



Getting To Know Matlab

The following worksheets will introduce Matlab to the new user. Please, be sure you really know each step of the lab you performed, even if you are asking a friend who has a better understanding of computers or Numerical Recipes. If there is anything you do not understand or you are not sure you understand, ask your advisor.

Please note:

1. All text that is *italic and underlined* is to be typed in Matlab.
2. Using the \updownarrow arrows you can scroll in the previous Matlab commands, so you do not have to re-type everything.
3. When you have the  sign please call your advisor to present the results.
4. When you have the  sign, you should save the m-file you have written for the job, and hand it to your advisor at the end of the lesson.

1. What is Matlab?

Matlab is a technical computing environment for high-performance numeric computation. It integrates numerical analysis, matrix computation and graphics in an easy to use environment - without using traditional programming. The name Matlab stands for Matrix Laboratory, and indeed Matlab essentially works with one type of object, a rectangular matrix. Scalars are referred to as 1-by-1 matrices and vectors are matrices with only one row or column. Matlab allocates storage automatically according to the abilities of the computer. Since Matlab's language is optimized for manipulating matrices, the result is an economical and easy to use way of expressing matrix operations.

The principle is quite simple: you load the a matrix, and using the matrix manipulations functions of Matlab you can process it. This introduction is devoted to get you more familiar with Matlab's working environment.

2. Entering Matlab

Now let us begin our trip into the fascinating world of *Matlab*. In order to get into Matlab you first have to go into the Start button at the bottom-left corner of the screen. Click it, and then click the Matlab icon. Once you have entered

Matlab, you will see a `>>` prompt. Before you begin working change the working directory by typing:

`cd c:\usr`

If the folder does not exist – create it!

3. General Issues

3.1 Arithmetic operations

We will first look at the simple arithmetic operations: +, -, *, /, \, ^.

Type:

`2+3, 2-3, 2*3, 2/3, 3\2, 2^3`

The expressions `2/3` and `3\2` have the same numerical value (check!). The difference between the two operations becomes apparent (and is very crucial) for operations on matrices (as will be shown later). The following mathematical functions are also available: ABS, SQRT, LOG, SIN, and COS.

How do we define variables?

Unlike many programming languages, Matlab does not require prior definition of the variables. You can simply write: **variable name = expression**. For example:

`a = sin(64) + 2;`

If you do not specify the name of the variable, Matlab automatically creates the variable **ans**. Type : `3+2`. You will get `ans = 5`. Let's give you a little exercise:

calculate $\left(1 + \frac{2}{n^2}\right)^n$ for $n=3, 7$.

3.2 Getting workspace information

* The information about the variables active in your Matlab session can be listed by typing: `who`. If more information is required about these variables, such as the size of the variables, you can type: `whos`.

* Matlab processes Unix commands, so you can use the commands: `ls`, `type`, `delete`, `cd`, `pwd`, `help`.

* If you want to learn about the purpose of a Matlab function and a list of its arguments, you can type: `help function-name`. What do you get for `help sin`.

3.3 Graphs

Matlab is very useful when you want to outline drawings and graphs. Let's try to plot the function: $y = e^{-at} \cos(\omega t)$, for $a = 2$, $w = 5$, and $t = 0-10$.

`a=2;`

`w=5;`

`t=0:10;`

`y=exp(-a*t) .* cos(w*t);` Note: We write '`.*`'! We'll explain why later.

`plot(t,y,'r-');`

title('My first graph');
xlabel('t');
ylabel('y=exp(-a*t) * cos(w*t)');

3.4 Getting user input from the keyboard

User input from the keyboard can be obtained with the INPUT function.

n=input('Enter number');

4. Matrix Manipulation

4.1 Definition of 1-D array (vector):

We can define a vector in a few ways:

v1=1:5 gives: _____

v2=0: 3: 27 gives: _____

4.2 Definition of 2D array (matrix)

Let's define the matrix A:

1	2	3
9	8	7
0	2	3

We can do so by typing (pay attention to the spaces between the numbers):

A=[1 2 3 ; 9 8 7 ; 0 2 3] or,

A=[1,2,3;9,8,7;0,2,3] or,

A=[1 2 3 [enter]

9 8 7 [enter]

0 2 3]

Remember: separate the matrix elements with blanks or commas, surround the matrix with brackets [], Use the ';' to indicate the ends of rows or go to new line.

4.3 Definition of zeros matrix, ones matrix, and the unity matrix.

* We can define a zeros matrix by : Z1=zeros(3); Z2=zeros(3,2);

* We can define a ones matrix by: O1=ones(4); O2=ones(2,5);

* The identity matrix can be defined as: I=eye(3) ;

These definitions can be useful when you want to pre-allocate memory for your result matrix.

4.4 Matrix Operations

When you want to add/subtract a constant value from the whole matrix, you can simply write:

$$\underline{B=A-3};$$

Addition and subtraction of matrices is possible when the two matrices have the same dimension. Look at the result of:

$$\underline{A+B, A-B}.$$

The special character ' denotes the transpose of a matrix.

Look at

$$\underline{C=B'}.$$

For multiplication the situation is a little bit more complex. For matrices, we can define **element by element** operation or **matrix** operation.

A * B is: _____

and A .* B is: _____.

So, when we want to refer to element by element operation we have to add a dot before the operation sign. When we want to refer to the normal definitions of matrix operations we omit the dot.

There are two definitions for matrix division in Matlab.

$$X1=A\B, \text{ means } X1=A^{-1}*B.$$

$$X2=B/A, \text{ means } X2=B*A^{-1}.$$

As you remember from Linear Algebra, X1 and X2 are not the same!

4.5 Creating a larger matrix

It is possible to create a larger matrix from smaller matrices. Look at:

$$C=[A,B]; C=[A;B]$$

4.6 Referring to matrix elements

Let's look at matrix A. We can refer to each matrix element separately. Find out who are A(1), A(3), A(5), A(8). As you can see Matlab orders the matrix elements as following:

$$A(1) \ A(4) \ A(7)$$

$$A(2) \ A(5) \ A(8)$$

$$A(3) \ A(6) \ A(9)$$

We can also refer to a certain matrix element, in the i'th row and the j'th column by: A(i,j); Look at:

$$A(2,3).$$

4.7 Referring to matrix columns, matrix rows and sub-matrices

We can refer to column j of matrix A , by typing: $A(:,j)$. We can also refer to row i of the matrix by typing: $A(i,:)$. Calculations can be more efficient when we want to refer to columns or rows as a whole. Let's look at the following exercise:

First we will find $A(:,3)$ and $A(2,:)$.

We can also look at a sub matrix of A . For instance, if we want to refer to the first 2 rows of matrix A , we can write: $D=A(1:2, :)$, then $D=$ _____.

E is defined as: $E=A(1:2,3)$, then $E=$ _____.

F is defined as: $F=A(1:2,2:3)$, then $F=$ _____.

Fairly sophisticated effects can be obtained using the following references: $A(:,[1\ 3]) = B(:,2:3)$; which replaces the first and third columns of A with the last two columns of B . In general, if v and w are vectors with integer components, then $A(v,w)$ is the matrix obtained by taking the elements of A with row subscripts from v and column subscripts from w . If $v=[1,2]$ and $w=[1,3]$ then $A(v,w)=$ _____.

This nice trick is called **Tony's Trick**. Another use of this trick is the following:

```
a=[17 29 31];  
V=a([1 1 1], :);
```

What is V ?

Remember this trick when you deal with the zooming problem.

5. M-files

Matlab is usually used in a command driven mode, and executes your commands immediately after you have entered them. Matlab can also execute sequences of commands that are stored in files. Matlab M-files are plain ASCII files. You can write a sequence of Matlab commands using a standard editor such as the notepad and save the file as: filename.m. When you want to run the sequence of the commands you can use the utility run M-file from your Matlab menu. You can put remarks in your code (which is very useful, for you and for the understanding of your advisor) with the % sign before the remark line.

The function *what* gives you a list of all the m-files available.

Two types of m-files can be used: **scripts**, which merely automate long sequences of commands, and **functions** which provide extensibility to Matlab, i.e. you can add your own functions. A function differs from a script in that arguments may be passed, and variables that are defined and manipulated inside the file are local to the function.

We will write a script file and a function file to illustrate their use.

The m-files will be saved in the directory: c:\usr.

5.1 A script file to load a matrix.

Both formatted text and binary files can be read into Matlab. We will now load a file that is saved as a binary file into Matlab. The first step is opening the file with the FOPEN command. This command assigns a special file identifier to the file opened, so it can refer to it later on. A file can be opened for reading, writing or appending.

The file `targill.dat` contains the coefficients of the matrix you should solve. Store it in `c:\usr`. Open the file using: `fid=fopen('targill.dat', 'rb');` under Solaris OS, or `fid=fopen('targill.dat', 'rb','b');` if it is a WINDOWS OS. This is a binary file, not a text file. Therefore we should specify that we would like to open the file for reading binary information (`rb=read binary`). If `fid > 0` then it is the identifier of the file. If it equals -1, then there is no such file. Check that the `fid` is a valid number.

You may use the FSEEK command to change the pointer location. `status=fseek(fid, 512, 'bof');` takes you 512 bytes forwards from the Beginning Of File (`bof`).

The next step will be reading the binary information into the matrix `A`. We will use the command `A=fread(fid,[5,6],'float');` The specification of `[5,6]` determines the matrix size, and `'float'` describes the 32bit data precision. In order to view the matrix, simply type: `A`.

You should always close a file when you do not need it anymore, using the command: `fclose(fid);`

5.2 A function file that calculates the mean value of a matrix

Write a file `imgmean.m` that contains the following statements:

```
function y=imgmean(x)
% This function calculated the mean value of an input matrix
[m,n] = size(x);
y=sum(sum(x))/(m*n);
```

Why do we use the function SUM twice? The function SUM calculates the sum of each column in a matrix. Check this on `x=[1, 2, 3; 4, 5, 6; 7, 8, 9];` What is `sum(x)`? What is `sum(sum(x))`?

Now, we will try to use this function to calculate the mean value of the matrix we have just loaded.

Type:
`m=imgmean(A);`

6. Control Flow in Matlab

6.1 FOR Loops

A FOR loop allows a statement to be repeated a fixed, predetermined number of times. Let's look at the following problem. We would like to fill the vector *b* with square roots of 1 to 1000. One way to do so, is by using a for loop.(In the next chapter we will learn about more efficient ways to do this). We will calculate the time required for this operation for comparing it with the more efficient version of this calculation.

Write this code in an m-file and save it under the name *tictoc.m*.

```
clear ;           To clear all previous variables, and to free memory.
tic ;           This function initializes an internal clock
for i = 1:1000
            b(i) = sqrt(i);
end
t=toc;
```

The time required was: _____.

The syntax for the FOR loops is:

```
        for v=expression
                statements
        end
```

6.2 WHILE Loops

A while loop allows a statement to be repeated an indefinite number of times, under control of a logical condition. The syntax for the WHILE loops is:

```
        while expression
                statements
        end
```

6.3 IF ... ELSE ...

The syntax for the for the IF ... ELSE ... is:

```
        if expression
                statements
        elseif expression
                statements
        else
                statements
        end
```

Try using the WHILE and the IF statements to calculate all the Fibonacci numbers so that the sum of two consecutive numbers is smaller than 10,000. How many are even? How many are odd? Try to plot them.

Hints:

1. Matlab can increase the size of a vector as it is being created.

2. To determine whether a number n is even or odd you can use the function `rem(n,2)`. If `rem(n,2)` equals 0 then the number is even, otherwise it is odd.

7. How to write effective Matlab code for IP applications

One nice thing about Matlab is that your program will work fine, no matter how inefficient you wrote it. However, we will try to get to know some useful ways to optimize our code. The purpose is not to use complex tricks to avoid a small for loop, rather we will try to match the optimization to matrix manipulation needs.

Matlab uses a great amount of memory for its variables and its functions. In order to write efficient code, you should always try to implement the following guidelines. You should try to make every effort to vectorize your algorithms. Moreover, pre-allocate matrices where the results of your operations are saved. For example: if you need a matrix of size 3×3 , pre-allocate memory by the command: `y=zeros(3)` ; Otherwise, Matlab has to resize the matrix as it is being created. If the matrix size is pre-allocated, execution is much faster.

7.1. How to vectorize your code?

Remember the for loop we have used for creating the square roots vector? We will now show you a better way to do this. Write the following code in an m-file under the name `tictoc2.m`.

```
clear;  
tic;  
a=1:1:1000; So, a is a vector which looks like : 1, 2, 3, .....1000  
b=sqrt(a);  
t=toc;
```

So the time now is _____.

We achieved this efficiency by vectorizing our code. This is a nice thing about Matlab. You can ignore the fact that you deal with two vectors. You write a vectorized equation, but there is still going to be an element by element calculation.

Let's say you are looking for a subset of a matrix that satisfies a certain condition and you want to do something with these elements.

Let's look at the matrix `M=magic(3)`. `MAGIC(N)` is an N -by- N matrix constructed from the integers 1 through N^2 with equal row and column sums.

`M=`

8	1	6
3	5	7
4	9	2

Let's look for all the elements which are greater than 4, and negate them.

Try writing the proper for loop, and execute it. 🔔

Now, we can use the Matlab command FIND.
Look at the following example:

```
ind=find(M>4);
```

What is ind? The FIND command returns the indices of the matrix elements which satisfy the condition $M>4$. All you have to do next is write:

```
M(ind)=-M(ind);
```

Another issue can be the need to do something different to each column of a matrix without a loop. Let's calculate the mean of each column of a matrix, and then remove it from each column.

```
M=rand(10,5)      this is a random matrix 10x5.  
V=mean(M);        calculates the mean value of each column.
```

🔔 Write your code to subtract from each element of M the mean of the appropriate column.

Now, compare your code to the following one:

```
M=M-V(ones(10,1),:);
```

 What does V look like? And $V(\text{ones}(10,1),:)$? This is a useful application of Tony's Trick.

x is a vector. $x=1:5$; When we want to refer to the third and fifth elements of this vector, we can do that in two ways:
 $x([3,5])$ or $x([0 0 1 0 1])$;

Files to submit at the end of the lesson:

```
grp#loading.m  
grp#fibo.m  
grp#maxfun.m  
grp#maxfun2.m
```