Lab 1: Introduction to MATLAB

1. Warm-up

MATLAB is a high-level programming language that has been used extensively to solve complex engineering problems. The language itself bears some similarities with ANSI C and FORTRAN.

MATLAB works with three types of windows on your computer screen. These are the `Command` window, the `Figure` window and the `Editor` window. The `Figure` window only pops up whenever you plot something. The `Editor` window is used for writing and editing MATLAB programs (called M-files) and can be invoked in Windows from the pull-down menu after selecting `File` | `New` | `M-file`. In UNIX, the `Editor` window pops up when you type in the command window: `edit filename` ('filename' is the name of the file you want to create).

The `command` window is the main window in which you communicate with the MATLAB interpreter. The MATLAB interpreter displays a command >> indicating that it is ready to accept commands from you.

- View the MATLAB introduction by typing

  >> `intro`

at the MATLAB prompt. This short introduction will demonstrate some basic MATLAB commands.

- Explore MATLAB's help capability by trying the following:

  >> `help`

  >> `help plot`

  >> `help ops`

  >> `help arith`

- Type `demo` and explore some of the demos of MATLAB commands.

- You can use the `command` window as a calculator, or you can use it to call other MATLAB programs (M-files).

Say you want to evaluate the expression $a^3 + \sqrt{bd} - 4c$, where a=1.2, b=2.3, c=4.5 and d=4. Then in the `command` window, type:

```
>> a = 1.2;
>> b=2.3;
>> c=4.5;
>> d=4;
>> a^3+sqrt(b*d)-4*c
```

```
ans =
       -13.2388
```

Note the semicolon after each variable assignment. If you omit the semicolon, then MATLAB echoes back on the screen the variable value.

2. Arithmetic Operations

There are four different arithmetic operators:

+ addition
− subtraction
* multiplication
/ division (for matrices it also means inversion)

There are also three other operators that operate on an element by element basis:

`.*` multiplication of two vectors, element by element
`./` division of two vectors, element-wise
`.^` raise all the elements of a vector to a power.

Suppose that we have the vectors $\mathsf{x} = [x_1, x_2, ..., x_n]$ and $\mathsf{y} = [y_1, y_2, ..., y_n]$. Then

$\mathsf{x}.*\mathsf{y} = [x_1y_1, x_2y_2, ..., x_ny_n]$
$\mathsf{x}./\mathsf{y} = [\frac{x_1}{y_1}, \frac{x_2}{y_2}, ..., \frac{x_n}{y_n}]$
$\mathsf{x}.\hat{}\,\mathsf{p} = [x_1^p, x_2^p, ..., x_n^p]$

The arithmetic operators + and – can be used to add or subtract matrices, scalars or vectors. By vectors we mean one-dimensional arrays and by matrices we mean multi-dimensional arrays. This terminology of vectors and matrices comes from Linear Algebra.

Example:

```
>> X=[1,3,4]
>> Y=[4,5,6]
>> X+Y
ans=
    5 8 10
```

For the vectors `X` and `Y` the operator + adds the elements of the vectors, one by one, assuming that the two vectors have the same dimension. In the above example, both vectors had the dimension 1×3, i.e., one row with three columns. An error will occur if you try to add a 1×3 vector to a 3×1 vector. The same applies for matrices.

To compute the **dot product** of two vectors (i.e. $\sum_i x_i y_i$), you can use the multiplication operator * . For the above example, it is:

```
>> X*Y'
ans =
      43
```

Note the single quote after Y. The single quote denotes the transpose of a matrix or a vector.

To compute an element by element multiplication of two vectors (or two arrays), you can use the .* operator:

```
>> X .* Y
ans =
     4 15 24
```

That is, `X.*Y` means $[1\times4, 3\times5, 4\times6] = [4\ 15\ 24]$. The '.*' operator is used very often (and is highly recommended) because it is executed much faster compared to the code that uses `for` loops.

3. Complex numbers

MATLAB also supports complex numbers. The imaginary number is denoted with the symbol `i` or `j`, assuming that you did not use these symbols anywhere in your program (that is very important!). Try the following:

```
>> z=3 + 4i % note that you do not need the '*' after 4
>> conj(z) % computes the conjugate of z
>> angle(z) % computes the phase of z
>> real(z) % computes the real part of z
>> imag(z) % computes the imaginary part of z
>> abs(z) % computes the magnitude of z
```

You can also define the imaginary number with any other variables you like. Try the following:

```
>> img=sqrt(-1)
>> z=3+4*img
>> exp(pi*img)
```

4. Array indexing

In MATLAB, all arrays (vectors) are indexed starting with 1, i.e., y(1) is the first element of the array y. Note that the arrays are indexed using parenthesis (.) and not square brackets [.] as in C/C++. To create an array having as elements the integers 1 through 6, just enter:

```
>> x=[1,2,3,4,5,6]
```

Alternatively, you can use the `:` notation,

```
>> x=1:6
```

The `:` notation above creates a vector starting from 1 to 6, in steps of 1. If you want to create a vector from 1 to 6 in steps of say 2, then type:

```
>> x=1:2:6
Ans =
     1 3 5
```

Try the following code:

```
>> ii=2:4:17
>> jj=20:-2:0
```

```
>> ii=2:(1/10):4
```

Extracting or inserting numbers in a vector can be done very easily. To concatenate an array, you can use the `[ ]` operator, as shown in the example below:

```
>> x=[1:3   4 6   100:110]
```

To access a subset of the array, try the following:

```
>> x(3:7)
>> length(x) % gives the size of the array or vector
>> x(2:2:length(x))
```

5. Allocating memory

You can allocate memory for one-dimensional arrays (vectors) using the `zeros` command. The following command allocates memory for a 100-dimensional array:

```
>> Y=zeros(100,1);
>> Y(30)
ans =
       0
```

Similarly, you can allocate memory for two-dimensional arrays (matrices). The command

```
>> Y=zeros(4,5)
```

defines a 4 by 5 matrix. Similar to the `zeros` command, you can use the command `ones` to define a vector containing all ones,

```
>> Y=ones(1,5)
ans=
    1 1 1 1 1
```

6. Special characters and functions

Some common special characters used in MATLAB are given below:

Symbol	Meaning
pi	π(3.14...)
sqrt	indicates square root e.g., sqrt(4)=2
^	indicates power(e.g., 3^2=9)
abs	Absolute value | .| e.g., abs(-3)=3
NaN	Not-a-number, obtained when comparing mathematically undefined operations, such as 0/0
Inf	Represents $+\infty$
;	Indicates the end of a row in a matrix. It is also used to suppress printing on the screen (echo off)
%	Denotes a comment. Anything to the right of % is ignored by the MATLAB interpreter and is considered as comments
'	Denotes transpose of a vector or matrix. It's also used to define strings, e.g.,str1='DSP';

Some special functions are given below:

`length(x)` - gives the dimension of the array x

`find` - Finds indices of nonzero elements.

Examples :

```
>> x=1:10;
>> length(x)
ans =
       10
```

The function `find` returns the indices of the vector X that are non-zero. For example, I = `find(X>100)`, finds all the indices of X when X is greater than 100. So for the above example:

```
>> find(x> 4)
ans =
     5 6 7 8 9 10
```

7. Control flow

MATLAB has the following flow control constructs:

- `if` statements
- `switch` statements
- `for` loops
- `while` loops
- `break` statements

The if, for, switch and while statements need to terminate with an end statement. Examples:

IF:

```
x=-3;
if x>0
    str='positive';
elseif x<0
    str='negative';
elseif x= = 0
    str='zero';
else
    str='error';
end
```

What is the value of 'str' after execution of the above code?

WHILE:

```
x=-10;
while x<0
    x=x+1;
end
```

What is the value of x after execution of the above loop?

FOR loop:

```
X=0;
for i=1:10
    X=X+1;
end
```

The above code computes the sum of all numbers from 1 to 10.

BREAK:

The break statement lets you exit early from a for or a while loop:

```
x=-10;
while x<0
    x=x+2;
    if x = = -2
        break;
    end
end
```

MATLAB supports the following relational and logical operators:

Relational Operators

Symbol	Meaning
<=	Less than equal
<	Less than
>=	Greater than equal
>	Greater than
==	Equal
~=	Not equal

Logical Operators

Symbol	Meaning
&	AND
|	OR
~	NOT

8. Plotting

You can plot arrays using MATLAB's function `plot.` The function `plot` (.) is used to generate line plots. The function `stem(`.) is used to generate "picket-fence" type of plots.

Example:

```
>> x=1:20;
>> plot(x)  %see Figure 1
>> stem(x)  % see Figure 2
```

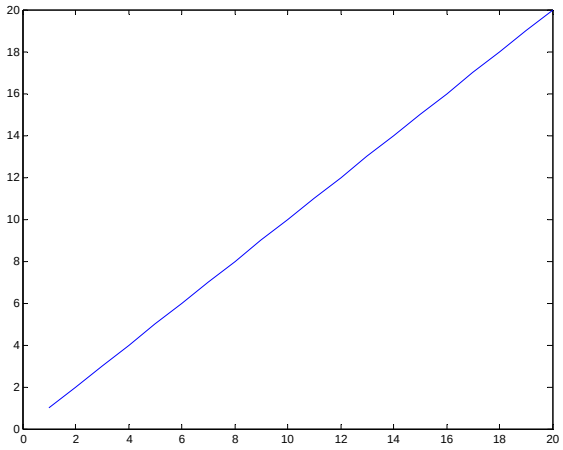

Figure 1: Plot obtained using the `plot` command.

Example of a plot generated using the `plot` command is shown in Figure 1, and example of a plot generated using the `stem` function is shown in Figure 2. More generally, `plot(X,Y)` plots vector Y versus vector X. Various line types, plot symbols and colors may be obtained

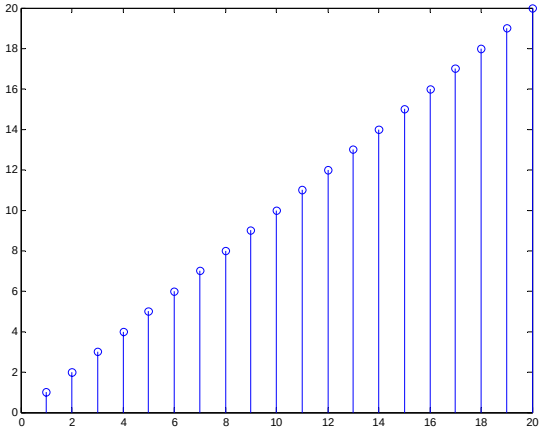

Figure 2: Plot obtained using the `stem` function.

using `plot(X,Y,S)` where S is a character string indicating the color of the line, and the type of line (e.g., dashed, solid, dotted, etc.). Examples for the string S include:

r	red	+	plus	−−	dashed
g	green	*	star		
b	blue	s	square		

You can insert x-labels, y-labels and title to the plots, using the functions `xlabel(.)`, `ylabel(.)` and `title(.)` respectively. To plot two or more graphs on the same figure, use the command `subplot`. For instance, to show the above two plots in the same figure, type:

```
>> subplot(2,1,1), plot(x)
>> subplot(2,1,2), stem(x)
```

The (m,n,p) argument in the subplot command indicates that the figure will be split in m rows and n columns. The 'p' argument takes the values 1, 2, ..., $m \times n$. In the example above, $m = 2, n = 1$,and, $p = 1$ for the top figure and $p = 2$ for the bottom figure.

To get more help on plotting, type: `help plot` or `help subplot`.

9. Programming in MATLAB (M-files)

MATLAB programming is done using M-files, i.e., files that have the extension .m. These files are created using a text editor. To open the text editor, go to the `File` pull-down menu, choose `New,` then `M-file`. After you type in the program, save it, and then call it from the command window to execute it.

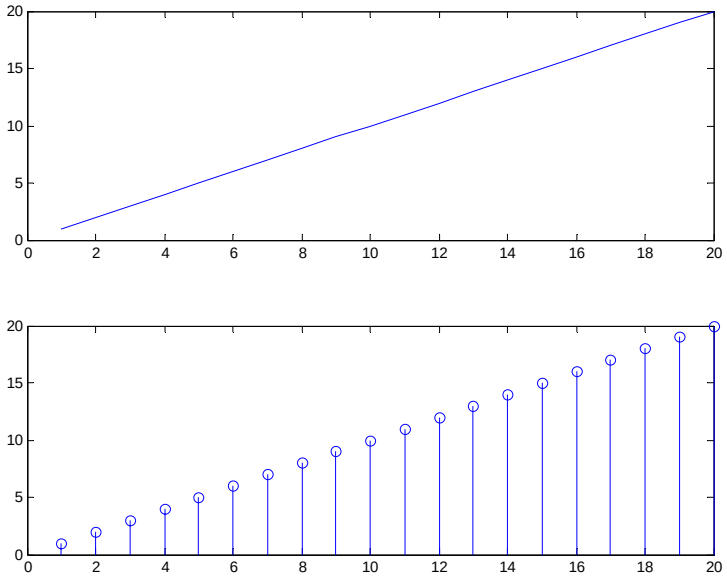

Figure 3: Output of the `subplot(2,1,p)` command.

Say for instance that you want to write a program to compute the average (mean) of a vector `x`. The program should take as input the vector `x` and return the average of the vector.

Steps:

1. You need to create a new file, called "average.m". If you are in Windows, open the text editor by going to the `File` pull-down menu, choose `New`, then `M-file`. If you are in UNIX, then type in the command window: `edit average.m`. Type the following in the empty file

```
function y=average(x)
L=length(x);
sum=0;
for i=1:L
    sum=sum+x(i);
end
y=sum/L; % the average of x
```

Remarks:

y – is the output of the function "average"
x – is the input array to the function "average"

average – is the name of the function. It's best if it has the same name as the filename. MATLAB files always need to have the extension .m

2. From the Editor pull-down menu, go to `File | Save`, and enter: `average.m` for the filename.

3. Go to the Command window to execute the program by typing:

```
>> x=1:100;
>> y=average(x)
ans =
       50.5000
```

Note that the function `average` takes as input the array `x` and returns one number, the average of the array. In general, you can pass more than one input argument and can have the function return multiple values. You can declare the function `average`, for instance, to return 3 variables while taking 4 variables as input with the following statement:

```
function [y1, y2, y3]=average(x1,x2,x3,x4)
```

In the command window it has to be invoked as:

```
>> [y1, y2, y3]=average(x1,x2,x3,x4)
```

10. MATLAB sound

If your PC has a sound card, then you can use the function `soundsc` to play back speech or audio files through the PC's speakers. The usage of this function is given below:

`soundsc(Y,FS)` sends the signal in vector `Y` (with sample frequency FS) out to the speaker on platforms that support sound. Stereo sounds are played, on platforms that support it, when `Y` is an N-by-2 matrix.

Try the following code, and listen to a 400-Hz tone:

```
>> t=0:1/8192:1;
>> x=cos(2*pi*400*t);
>> soundsc(x,8192);
```

Now, try listening to noise:

```
>> noise=randn(8192,1);  % generate 8192 samples of noise
>> soundsc(noise,8192);
```

The function `randn` generates Gaussian noise.

11. Loading and saving data

You can load or save data using the commands `load` and `save`. To save the variable `x` of the above code in the file `data.mat`, type:

```
>> save data.mat x
```

Note that MATLAB's data files have the extension `.mat`. To retrieve the data that was saved in the vector `x`, type:

```
>> load data.mat
```

The vector x is loaded in memory. To see the contents of memory use the command `whos`:

```
>> whos

Name     Size     Bytes    Class
x        1x8193   65544    double array
Grand total is 8193 elements using 65544 bytes
```

The command `whos` gives a list of all the variables currently in memory, along with their dimension. In our case, `x` contained 8193 samples.

To clear up memory after loading a file, you may type `clear all` when done. That is very important, because if you do not clear all the variables in memory, you may run into problems with other programs that you will write that use the same variables.

12. Problems

1. Write a MATLAB program that will add all the numbers corresponding to the even indices of an array. For instance, if the array x was `x=[1, 3, 5, 10]`, then it should return 13 ($= 3 + 10$). Use that program to find the sum of all even integers from 1 to 1000. Write your program so that it is flexible. That is, you should be able to invoke your program from the command window as follows:

   ```
   >> y=addeven(x)
   ```

   where `x` is the input vector, and `y` is the sum of all the numbers corresponding to the even indices of `x`.

2. Can you explain what the following program does:

   ```
   L=length(x);
   for i=1:L
       if x(i) < 0
           x(i)=-1;
       end
   end
   ```

   Can you rewrite this program without using `for` loops?

3. Write a program to compute the variance of an array `x`. The variance σ is defined to be:

$$\sigma = \frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2 \tag{1}$$

where $\overline{x}$ is the average of the array `x`. For `x,` use all the integers from 1 to 1000.

4. Write a program that implements the following hard-limiting function:

$$f(x) = \begin{cases} 0.2 & x \geq 0.2 \\ -0.2 & x \leq 0.2 \end{cases} \tag{2}$$

For `x,` use 1000 random numbers generated using the function `rand`.

Lab 1
Instructor Verification Sheet

Staple this page to the end of your lab report

Name: _____ Date: _____

Section 7. Run the while loop code.

Instructor verification _____

Section 8. Run the subplot command.

Instructor verification _____

Section 9. Run the average function.

Instructor verification _____

Section 10. Run the sound command.

Instructor verification _____