometric functions require that the angle is expressed in radians, we will create our own functions in order to convert between radians and degrees.

It is quite easy to convert from radians to degrees or from degrees to radians. We have that:

$$2\pi\ [radians] = 360\ [degrees]$$

This gives:

$$d\ [degrees] = r[radians] \cdot \left(\frac{180}{\pi}\right)$$

$$r[radians] = d[degrees] \cdot \left(\frac{\pi}{180}\right)$$

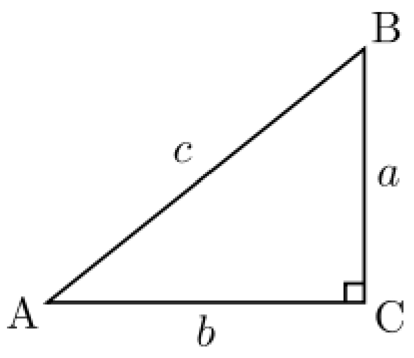
→ Create two functions that convert from radians to degrees (***r2d(x)***) and from degrees to radians (***d2r(x)***) respectively.

Test the functions to make sure that they work as expected.

[End of Task]

Task 22: Trigonometric functions on right triangle

Given right triangle:



→ Create a function that finds the angle A (in degrees) based on input arguments (a, c), (b, c) and (a, b) respectively.

Use, e.g., a third input "type" to define the different types above.

→ Use you previous function **r2d()** to make sure the output of your function is in degrees and not in radians.

Test the functions to make sure it works properly.

Tip! We have that:

$$\sin A = \frac{a}{c}, A = arcsin\left(\frac{a}{c}\right)$$

$$\cos A = \frac{b}{c}, A = arccos\left(\frac{b}{c}\right)$$

$$\tan A = \frac{a}{b}, A = arctan\left(\frac{a}{b}\right)$$

We may also need the Pythagoras' theorem:

$$c^2 = a^2 + b^2$$

Testing the function can be done like this in the Command Window:

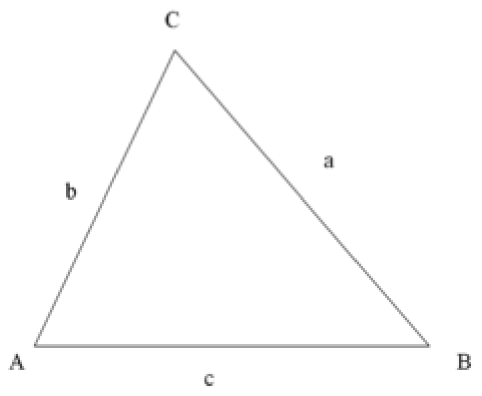
```
>> a=5, b=8, c=sqrt(a^2+b^2);
>> A = right_triangle(a,c,'sin')
A =
   32.0054
>> A = right_triangle(b,c,'cos')
A =
   32.0054
>> A = right_triangle(a,b,'tan')
A =
  32.0054
```

We also see that the answer in this case is the same, which is expected.

[End of Task]

Task 23: Law of cosines

Given:



Create a function where you find c using the **law of cosines**.

$$c^2 = a^2 + b^2 - 2ab\ cosC$$

Test the functions to make sure it works properly.

[End of Task]

Task 24: Plotting

Plot $sin(\theta)$ and $cos(\theta)$ for $0 \leq \theta \leq 2\pi$ in the same plot.

Make sure to add labels and a legend and use different line styles and colors for the plots.

[End of Task]

9.4 Complex Numbers

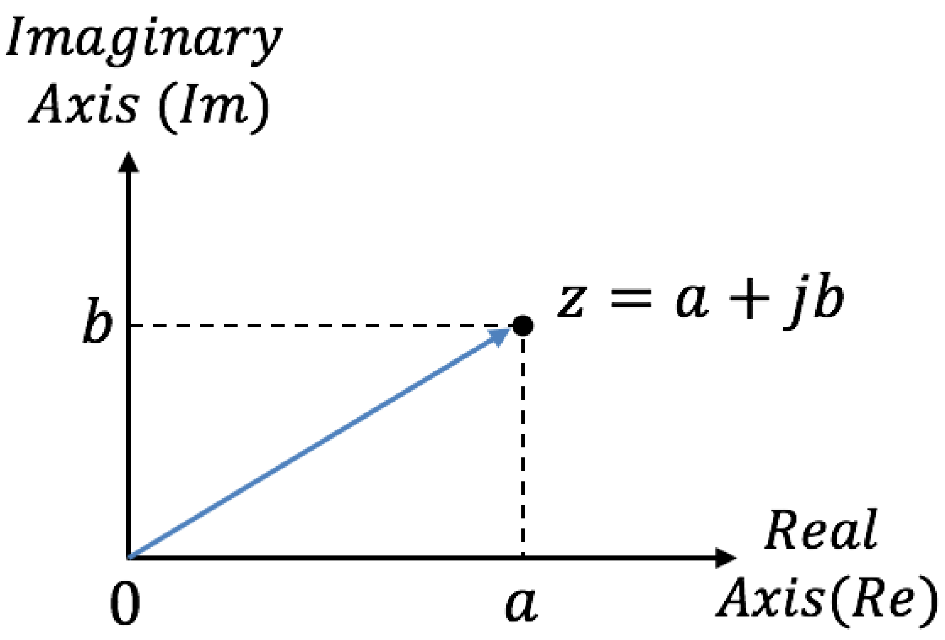
Complex numbers are important in modelling and control theory.

A complex number is defined like this:

$$z = a + ib$$

or

$$z = a + jb$$



The imaginary unit i or j is defined as:

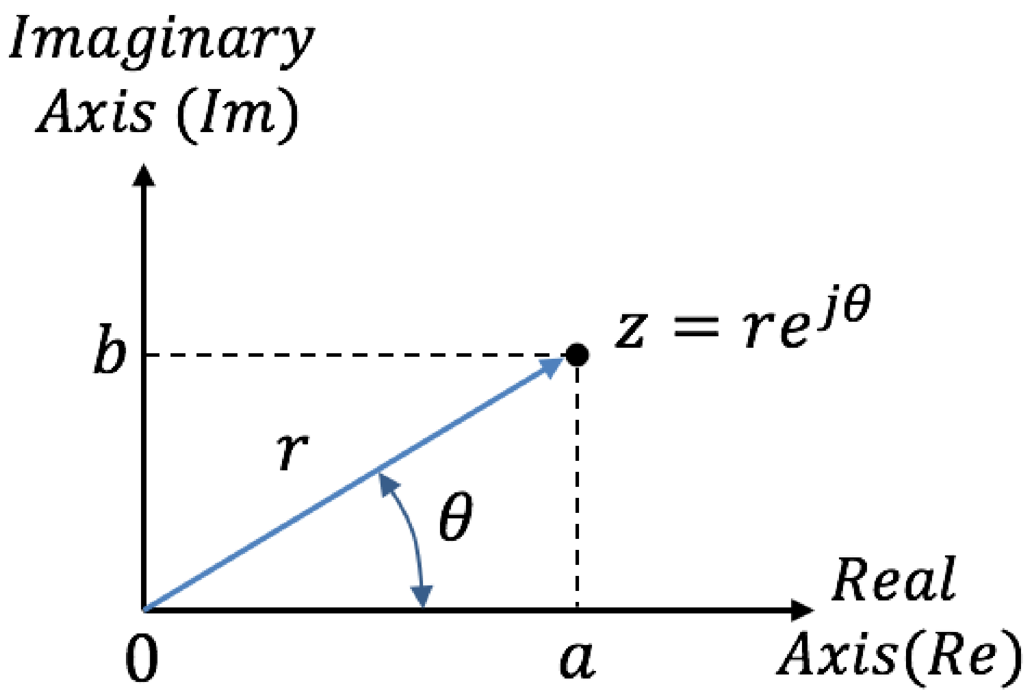
$$i = \sqrt{-1}$$

Where a is called the real part of z and b is called the imaginary part of z, i.e.:

$Re(z) = a,\ Im(z) = b$

You may also imaginary numbers on exponential/polar form:

$$z = re^{j\theta}$$



where:

$$r = |z| = \sqrt{a^2 + b^2}$$

$$\theta = atan\frac{b}{a}$$

Note that $a = r\cos\theta$ and $b = r\sin\theta$

Example:

Given the following complex number:

$$z = 2 + i3$$

In MATLAB we may type:

```
>> z=2+3i
```

or:

```
>> z=2+3j
```

[End of Example]

The complex conjugate of z is defined as:

$$z^* = a - ib$$

To add or subtract two complex numbers, we simply add (or subtract) their real parts and their imaginary parts.

In Division and multiplication, we use the polar form.

Given the complex numbers:

$z_1 = r_1 e^{j\theta_1}$ and $z_2 = r_2 e^{j\theta_2}$

Multiplication:

$$z_3 = z_1 z_2 = r_1 r_2 e^{j(\theta_1 + \theta_2)}$$

Division:

$$z_3 = \frac{z_1}{z_2} = \frac{r_1 e^{j\theta_1}}{r_2 e^{j\theta_2}} = \frac{r_1}{r_2} e^{j(\theta_1 - \theta_2)}$$

MATLAB functions:

Some Basic functions for complex numbers in MATLAB: **abs**, **angle**, **imag**, **real**, **conj**, **complex**, etc.

Function	Description	Example

i,j	Imaginary unit. As the basic imaginary unit SQRT(-1), i and j are used to enter complex numbers. For example, the expressions 3+2i, 3+2*i, 3+2j, 3+2*j and 3+2*sqrt(-1) all have the same value.	`>>z=2+4i`
		`>>z=2+4j`

abs	abs(x) is the absolute value of the elements of x. When x is complex, abs(x) is the complex modulus (magnitude) of the elements of X.	`>>z=2+4i`
		`>>abs(z)`

angle	Phase angle. angle(z) returns the phase angles, in radians	`>>z=2+4i`
		`>>angle(z)`

imag	Complex imaginary part. imag(z) is the imaginary part of z.	`>>z=2+4i`
		`>>b=imag(z)`

real	Complex real part. real(z) is the real part of z.	`>>z=2+4i`
		`>>a=real(z)`

conj	Complex conjugate. conj(x) is the complex conjugate of x.	`>>z=2+4i`
		`>>z_con=conj(z)`

complex	Construct complex result from real and imaginary parts. c = complex(a,b) returns the complex result A + Bi	`>>a=2;`
		`>>b=3;`
		`>>z=complex(a,b)`

Look up these functions in the Help system in MATLAB.

`Task 25: Complex numbers`

Given two complex numbers

$$c = 4 + j3, d = 1 - j$$

Find the real and imaginary part of c and d in MATLAB.

→ Use MATLAB to find $c + d, c - d, cd \; and \; c/d$.

Use the direct method supported by MATLAB and the specific complex functions **abs**, **angle**, **imag**, **real**, **conj**, **complex**, etc. together with the formulas for complex numbers that are listed above in the text (as you do it when you should calculate it using pen & paper).

→ Find also r and θ. Find also the complex conjugate.

`[End of Task]`

`Task 26: Complex numbers`

Find the roots of the equation:

$$x^2 + 4x + 13$$

We can e.g., use the **solveeq** function we created in a previous task. Compare the results using the built-in function **roots**.

Discuss the results.

Add the sum of the roots.

`[End of Task]`

9.5 Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + \; p_{n+1}$$

where $p_1, p_2, p_3, \ldots$ are the coefficients of the polynomial.

MATLAB represents polynomials as row arrays containing coefficients ordered by descending powers.

Example:

Given the polynomial:

$$p(x) = -5.45x^4 + 3.2x^2 + 8x + 5.6$$

In MATLAB we write:

```
>> p=[-5.45 0 3.2 8 5.8]
p =
   -5.4500         0    3.2000    8.0000    5.8000
```

[End of Example]

MATLAB offers lots of functions on polynomials, such as **conv**, **roots**, **deconv**, **polyval**, **polyint**, **polyder**, **polyfit**, etc. → Look up these functions in the Help system in MATLAB.

Task 27: Polynomials

Define the following polynomial in MATLAB:

$$p(x) = -2.1x^4 + 2x^3 + 5x + 11$$

→ Find the roots of the polynomial ($p(x) = 0$) (and check if the answers are correct)

→ Find $p(x = 2)$

Use the polynomial functions listed above.

[End of Task]

Task 28: Polynomials

Given the following polynomials:

$$p_1(x) = 1 + x - x^2$$

$$p_2(x) = 2 + x^3$$

→ Find the polynomial $p(x) = p_1(x) \cdot p_2(x)$ using MATLAB and find the roots

→ Find the roots of the polynomial ($p(x) = 0$)

→ Find $p(x = 2)$

→ Find the differentiation/derivative of $p_2(x)$, i.e., $p_2{}'$

Use the polynomial functions listed above.

[End of Task]

Task 29: Polynomial Fitting

Find the 6. order Polynomial that best fits the following function:

$$y = \sin(x)$$

Use the polynomial functions listed above.

→ Plot both the function and the 6. order Polynomial to compare the results.

[End of Task]

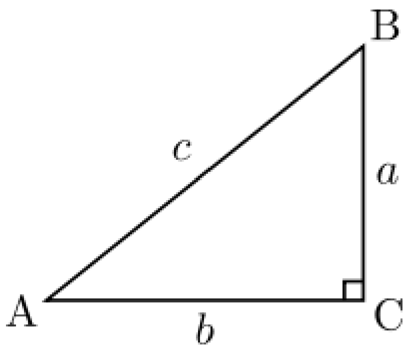
10 Additional Tasks

If you have time left or need more practice, solve the tasks below. Its highly recommended to solve these tasks as well, since some of these will most likely be part of the final test.

Task 30: User-defined function

Create a function that uses Pythagoras to calculate the hypotenuse of a right-angled triangle, e.g.:

```
function h = pyt(a,b)
% ..
…
h = …
```



Pythagoras theorem is as follows: $c^2 = a^2 + b^2$

Note! The function should handle that a and b could be vectors.

[End of Task]

Task 31: MATLAB Script

Given the famous equation from Albert Einstein:

$$E = mc^2$$

The sun radiates $385\ x\ 10^{24} J/s$ of energy.

→ Calculate how much of the mass on the sun is used to create this energy per day.

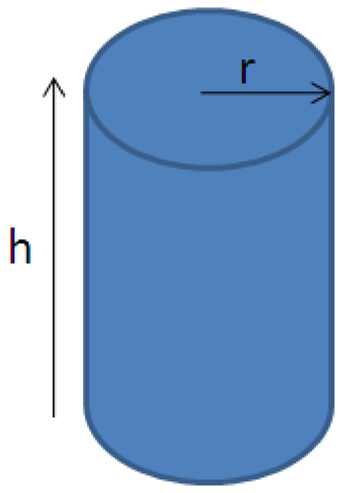
→ How many years will it take to convert all the mass of the sun completely? Do we need to worry if the sun will be used up in our generation or the next?

The mass of the sun is $2\,x\,10^{30} kg$

[End of Task]

Task 32: Cylinder surface area

Create a function that finds the surface area of a cylinder based on the height (h) and the radius (r) of the cylinder.



[End of Task]

Task 33: Create advanced expressions in MATLAB

Create the following expression in MATLAB:

$$f(x) = \frac{\ln(ax^2 + bx + c) - \sin\,(ax^2 + bx + c)}{4\pi x^2 + \cos\,(x - 2)(ax^2 + bx + c)}$$

Given $a = 1, b = 3, c = 5$

→ Find $f(9)$

(The answer should be $f(9) = 0.0044$)

Tip! You should split the expressions into different parts, such as:

poly = $ax^2 + bx + c$

num =…

den =….

f =…

This makes the expression simpler to read and understand, and you minimize the risk of making an error while typing the expression in MATLAB.

[End of Task]

Task 34: Solving Equations

Find the solution(s) for the given equations:

$$x_1 + 2x_2 = 5$$

$$3x_1 + 4x_2 = 6$$

$$7x_1 + 8x_2 = 9$$

[End of Task]

Task 35: Pre-allocating of variables and vectorization

Here we will use pre-allocating of variables and vectorization and compare with using a For Loop.

We will use the functions **tic** and **toc** to find the execution time.

We will create a simple program that calculates $y = cos(t)$ for t=1 to 100 000.

Create the following Script:

```
% Test 1: Using a For Loop
clear
tic
tmax=100000;
for t=1:tmax
    y(t,1)=cos(t);
```

```
end
toc
```

→ What was the execution time?

We will improve the Script by preallocating space for the variable y. Create the following Script:

```
% Test 2: For Lopp with preallocating
clear
tic
tmax=100000;

y=zeros(tmax,1); % preallocating

for t=1:tmax
   y(t,1)=cos(t);
end
toc
```

→ What was the execution time?

We will improve the Script further by removing the For Loop by using vectorization instead:

```
% Test 3: Vectorization
clear
tic
tmax = 100000;

t = 1:tmax; %vectorization

y = cos(t);

toc
```

→ What was the execution time?

Discuss the result.

[End of Task]

Task 36: Nested For Loops

Given the matrices $A \in R^{nxm}$ and $B \in R^{mxp}$, then

$$C = AB \in R^{nxp}$$

where

$$c_{jk} = \sum_{l=1}^{n} a_{jl} b_{lk}$$

In MATLAB it is easy to multiply two matrices:

```
>> A = [0 1;-2 -3]
A =
     0     1

    -2    -3
>> B = [1 0;3 -2]
B =
     1     0

     3    -2
>> A*B
ans =
     3    -2

   -11     6
```

But her you will create your own function that multiply two matrices:

```
function C = matrixmult(A,B)
…
```

Tip! You need to use 3 nested For Loops.

[End of Task]

Task 37: Prime Numbers

The first 25 prime numbers (all the prime numbers less than 100) are:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

By definition a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Create a MATLAB Script where you find all prime numbers between 1 and 200.

Tip! I guess this can be done in many ways, but one way is to use 2 nested For Loops.

[End of Task]

Task 38: Prime Number Function

The first 25 prime numbers (all the prime numbers less than 100) are:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

By definition, a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Create a MATLAB function where you check if a given number is a prime number or not.

You can check the function in the Command Window like this:

```
>> number = 4
>> checkifprime(number)
```

[End of Task]

Appendix A: MATLAB Functions

This Appendix gives an overview of the most used functions in this course.

Built-in Constants

MATLAB have several built-in constants. Some of them are explained here:

Name	Description
i, j	Used for complex numbers, e.g., z=2+4i
pi	π
inf	∞, Infinity
NaN	Not A Number. If you, e.g., divide by zero, you get NaN

Basic Functions

Here are some descriptions for the most used basic MATLAB functions.

Function	Description	Example
help	MATLAB displays the help information available	`>>help`
help <function>	Display help about a specific function	`>>help plot`

who, whos	who lists in alphabetical order all variables in the currently active workspace.	`>>who`
		`>>whos`

clear	Clear variables and functions from memory.	`>>clear`
		`>>clear x`

size	Size of arrays, matrices	`>>x=[1 2 ; 3 4];`
		`>>size(A)`

length	Length of a vector	`>>x=[1:1:10];`
		`>>length(x)`

format	Set output format	

disp	Display text or array	`>>A=[1 2;3 4];`
		`>>disp(A)`

plot	This function is used to create a plot	`>>x=[1:1:10];`
		`>>plot(x)`
		`>>y=sin(x);`
		`>>plot(x,y)`

clc	Clear the Command window	`>>cls`

rand	Creates a random number, vector or matrix	`>>rand`
		`>>rand(2,1)`

max	Find the largest number in a vector	`>>x=[1:1:10]`
		`>>max(x)`

min	Find the smallest number in a vector	`>>x=[1:1:10]` `>>min(x)`
mean	Average or mean value	`>>x=[1:1:10]` `>>mean(x)`
std	Standard deviation	`>>x=[1:1:10]` `>>std(x)`

Linear Algebra

Here are some useful functions for Linear Algebra in MATLAB:

Function	Description	Example
rank	Find the rank of a matrix. Provides an estimate of the number of linearly independent rows or columns of a matrix A.	`>>A=[1 2; 3 4]` `>>rank(A)`
det	Find the determinant of a square matrix	`>>A=[1 2; 3 4]` `>>det(A)`
inv	Find the inverse of a square matrix	`>>A=[1 2; 3 4]` `>>inv(A)`
eig	Find the eigenvalues of a square matrix	`>>A=[1 2; 3 4]` `>>eig(A)`
ones	Creates an array or matrix with only ones	`>>ones(2)` `>>ones(2,1)`
eye	Creates an identity matrix	`>>eye(2)`
diag	Find the diagonal elements in a matrix	`>>A=[1 2; 3 4]` `>>diag(A)`

Type "**help matfun**" (Matrix functions - numerical linear algebra) in the Command Window for more information, or type "**help elmat**" (Elementary matrices and matrix manipulation).

You may also type "**help <functionname>**" for help about a specific function.

Plotting

Plots functions: Here are some useful functions for creating plots:

Function	Description	Example
plot	Generates a plot. plot(y) plots the columns of y against the indexes of the columns.	`>X = [0:0.01:1];` `>Y = X.*X;` `>plot(X, Y)`
figure	Create a new figure window	`>>figure` `>>figure(1)`
subplot	Create subplots in a Figure. subplot(m,n,p) or subplot(mnp), breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot. The axes are counted along the top row of the Figure window, then the second row, etc.	`>>subplot(2,2,1)`
grid	Creates grid lines in a plot. "grid on" adds major grid lines to the current plot. "grid off" removes major and minor grid lines from the current plot.	`>>grid` `>>grid on` `>>grid off`

axis	Control axis scaling and appearance. “axis([xmin xmax ymin ymax])” sets the limits for the x- and y-axis of the current axes.	`>>axis([xmin xmax ymin ymax])` `>>axis off` `>>axis on`
title	Add title to current plot title('string')	`>>title('this is a title')`
xlabel	Add xlabel to current plot xlabel('string')	`>> xlabel('time')`
ylabel	Add ylabel to current plot ylabel('string')	`>> ylabel('temperature')`
legend	Creates a legend in the corner (or at a specified position) of the plot	`>> legend('temperature')`
hold	Freezes the current plot, so that additional plots can be overlaid	`>>hold on` `>>hold off`

Type “**help graphics**” in the Command Window for more information, or type “**help <functionname>**” for help about a specific function.

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	|

Complex Numbers

Functions used to create or manipulate complex numbers.

Function	Description	Example
i,j	Imaginary unit. As the basic imaginary unit SQRT(-1), i and j are used to enter complex numbers. For example, the expressions 3+2i, 3+2*i, 3+2j, 3+2*j and 3+2*sqrt(-1) all have the same value.	`>>z=2+4i` `>>z=2+4j`

abs	abs(x) is the absolute value of the elements of x. When x is complex, abs(x) is the complex modulus (magnitude) of the elements of X.	`>>z=2+4i`
		`>>abs(z)`

angle	Phase angle. angle(z) returns the phase angles, in radians	`>>z=2+4i`
		`>>angle(z)`

imag	Complex imaginary part. imag(z) is the imaginary part of z.	`>>z=2+4i`
		`>>b=imag(z)`

real	Complex real part. real(z) is the real part of z.	`>>z=2+4i`
		`>>a=real(z)`

conj	Complex conjugate. conj(x) is the complex conjugate of x.	`>>z=2+4i`
		`>>z_con=conj(z)`

complex	Construct complex result from real and imaginary parts. c = complex(a,b) returns the complex result A + Bi	`>>a=2;`
		`>>b=3;`
		`>>z=complex(a,b)`

Hans-Petter Halvorsen

E-mail: hans.p.halvorsen@usn.no

Blog: http://www.halvorsen.blog

University of South-Eastern Norway

www.usn.no

Introduction to MATLAB