



**ROYAL INSTITUTE
OF TECHNOLOGY**

Introduction to MATLAB

August 2007

Foreword

The compendium gives an introduction to MATLAB programming. It is so structured that it can be used as self-study materials. Readers are assumed to be familiar with basic concepts of programming and linear algebra (such as vectors and matrices etc).

This compendium is a revised version of an old compendium written by Karin Larsson and Lars Harrie, Lund University. The revision was first made by Johnny Bärlund and Huaan Fan in August 2006, with enhanced materials on matrix operations, new exercise problems and other minor additions. The revision has also utilized materials from two external sources:

- Eva Pärt-Enander and Anders Sjöberg (1998). *Användarhandledning för MATLAB 5, Appendix A*, Uppsala University, and
- Per Jönsson (2001). *Laborationer i MATLAB*, Malmö Högskola.

Further improvement has been made by Wan Wen in August 2007.

Stockholm, August 30, 2007

Huaan Fan

Contents

1	Start working with MATLAB command window
1.1	Starting MATLAB and the command window
1.2	Basic computation in the command window
1.3	More basic computations and the use of the semicolon
1.4	Standard functions
1.5	Hints for easy handle of the command window
2	Write a MATLAB program (m-file)
2.1	Create the m-file
2.2	Basic computation with the m-file and running the program
2.3	Vectors & matrixes
2.3.1	Basic about Vectors
2.3.2	Basic about Matrixes
2.3.3	More about Matrixes and Vectors
2.3.4	Writing vectors in another way
2.3.5	An short introduction to plot(x,y) and the help command
3	Input and output of data files.
3.1	More about variables
3.2	Save and retrieve variables
3.3	Basic input and output
3.4	Decimals and output format
4.	More on programming in MATLAB
4.1	Logical expressions
4.2	Controlling program flow
4.3	Structuring a program
5	More on graphics
6	Functions
7	Exercise problems


Introduction to Matlab

The name MATLAB is a acronym that stands for MATrix LABaratory. It is a program constructed for matrix computation, numerical analysis and graphics. The program makes it easy to perform technical computations. You not need to compile the program code before using the program. MATLAB is an interactive environment and you can use the code immediately after creating it. You don't need to define variables before using them, instead you just write for instance `a= 10` and the variable `a` is created and gets the value 10. You can use MATLAB as a pocket calculator or for more advanced programming routines. It is also possible to extend MATLAB with some toolboxes for more professional uses in some researches areas or different engineering fields. There are toolboxes for signal processing, optimisation, control theory, mathematical statistics and map uses and many more. The big strength in this program is its ability to handle matrixes and the many built-in functions that handle matrixes in an easy way.

1 Start working with MAT LAB command window

1.1 Starting MATLAB and the command window



Open the program MATLAB with a double click on this icon: , or you can also open it by clicking 'Start- All Programs- Matlab 7.1- Matlab 7.1'

A MATLAB window opens consisting of three parts, to the right is the **command window** and you can use it for instant computation. Another way to use MATLAB is to create new m-files for real programming, as we will see in chapter 2.

1.2 Basic computation in the command window

MATLAB can be used as a pocket calculator: After `>>` Write `5+4` followed by return

```
>> 5+4  
ans =9
```

Here, the mark `>>` is the command prompt in Matlab., which means that Matlab is ready for accepting a new command.

Power of is written by `^`. To compute 2^3 we write:

```
2^3  
ans=8
```

Multiplication by `*`

```
3*4  
ans=12
```

MATLAB works the way mathematicians think. Then it's easy to remember for instance that MATLAB works with radians for angels.

```
Try to write pi  
Ans=3.1416
```

Write: `cos(pi)` and you get `ans=-1`.

WARNING: NOTICE THAT MATLAB USES THE **POINT AS A DECIMAL COMMA!!!** It is a common mistake to write a decimal number with “,” instead of “.” as MATLAB wants it.

Try to write $x=3, 1416$ and see what happens!! Not the thing you wanted.
The reason for this is that the sign “,” is used in an other way by MATLAB:
You can use several commands on a single row:
 $2^5, 2*(3+2)$

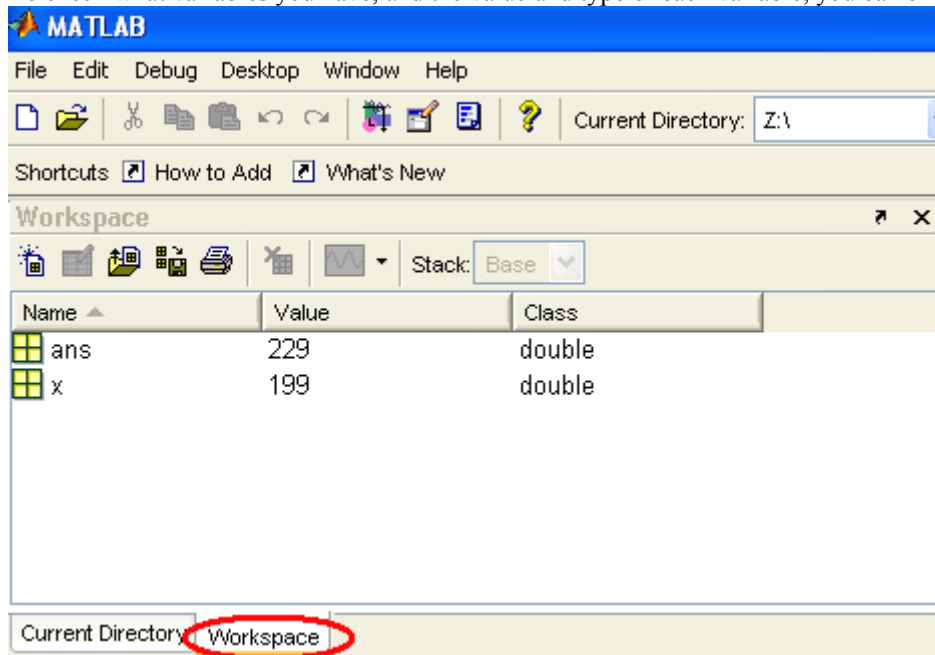
```
ans =  
    32  
  
ans =  
    10
```

If you do not add an own result variable the result is add to the variable *ans*. But it is recommended to use own variables in most cases:

x=14

```
x =  
    14  
  
y=3*x  
  
y =  
    42
```

To check what variables you have, and the value and type of each variable, you can click ‘Workspace’.



1.3 More basic computations and the use of the semicolon

Parentheses are used as usual:

```
u = 2*x - y;  
w = 2*(x-y); (*)  
exp((2-u)/(w-2)) (corresponds to e(2-u)/(w-2))  
ans =  
    0.7589
```

(*) Notice that the use of semicolon (;) means that MATLAB doesn't print the result just save the value for the variable for later use

Write $x=25;$ (Notice the sign “;”)

And you will not see the result. But if checking the ‘Workspace’, you can see that $x=25$ has already been recorded.

Write then x and you get the answer.

1.4 Standard functions

MATLAB includes most standard mathematical functions:

<code>abs(x)</code>	absolute value, i.e., $ x $
<code>sqrt(x)</code>	\sqrt{x}
<code>exp(x)</code>	e^x
<code>log(x)</code>	$\ln x$
<code>log10(x)</code>	10-logarithm, i.e., $\log x$
<code>sin(x)</code>	radians is default
<code>cos(x)</code>	radians is default
<code>tan(x)</code>	radians is default
<code>cot(x)</code>	radians is default
<code>asin(x)</code>	$\arcsin x$
<code>acos(x)</code>	$\arccos x$
<code>atan(x)</code>	$\arctan x$

1.5 Hints for easy handle of the command window

```
Write a=sqrt(24.13^2+13.53^2)
a =
27.6644
```

You did a mistake and want another name on the variable. You need not to write all the formula again; just try to **push the arrow up symbol** and **you recall previous input line** and then you use the mouse or the home button and just change the a to the new variable name distance.

```
distance=sqrt(24.13^2+13.53^2)
distance =
27.6644
```

We have also used the square root command `sqrt` here and also notice again the use of “.” as a decimal comma in Matlab. (**A common mistake!!!** is to write 24,13 instead of 24.13 as Matlab want it.

↑	Recall previous input line
a ↑	Recall input lines starting with a
ESC	CLEAR THE CURRENT LINE

2 Write a MATLAB program (m-file)

So far we have been working with the command window and have been more or less using the MATLAB as a calculator. Now it’s time to learn how to write real programs in MATLAB.

The major functionality in MATLAB is coded as so called m-files, which are plain text files having file extension `.m`. These files consist of source code written in the MATLAB programming language. All m-files can be viewed and edited by a standard text editor.

2.1 Create the m-file

To the top and left side of your window you click on **File** and **choose new m-file**. Or you can use the white bottom that looks like an empty page for the same purpose. A new window opens which is a text editor where

you can write your MATLAB programs. It's good to have a routine to **give this m-file a name immediately**. Do this by going to file (on the top left side) choose save as and write a name in the line for filename at the bottom, save the name by clicking on the save bottom to the right.

2.2 Basic computation with the m-file and running the program

You can do everything you did in chapter 1 with the text editor and with the m-files. The difference is that with the m-files you can do many things maybe solve a more complex problem and use many commands after each other before executing them.

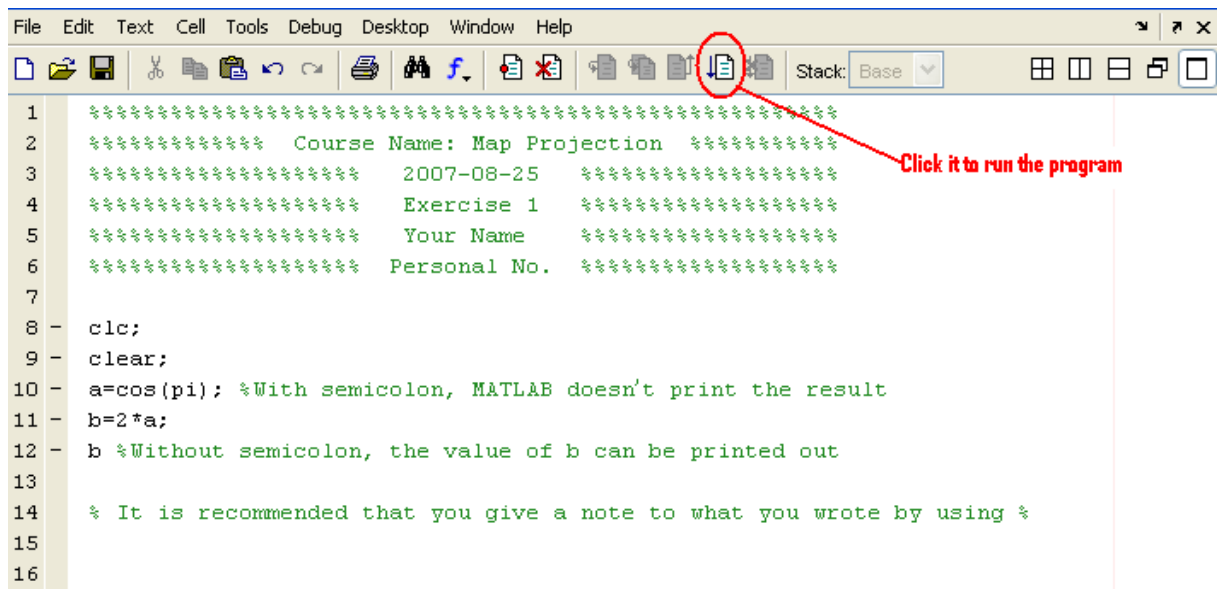
Try for instance to write: %

After the “%” sign you write a message to yourself and MATLAB will ignore everything on the line %after the % sign

```
a=cos(pi);
b=2*a
```

Or something else you like to try

Running the program



To run the program you just need to push the bottom showed in the picture above (white bottom with a blue arrow pointing down). Then you switch to the command window we used in chapter 1 and you will see the result $b=-2$.

If only an error message occur and nothing else

The m-file should be stored in the **current directory**. The m-file can also be run from the *command window* by simply writing the name of the program. But if you do not have stored the m-file in the current directory (or another associated directory, but this details is not described here) MATLAB will not be able to find your program and you will get an error message.

2.3 Vectors & matrixes

2.3.1 Basic about Vectors

To generate the vector $x = (1, 2, 3)$ you write:

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

(The sign “[” is done with Alt Gr and Bottom 8)

MATLAB answers with

```
x =
     1     2     3
```

To use a single element in a vector
x(2)

MATLAB shows the value of the second element
ans =
2

We can also change a value of a single element:
x(2)=4.2
Write: x

MATLAB answers
ans=
1.0000 4.2000 3.0000

We have used one decimal in the matrix and all the answers will be given with decimals.

2.3.2 Basic about Matrixes

Create your first Matrix in two different kinds of ways

Write it out explicit
A = [1 2 3 4
5 6 7 8
9 10 11 12]

Write also the same thing **using “;” semicolon as a new row**

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

Run the program

And you get

A =
1 2 3 4
5 6 7 8
9 10 11 12

B =
1 2 3 4
5 6 7 8
9 10 11 12

Hint I have found it's **easier as a beginner to write matrixes like the example of A above**. Then you get two advantages it's easier to understand what you have done in example A than in B and you just use the semicolon sign only for its other purpose to hide the computation results from the command window. Then it's easier to remember the use of the different signs like the “;” and “,” signs and others.

Remember if you write;
y=3^4*5; %then the **“;” hide the value of y** in the command window when running the program.

2.3.3 More about Matrixes and Vectors

To retrieve the **size of a vector or matrix** you should use the `size` command:

```
[numRows, numColumns] = size (A)
```

```
numRows = 3
```

```
numColumns = 4
```

You can define both column and row vectors using:

```
vcol = [1;2;3;4], vrow = [5 6 7 8]
```

```
vcol =
```

```
1
```

```
2
```

```
3
```

```
4
```

```
vrow =
```

```
5 6 7 8
```

The best way though to make column vectors is to use the transpose sign in that is `'` in MATLAB.

```
Write: vcol=[1 2 3 4]'
```

```
And you get
```

```
1
```

```
2
```

```
3
```

```
4
```

The functions are applied on each element in a vector or matrix

```
sqrt(vcol)
```

```
ans =
```

```
1.0000
```

```
1.4142
```

```
1.7321
```

```
2.0000
```

Be aware of that MATLAB here uses a short form of the full expression (please try it):

```
[sqrt(vcol(1)); sqrt(vcol(2)); sqrt(vcol(3)); sqrt(vcol(4))]
```

2.3.3 Arithmetic operations on vectors and matrixes

Suppose we have a vector $t = (-1, 0, 2)$. The commands

```
u=3*t
```

```
v=t+2
```

gives $u = (-3, 0, 6)$ and $v = (1, 2, 4)$. Note that in the second row we have used a MATLAB short form of the full expression:

```
v = t + [2, 2, 2]
```

Create the vectors $s = (-1, 1, 2)$ and $t = (2, 3, 4)$. Then test the commands:

```
u=s+t
```

```
v=s.*t
```

Notice the command `.*`. **The point before the * means that the multiplication is done element by element. Without the point before * it means matrix multiplication.**

```
w=s./t
z=t.^2
```

which gives $u = (1, 4, 6)$, $v = (-2, 3, 8)$, $w = (-1/2, 1/3, 1/2)$ and $z = (4, 9, 16)$. In this case the operations are performed on each element. A dot before the operator is used as a short writing of element by element operation between vectors.

```
v=s.*t
```

is a short writing of the expression:

```
v=[ s(1)*t(1), s(2)*t(2), s(3)*t(3) ]
```

The transpose of matrixes and vectors

Write the following sentences in command window:

```
C=[1 2 4]; %Create a vector
C' %which means CT in MATLAB code
```

Now we can get the result as below:

```
ans=
[1
 2
 4]
```

Try it also on a matrix

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

```
A' gives [1 3
          2 4]
```

Inverting a matrix

To invert a the matrix A you write **inv(A)**

The determinant of a matrix

Create a matrix A as

```
A=[1 2
   2 4]
```

```
A =
```

```
1 2
2 4
```

```
det(A) %Computes the determinant of the matrix
ans = 0
```

Eigenvalues and eigenvectors of a matrix A

[V,D]=EIG(A) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that $A*V = V*D$.

You might be familiar with **scalar product and vector cross product** from linear algebra courses. To use them in MATLAB you write as follows (if you are not familiar with these expressions you do not have to be concerned; they are not essential for this course):

Scalar product:

```
dot(s,t)
(defined as  $\text{dot}(s,t)=\sum s_i*t_i$ )
```

Vector cross product

```
cross(s,t)
(defined as  $\text{cross}(s,t)=[s_2*t_3-s_3*t_2, s_1*t_3-s_3*t_1, s_1*t_2-s_2*t_1]$ )
```

2.3.4 Writing vectors in another way

Create the vector bellow

```
vector = [0 1 2 3 4 5 6 7 8]
```

This kind of vector can be done in a more easy way

The operator (:) can be used to create vectors:

```
vector = 0:8 %Step=1 is standard
```

is equal to

```
vector = [0 1 2 3 4 5 6 7 8]
```

which gives

```
vector =  
    0     1     2     3     4     5     6     7     8
```

To define a vector with the step 0.5 you write:

```
vector2 = 0:0.5:2 %Here you choose step=0.5
```

which gives the answer:

```
vector2 =  
    0    0.5000    1.0000    1.5000    2.0000
```

The general form of the notation is

x=a:h:b %a is the start value, h the increments and b the last value

2.3.5 An short introduction to plot(x,y) and the help command

Here we will learn how to make a mathematical plot of the graph for $y = f(x)$.

In our example we make the graph of $y = x^2$.

First we create a vector with different x-values as we learned in 2.3.4

x=-6:6;

Compute y values for each x-value

y=x.^2; %Notice the use of “.” **the point** here. We want every x-value in the vector x to be squared.

Create the graph with the plot command

plot(x,y)

Then run the program and you will get a graph in another window.

Notice if you do not **use the semicolon** afterwards when you create the x and y vectors you will get a lot of unnecessary values in the command window.

MATLABS help commands (Use this important help command to see the syntax in MATLAB)

There are, at least, three possibilities to get help in MATLAB.

1) Go to the help functions in the top menu.

2) **If you know a command (e.g. plot), but is unsure how to use it**, you can write:

help plot

```
PLOT Linear plot.
```

```
PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,  
then the vector is plotted versus the rows or columns of the  
matrix, whichever line up. If X is a scalar and Y is a vector,  
length(Y) disconnected points are plotted.  
etc.
```

3) If you want information about a certain concept, you can use the `lookfor` command, e.g.:

lookfor logarithm (You get information about different predefined functions dealing with logarithms)

LOGSPACE Logarithmically spaced vector.
LOG Natural logarithm.
LOG10 Common (base 10) logarithm.
LOG2 Base 2 logarithm and dissect floating point number.
BETALN Logarithm of beta function.
GAMMALN Logarithm of gamma function.
LOGM Matrix logarithm.

3 Input and output data files

3.1 Different types of variables

MATLAB is case sensitive.

Notice that MATLAB distinguish between capital letters and lower case letters in the name of the variables. **Dist and dist are two different variables** and the variables `a` and `A` are also different.

Three types of variables

MATLAB has three types of variables: ***numbers, Boolean (true/false) and strings***. A string variable is a variable that contains text and that you can't calculate with. As mentioned above, you do not have to declare the variables. But when you design a string variable you always have to use **quotes**, as in:

```
myString='hello'  
myString =  
           hello
```

The Boolean variables are described in chapter 4

3.2 Save and retrieve variables

To present all defined parameters, you can the command 'who':

```
who  
Your variables are:  
  ans  u    w    x    y
```

Try also `whos`. What is the difference?

MATLAB remembers all the variables during a session. To remove the variables from the memory you write:
`clear`

Remove all your variables (using `clear`) and try `who` again.

Save	Save all the variables in the file MATLAB.mat . This is the default filename if no name is specified.
save filename	Save the variables in the file filename.mat Be aware! The file is saved in the <i>current directory</i> .

save filename v1 v2	Save the variables $v1$, $v2$
save filename v -ascii	Saves the variable v in a text-file (ASCII). Uses 8 decimals.
save filename v -ascii -double	Uses 16 decimals.
Load	Retrieve all variables in the file <i>MATLAB.mat</i>
load filename	Retrieve data from the specified file. There are some details here that can be important. See the help function.

3.3 Basic input and output

Make the program ask for a value and put the answer from the keyboard in a variable

```
x = input('Give a value of x: '); %You will see the text and the program
%stops until you write a value followed by enter then x gets the value you
%wrote
```

You can then **display the value with**

```
disp('The element x has the value:')
disp(x)
```

3.4 Decimals and output format

MATLAB's floating-point numbers are stored with a relative precision of roughly 16 decimal digits and MATLAB is always calculating with all the digits but you have the possibility to choose how many digits you want to display.

Write **format long** followed by pi and see how many decimals you get.

Then write **format short** and see how many decimals pi now gets. Format short with 4 decimals is default. Try also **format compact**.

Formatted output can be created by **sprintf**:

```
x=1;
y=23;
disp(sprintf('The value of x is %u and y is %u',x,y))
```

You can also write (which should be familiar from other languages; MATLAB here uses *Fortran* syntax):

```
disp(sprintf('The value of x is %8.6f and y is %4.2f',x,y))
```

The bold values 6 and 2 above chooses the number of decimals

There are other ways to print the formatted output; you can input 'help sprintf' to read more.

4. Programming in MATLAB

4.1 Logical expressions

The logical expressions are important in the conditional statements and the loops. Here they are listed.

<u>operator</u>	<u>means</u>
<	obvious
<=	obvious
==	equal to (be aware that "==" is an assign operator)
>	obvious
>=	obvious
~=	not equal to

<u>logical operator</u>	<u>means</u>
&	and
	or
~	not

4.2 Controlling program flow

Several fundamental programming structures exist to control the program execution flow. These flow control statements generally consist of selection and repetition, or any combination of these two.

Conditional statements

```
if logical expression
    commands ...
end
```

or a more general case

```
if logical expression
    commands 1
else
    commands 2
end
```

You can also use `elseif`

Example. Let a and b be two numbers. If $a > 0$ or $b > 0$ we create $c = a + b$ otherwise $c = 0$ (be aware of that this example require the parameters a and b to be defined previously in the program. Here follow a description of three of the most common ones if, while and for.

```
if a > 0 | b > 0
    c = a+b
else
    c = 0
end
```

while loop

```
while logical expression
    commands
end
```

Example. To give the sum of all integers between 0 and 100 we write:

```
sum = 0;
i=0;
while i <= 100           %This will continue in a loop until i>100
    sum = sum + i;
    i = i + 1;
end
disp(sprintf('The sum is %u',sum))
```

for-loops

The general form of a for-loop is:

```
for variable = expression
    command
end
```

Example. Compute the sum of 2^i when i starts with the value 1 continues with 2,3,4 until it has reach the end value 10.

```
sum = 0;
for i = 1:10
    sum = sum + 2^i;
end
sum
```

Example. To compute the sum of all even integers between 0 and 100 we write

```
sum = 0;
for i = 0:2:100      %This loop starts with i=0 and
    sum = sum + i;
end
disp(sprintf('The sum is %u',sum))
```

4.3 Structuring a program

When you write a program it is essential that the program is well structured and documented. Notice the following (and compare with the program example above):

- 1) A comment line starts with the percentage sign (%). What you write after that sign will not be read by MATLAB; these lines are only made for making the program easier to read by humans. Use many comments to explain the different parts of the program.
- 2) Always start a program with a header that includes the name of the program, author, date and a description.
- 3) Use as informative variable names as possible (such as *startX*, *dist*, etc.). Variable names such as *i*, *j*, etc. should only be used for indexing parameters.
- 4) Always clear the memory of all variables (`clear`) in the beginning of the program, and if applicable also the figure (`clf`).
- 5) Always use indentations for conditional statements and loops (see below).

Example of a program

```
% *****
%
%                               pointDistance.m
%
% Author: Lars Harrie
% Date: 2005-05-11
%
% Description: The program computes the (Euclidean) distance
% between two points. The points are user input values.
% The distance is computed by the Pythagorean theorem.
%
% *****
%
clear %A good routine is to always start a program with clear
format short
%
% Give the input points
startX=input('Give startpoint (x-coordinate): ');
startY=input('Give startpoint (y-coordinate): ');
endX=input('Give endpoint (x-coordinate): ');
endY=input('Give endpoint (y-coordinate): ');
```

```

%
% Compute the distance and display it
dist = sqrt ( (endX-startX)^2 + (endY-startY)^2 );
%
% Display the distance. If the distance is longer than
% 100 then 2 decimals are used otherwise 4.
if (dist>100)
    disp(sprintf('The distance between the points is %8.2f', dist))
else
    disp(sprintf('The distance between the points is %8.4f', dist))
end
%
% End of program
%
```

5 More on graphics

To plot the function $y = \sin(2\pi x)$ in the interval $[0,1]$ we write:

```

x=linspace(0,1,100); %Or use x=0:0.01:1; as in chapters (2.3.4 and 2.3.5)
% The function linspace(X1, X2, N) generates a row vector of N linearly
% equally spaced points between X1 and X2
y=sin(2*pi*x);
plot(x,y)
title('The function y=sin(2*pi*x)')
xlabel('x is in radians')
ylabel('y')
```

Go into the MATLAB help functions and read more about the command **plot**. Be aware of that the input parameters to the functions `plot`, `xlabel` and `ylabel` are all strings; this implies that you have to use quotes around the input values.

If you want to clear the figure you write:
`clf`

The general form of the plot command is:
`plot(x,y,str)`

where *str* is a string that specifies e.g. type of line, type of point, colour (see the table below). For example, if you want to plot a blue, dashed line you write:
`plot(x,y, 'b--')`

Points	Lines
.	- solid line
*	-- dashed line
square	-· point-dashed line
Diamond	: point line
hexagram	Colours
o	g green
+	m magenta

x	b	blue
<	c	cyan
>	k	black
^	Y	yellow
v	r	red

Other useful plot commands are:

hold on	keep everything in the figure
hold off	end “hold on”
grid on	plot a grid
grid off	remove the grid
title (txt)	print a title
xlabel (txt)	write an x-label
ylabel (txt)	write an y-label
text (x, y, txt)	write at position (x; y)
gtext (txt)	the user can specify the position with the mouse
legend (txt)	creates a legend
axis	give you the possibility to scale the axes

hold on is useful when, for example, you would like to add two curves into the same plot. Try the following:

```
x=linspace(0,1,100);
y=sin(2*pi*x);
z=cos(2*pi*x);
plot(x,y,'b')
hold on
plot(x,z,'r')
```

If you had not used the command `hold on` here the first curve would have been removed from the figure when the second curve was drawn.

6 Writing your own functions

Function files is a special type of m-files. What characterise a function is that it returns a value. As a simple example we can take the function `fun1` (which returns the value f):

Firstly, create a new `.m` file, and write the following sentences:

```
function f=fun1(x);
f=x.*sin(x);
```

Save this file as ‘`fun1.m`’, which has the same name as the function. You cannot run the function file. After you have created the function `fun1` you could try to type the following sentences in command window:

```
x=linspace(0,12*pi,100);
y=fun1(x);
plot(x,y,'b-.')
grid on
```

```
xlabel('x')  
ylabel('y')  
legend('x*sin(x)')
```

In this example you have called the function *fun1* from the command window. It is also possible to call a function from a program (m-file) or another function.

WARNING Be aware of that a file that contains a function must have the same name as the function.

7. Exercise Problems

- Plot a graph for $y=\cos(x)$, where x is a vector that starts with the value 0 and goes to 10. You shall make a graph that shows the function smooth enough. Read (2.3.4) and (2.3.5)

- Matrices A and B are defined as follows:

$$A = \begin{bmatrix} 3 & 6 & 7 \\ 2 & 6 & 5 \\ 3 & 2 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Compute $A*B$ and $A.*B$ and explain the difference. Compute also the determinant of A and B and their inverse matrices, respectively.

- Compute and display matrix T :

$$T = \begin{bmatrix} 3^2 & \cos(\pi) \\ 1 & e^3 \end{bmatrix}$$

Change one of the elements to 10 using $T(2,1)=10$ and display T again. Explain what happened.

- (Inverse matrix and eigenvalue analysis)
Let A denote a matrix of dimension 5×2 :

$$A = \begin{bmatrix} -0.246 & 754 & 45 & -2.047 & 670 & 68 \\ +1.896 & 681 & 98 & +0.810 & 309 & 87 \\ -1.649 & 927 & 53 & +1.237 & 360 & 81 \\ +0.992 & 817 & 41 & -0.119 & 639 & 41 \\ +0.392 & 872 & 97 & -0.919 & 592 & 75 \end{bmatrix}$$

- Compute matrix N which is defined as:

$$N = A^T A \quad (\text{where } ^T \text{ denotes transpose})$$

- Compute the inverse matrix of N
- Compute the eigenvalues (λ_1, λ_2) of the matrix N and the corresponding eigenvectors (v_1, v_2)

- Write a program that reads an input value (x) and prints $\sin(x)$ with 10 decimals using “disp(sprintf...“ (see 3.3 and 3.4 in the Matlab manual)
- Write a program that computes the sum (**hint:** use the “for” command in chapter 4.2):

$$\sum_{n=1}^{10} n^n = 1^1 + 2^2 + \dots + 10^{10}$$

- Write a function that computes the volume of the sphere $\left(V = \frac{4\pi r^3}{3}\right)$, where the radius (r) is input value to the function. Then write a program that calls the function and plot the volume of a sphere for radius between 0 and 10 meters. The figure should have appropriate labels and titles.

- (Bilinear interpolation)

Write a Matlab program that estimates the height for an arbitrary point based on a Digital Elevation Model (DEM) using bilinear interpolation. The DEM data is stored in a text file (DEM.txt). This DEM is constructed by National Land Survey of Sweden (LMV) and therefore may only be used for the exercise. You are not allowed to use it for any other purposes. The DEM has the following metadata:

Resolution = 50 m (in both North and East direction)
North Min = 6490000 (in Swedish reference system RT90)
East Min = 1520000 (in Swedish reference system RT90)
Size = 101x101

The values in the DEM is organised as the lower left cell (North Min, East Min) has the index: Row=101, Column=1

Bilinear interpolation

Bilinear interpolation requires that the input points are given in a regular grid. It is defined as (using notations in the figure below (Figure 1). Be aware of that a mathematical definition of the axes is used and **NOT** the definition according to RT90):

$$z(x_p, y_p) = z_1 + (z_2 - z_1) \cdot w + (z_3 - z_1) \cdot u + (z_1 - z_2 - z_3 + z_4) \cdot uw \tag{1}$$

where

$z(x_p, y_p)$ is the interpolated value for the point $P(x_p, y_p)$, i.e. the point to be interpolated

z_1, z_2, z_3, z_4 are z-values of the four points closest to point $P(x_p, y_p)$

$w = (x_p - x_l) / \Delta x$

$u = (y_p - y_l) / \Delta y$

$\Delta x, \Delta y$ denote the spatial resolution of the DEM.

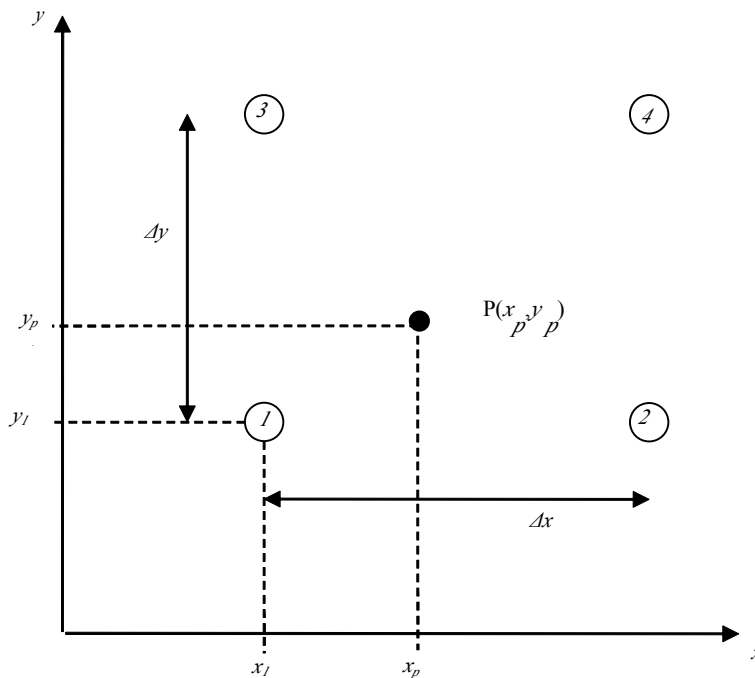


Figure 1: Bilinear interpolation.

Requirement: The program should ask the user to input the coordinates of the point to be interpolated. If the coordinates are not within the range of the DEM, a warning message should appear and then the program terminates. If the user specify coordinates within the DEM he/she should get a message such as: “The value of the interpolated point is 22.0 m”.

Hint: You are advised to check the Matlab functions: *ceil*, *floor* and *mod*.

9. Drawing a circle through three given points

Write a Matlab program that allows the user to click in the figure using the mouse (use the function *ginput*). Then a circle should be drawn that passes through these three points; and also the centre point O of the circle should be marked. See the figure below (Figure 2).

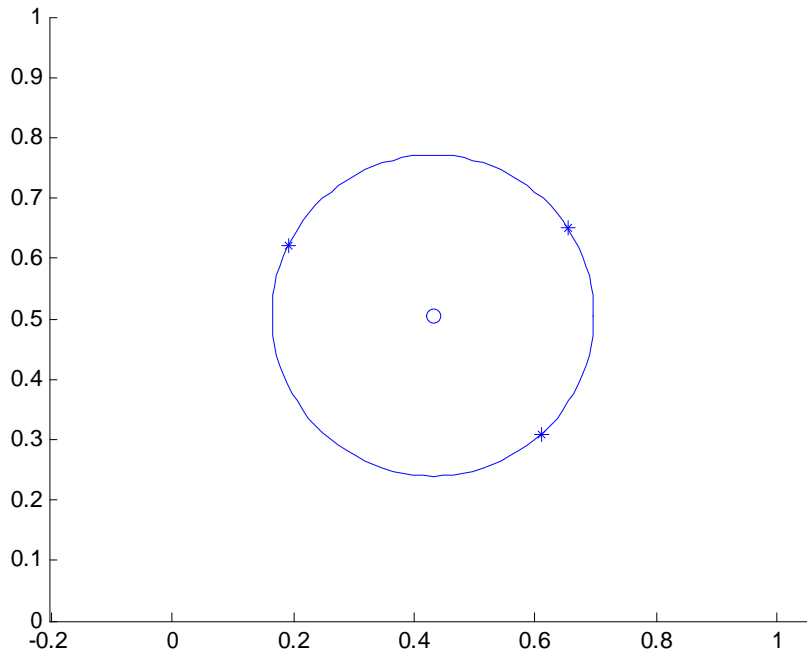


Figure 2: A circle passes through three points.

Hint: To determine a circle, one should know three parameters: the radius (R) and the coordinates (x_0, y_0) of the centre O of the circle. These three parameters can be solved from the following three non-linear equations:

$$\begin{cases} (x_1 - x_0)^2 + (y_1 - y_0)^2 = R^2 \\ (x_2 - x_0)^2 + (y_2 - y_0)^2 = R^2 \\ (x_3 - x_0)^2 + (y_3 - y_0)^2 = R^2 \end{cases}$$

By introducing three new variables:

$$\begin{cases} a = R^2 - x_0^2 - y_0^2 \\ b = 2 \cdot x_0 \\ c = 2 \cdot y_0 \end{cases}$$

The three non-linear equations can be reduced into three linear equations:

$$\begin{cases} a + x_1 \cdot b + y_1 \cdot c = x_1^2 + y_1^2 \\ a + x_2 \cdot b + y_2 \cdot c = x_2^2 + y_2^2 \\ a + x_3 \cdot b + y_3 \cdot c = x_3^2 + y_3^2 \end{cases}$$

When a , b and c are solved from the last three equations, R , x_0 and y_0 can then be easily obtained:

$$\begin{cases} R = \sqrt{a + x_0^2 + y_0^2} \\ x_0 = \frac{b}{2} \\ y_0 = \frac{c}{2} \end{cases}$$

10. Least squares adjustment

A geodetic survey has $n=5$ measurements and $m=2$ unknown parameters. The corresponding observation equations are as follows:

$$\underset{5 \cdot 1}{L} - \underset{5 \cdot 1}{\varepsilon} = \underset{5 \cdot 2}{A} \underset{2 \cdot 1}{X}$$

where:

L denotes the observation vector

ε denotes the measurement errors

A denotes the design matrix and

X denotes the unknown parameters to be determined

$$L = \begin{bmatrix} 0 \\ -8.81 \\ +14.83 \\ 0 \\ -9.12 \end{bmatrix} \quad (mm), \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \end{bmatrix}, \quad \underset{5 \cdot 2}{A} = \begin{bmatrix} -0.246 & 754 & 45 & -2.047 & 670 & 68 \\ +1.896 & 681 & 98 & +0.810 & 309 & 87 \\ -1.649 & 927 & 53 & +1.237 & 360 & 81 \\ +0.992 & 817 & 41 & -0.119 & 639 & 41 \\ +0.392 & 872 & 97 & -0.919 & 592 & 75 \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

When all observations are un-correlated with equal standard errors, the least squares estimate X is given by the following equation:

$$X = (A^T A)^{-1} A^T L$$

Write a MATLAB program (m-file) to compute the least squares estimate X . **Hint:** Use the result of exercise 4.

Requirements:

- The program should be able to handle any arbitrary values for n and m
- Numerical values of L and A should be read from an input data file (in ASCII format) and
- the final results should be saved in an output data file.