MatLab/Octave Quick Reference and Tutorial

1 Matlab installation

In the class, the Matlab package will be used to solve laboratory exercises.

Matlab is available to students for installation in own laptop, you can find for download and installation at `https://www.tcd.ie/itservices/assets/doc/software/Matlab-UGs-PGs-Personal-PCs.pdf`

2 Basic operations

2.1 Execution modes

You can execute commands in Matlab in two modes:

- **prompt mode** - first you need to start Matlab, then you can type command and execute it by pressing Enter. After command is executed an immediate result appears, example:

```
>> 8*8
ans =  64
>> 3-1
ans =  2
>>
```

- **script mode** - on the top left of the window click on New $->$ Script

```
8*8
3-1
```

Push the green play button called Run; Matlab will prompt a file save if it is the first time the file is created. You should get following output:

```
ans =  64
ans =  2
```

2.2 Adding comments

To comment and describe your work you may use comments in source codes. To start a comment use '%' symbol and type comment after it, example *file_mode_with_comments.m*:

```
   % How to comment in Matlab
8*8 % multiplication
3-1 % subtraction
```

2.3 Arithmetic

After executing in file mode the *arithmetics.m*:

```
% basic arithmetic
5+3 % addition
6-2 % subtraction
5*8 % multiplication
1/3 % division
2^3 % power
sqrt(16) % square root
```

, you should get:

```
ans =  8
ans =  4
ans =  40
ans =  0.33333
ans =  8
ans =  4
```

2.4 Assigning values

After executing in **file mode** the *assign.m*:

```
% Assigning values to variables
a=5 %  assign a value to a variable
a=5; % the semicolon suppresses the output
a='Hello' % assign a string to a variable. Notice, that variables are not
  declared and therefore can be any type.
a=true  % assign the boolean value True to 'a'
a=pi % assign the value of Pi to 'a'
```

, you should get:

```
a =  5
a = Hello
a =  1
a =  3.1416
```

2.5 Displaying values

To display a variable in **prompt mode** just type the name of variable and press Enter. Moreover, especially in **file mode**, the command *disp()* function can be used.

After executing in **file mode** the *display.m*:

```
% Displaying values
a=pi
a
disp(a)
disp(sprintf('2 decimals: %0.2f', a)) % works in analogous way as the C
  language sprintf function, displays 'a' as float number to 2 decimal
  places
```

, you should get:

```
a =  3.1416
a =  3.1416
 3.1416
2 decimals: 3.14
```

2.6 Comparing values

After executing in **file mode** the *compare.m*:

```
% Comparing values
1==2    % is 1 equal to 2?
1~=2   % is 1 not equal to 2?
a=(3>=1) % assign to 'a' the boolean value resulting from evaluating the
  expression '3>=1'
```

, you should get:

```
ans = 0
ans =  1
a =  1
```

3 Vectors and Matrices

This section describe basic operations on vectors and matrices.

Matlab supports vectors and matrices very well. Its libraries are built-in and therefore make operations with vectors and matrices very efficient. After executing in **file mode** the *matrices_basic.m*:

```
% Basic operations on vectors and matrices
a=[1 2 3 4 5]  % assign to variable 'a', a ROW vector with elements
  1,2,3,4,5
b=[1,2,3,4,5] % does exactly the same as above
c=[1;2;3;4;5] % assign to 'c', a COLUMN vector with elements 1,2,3,4,5

TransposeMatrix=c'  % transpose operator on matrix c

MatrixStep1=0:1:10 % creates a vector with values from 0 to 10 with step
  1, i.e.: 0,1,2,3,4,5,6,7,8,9,10
MatrixStep01=0:0.1:1 % analogous as above but the elements are from 0 to 1
   with step 0.1, i.e.: 0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0

MultiDimMatrix=[1 2 3;4 5 6;7 8 9;10 11 12] % creates a 4x3 matrix one row
   after the other

size(MultiDimMatrix) % gives the size of matrix in number of rows and
  number of columns
size(MultiDimMatrix,1) % gives the number of rows of matrix
size(MultiDimMatrix,2) % gives the number of columns of matrix

disp(MultiDimMatrix) % shows the elements of matrix organized in rows and
  columns. Or you can just type the variable name here.

MultiDimMatrix(1,3) % shows the element in row 1 and column 3
MultiDimMatrix(2,:) % shows all elements of row 2
MultiDimMatrix(:,3) % shows all elements of column 3

OneMatrix=ones(2,3) % creates a matrix of all 1 of size 2x3
ZeroMatrix=zeros(5,2) % creates a matrix of 0 of size 5x2

MatrixByConstant=3*ones(2,3) % a simple way for creating a matrix with all
   values equal to 3, of size 2x3

IdentityMatrix=eye(3) % generates a 3x3 identity matrix, i.e. all elements
   are 0 except this in the diagonal, which are 1
```

,you should get:

```
a =   1   2   3   4   5
b =   1   2   3   4   5
c =
   1
   2
   3
   4
   5
TransposeMatrix =   1   2   3   4   5
MatrixStep1 =    0    1    2    3    4    5    6    7    8    9   10
MatrixStep01 =  0.00000    0.10000    0.20000    0.30000    0.40000
  0.50000    0.60000    0.70000    0.80000    0.90000    1.00000
MultiDimMatrix =
    1    2    3
    4    5    6
    7    8    9
   10   11   12
ans =   4   3
ans =  4
ans =  3
    1    2    3
```

```
    4    5    6
    7    8    9
   10   11   12
ans =  3
ans =   4   5   6
ans =
    3
    6
    9
   12
OneMatrix =
   1   1   1
   1   1   1
ZeroMatrix =
   0   0
   0   0
   0   0
   0   0
   0   0
MatrixByConstant =
   3   3   3
   3   3   3
IdentityMatrix =
Diagonal Matrix
   1   0   0
   0   1   0
   0   0   1
```

4 Advanced Operations on Vectors and Matrices

This section describe advanced operations on vectors and matrices.

Notice, that all operators that work on scalars (objects that are not vectors or matrices) can also be applied to vectors and matrices.
After executing in **file mode** the *matrices_advanced.m*:

```
   % Advanced Operations on vectors and matrices
A=[1 3 5;2 4 6] % 2x3
A2=[6 8 9;4 2 1] % 2x3
B=[1 2; 3 4; 5 6] % 3x2
C=[1 2 3;4 5 6;7 8 9] % 3x3
A+A2 % sums the corresponding elements of A and A2. Only works if A and A2
   have the same dimensions.
A-A2 % same as above but operates the subtraction
A*B % operates the row-by-column multiplication between matrices. A and B
  need to be of suitable sizes (sizes n x m and m x k, for any value of n,
  m and k).

comment='Element-wise multiplication (A .* B) result:'
A.*A2 % operates the element-wise multiplication of the elements of A and
  A2, i.e. it multiplies the corresponding elements of A and A2 (similar
  to the + and - operators). A and A2 needs to be of the same size.

C^2 % operates the same as C*C. Notice that C must be square (because C
  needs to be n x m, but also m x k, so n=m, and m=k => n=m=k).

C.^2 % operates the square of each element of matrix C

comment='Division (A/B) result:'
A/A2 % operates the matrix division between A and B

A./A2 % operates the element-wise division between elements of A and B
% The "." before the operator transform the command into an element-wise
  operations.

comment='Logarithms results:'
```

```
log(A) % operates the element-wise natural logarithm of the elements of
  matrix A
log2(A) % same but with base-2 logarithm
log10(A) % same but with base-10 logarithm

exp(A) % operates the element-wise exponential, i.e. e^n, where n are
  elements of A
abs(A) % gets the absolute value of each element of A

comment='Row/column sums results:'
sum(A) % operates the sum of the elements of each COLUMN of A
sum(A,1) % does exactly the same
sum(A,2) % operates the sum of the elements of each ROW of A
sum(sum(A)) % sums up all the elements of matrix A

comment='Max value results:'
max(A) % gives the maximum value for each \emph{column} of A
max(A,[],2) % gives the maximum value for each \emph{row} of A
max(max(A)) % gives the largest element of matrix A
```

, you should get:

```
A =
   1   3   5
   2   4   6
A2 =
   6   8   9
   4   2   1
B =
   1   2
   3   4
   5   6
C =
   1   2   3
   4   5   6
   7   8   9
ans =
    7   11   14
    6    6    7
ans =
  -5  -5  -4
  -2   2   5
ans =
   35   44
   44   56
comment = Element-wise multiplication (A .* B) result:
ans =
    6   24   45
    8    8    6
ans =
    30    36    42
    66    81    96
   102   126   150
ans =
    1    4    9
   16   25   36
   49   64   81
comment = Divison (A/B) result:
ans =
   0.60000  -0.68571
   0.70000  -0.58571
ans =
   0.16667   0.37500   0.55556
   0.50000   2.00000   6.00000
comment = Logarithms results:
ans =
   0.00000   1.09861   1.60944
   0.69315   1.38629   1.79176
```

```
ans =
   0.00000   1.58496   2.32193
   1.00000   2.00000   2.58496
ans =
   0.00000   0.47712   0.69897
   0.30103   0.60206   0.77815
ans =
    2.7183    20.0855   148.4132
    7.3891    54.5982   403.4288
ans =
   1   3   5
   2   4   6
comment = Row/column sums results:
ans =
    3    7   11
ans =
    3    7   11
ans =
    9
   12
ans =  21
comment = Max value results:
ans =
   2   4   6
ans =
   5
   6
ans =  6
```

5 Flow Control - Loops and If statement

This section describes how to control flow of your program with loops and If statement. After executing in **file mode** the *loops_if.m*:

```
% Loops and If statement
% 'for' loop:
comment='Loop:'
for i=1:10
    disp(i); % the indentation is not required but is useful for clarity
end;

comment='While:'
% 'while' loop:
i=1;
while i<=5,
    disp(i);
    i=i+1;
end;

comment='Break:'
% the "break" command escape from a loop:
i=1;
while true,
    disp(i);
    if i==6,
        break;
    end;
    i=i+1;
end;
disp(i);

comment='If statement:'
% Below example shows syntax for If statement:
v=2;
if v==1,
```

```
    disp('The value is one!');
elseif v==2,
    disp('The value is two!');
else
    disp('The values is not one and two!');
end;
```

, you should get:

```
comment = Loop:
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
comment = While:
 1
 2
 3
 4
 5
comment = Break:
 1
 2
 3
 4
 5
 6
 6
comment = If statement:
The value is two!
```

6 Basic statistical functions

Basic statistical functions are presented in this section. After executing in **file mode** the *statistical_basic.m*:

```
   % Basic statistical functions

c=rand(1,5) % assigns to 'c' a matrix sized 1x5 (i.e. a column vector with
   5 elements) of random values. The random values are uniformly
  distributed between 0 and 1.

d=randn(1,5) % assigns to 'd' a vector of normally (i.e. Gaussian)
  distributed random values. The Gaussian has mean 0 and variance 1

e=3+sqrt(10)*randn(1,5) % assigns to 'e' a vector of Gaussian distributed
  random values, with mean 3 and variance 10

dist=4+sqrt(10)*randn(1,10000); % same as above but with 10,000 elements

hist(dist); % plots an histogram of the vector 'c'. Here we can clearly
  see the "bell" shape of the Gaussian distributions

hist(dist,50); % increases the number of bins to 50, to get a finer view
  of the histogram
```

, you should get:

```
c =
```

```
    0.70517    0.75996    0.75788    0.16708    0.51960
d =
    0.68728   -0.73611   -1.27099   -0.73941    0.37123
e =
    10.44354    -1.00649     4.47108     6.16741    -0.15365
```

, and also two plots as in Fig. 1 and 2.

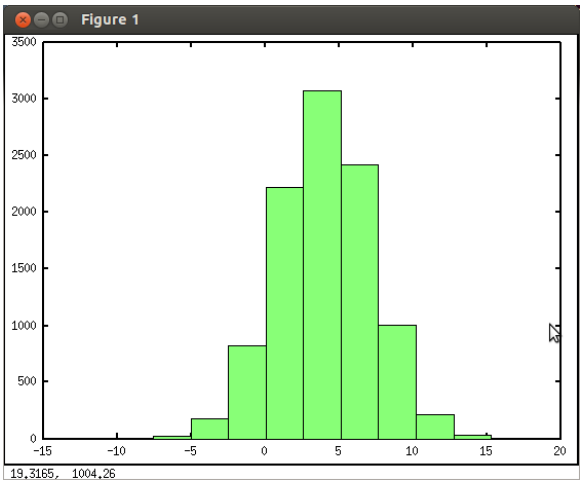

Figure 1: Hist() example 1

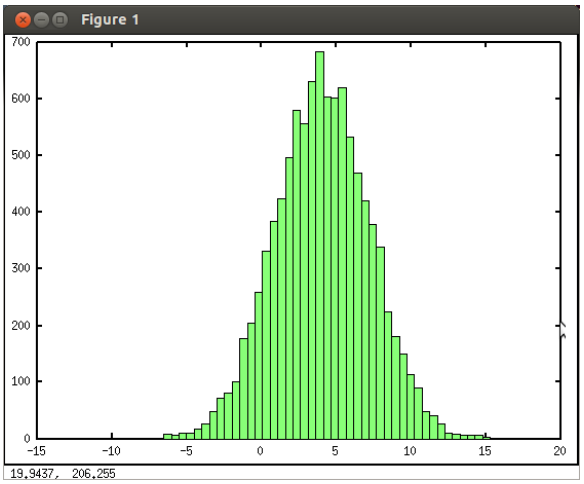

Figure 2: Hist() example 2

7 Creating functions in files

On the top left side of the window click New $->$ Function

```
function [ output_args ] = untitled4( input_args )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here


end
```

Now let's write a function called addNumbers and save it with the same name.

```
function [y1,y2] = addNumbers( a,b )
%addNumbers this function adds two numbers
y1=a+b;
y2=a-b;


end
```

You can call the function, for example in **prompt mode**, by typing: ¿¿addNumbers (5,-6)

```
>> addNumbers(5,-6)
ans = -1
```

Notice that function must be called the same as the file.

You can return multiple values from function as well. For example, create a file "addAndSubtractNumbers.m":

```
function [y1,y2] = addAndSubtractNumbers(a,b)
    y1=a+b;
    y2=a-b;
```

You can use above function in **prompt mode** by typing:

```
>> [y1,y2]= addAndSubtractNumbers(-4,2)
y1 = -2
y2 = -6
```

8 Plotting functions

Matlab shows plots by displaying points and connecting them with lines.

Example of plotting a square function in *square_function_plot.m*:

```
x=-5:0.1:5;
y=x.^2;
plot(x,y);
```

, you should get plot as in Fig. 3.

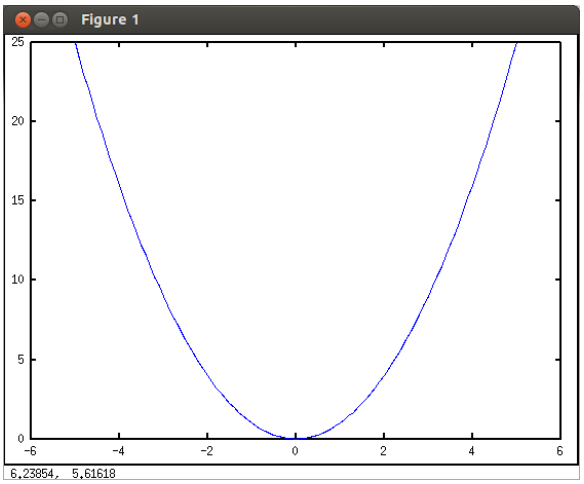

Figure 3: Square function example.

Notice that if we select the step too small the plot will look like a connection of straight lines rather than a curve in *square_function_plot_small_step.m*:

```
x=-5:1:5;
y=x.^2;
plot(x,y);
```

, you should get plot as in Fig. 4.

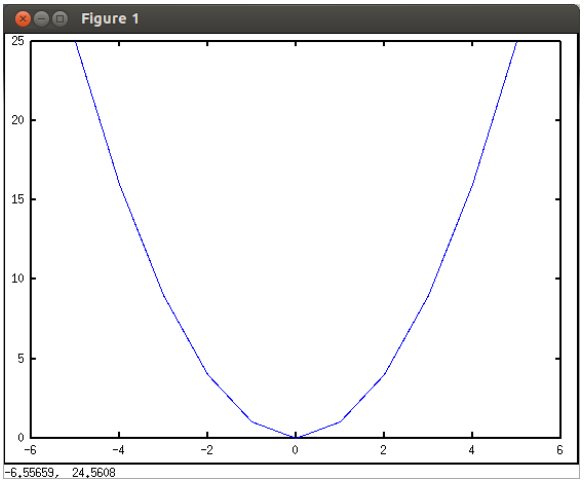

Figure 4: Square function example with small step.

Another useful plot can be generated by *stem()* command, which plots a vertical line for each sample and circle (at top of this line) in *stem_plot.m*:

```
x=0:0.1:8*pi;
y=sin(x);
stem(x,y);
```

, you should get plot as in Fig. 5.

Complete example of *sin()* and *cos()* functions plotting in *sin_and_cos_functions_plot.m*:

```
% Example of sin() and cos() plotting on the same figure
hold on; % draw successive plots together on the same figure (do not
  remove/overwritte previous plot)

x=0:0.1:8*pi;
y=sin(x); % sine function
y2=cos(x); % cosine function

plot(x,y); % plot sine function
plot(x,y2,'r'); % the command 'r' tells to plot in red color

% add labels, legend and title
xlabel('time'); % add the x axis label to the figure
ylabel('amplitude'); % add the y axis label
legend('sin', 'cos'); % add a legend to the plot
title('Sine and Cosine functions.'); % add a title to the plot
print -dpng 'sin_cos_functions.png' % finally save the plot
```

, you should get plot as in Fig. 6.

8.1 Multiple plots on the same figure

You can show multiple plots on the same figure.

Multiple figures example in *multiple_figures_and_plots.m*:

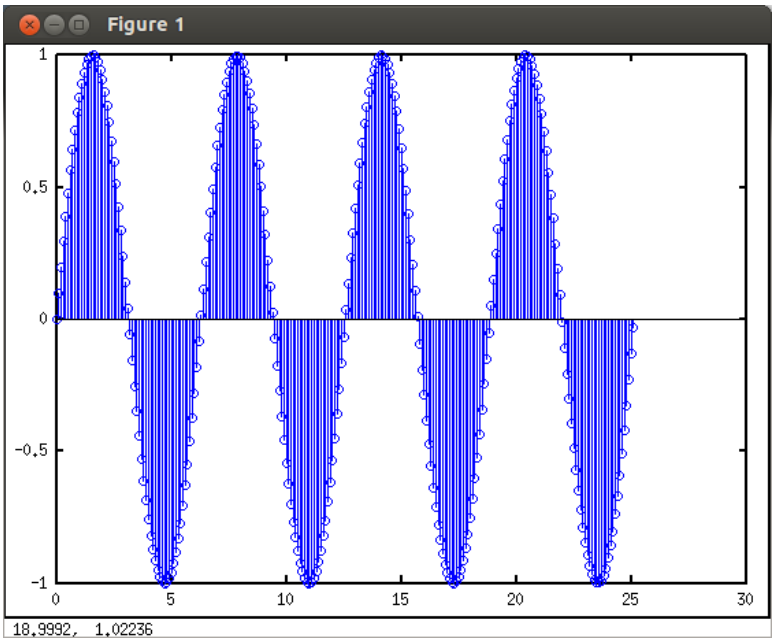

Figure 5: Stem() function example.

```
% Multiple figures and plots
x=0:0.1:8*pi;
y=sin(x); % sine function
y2=cos(x); % cosine function

figure(1); % create figure '1'
figure(2); % create figure '2'

figure(1) % go to figure '1'
plot(x,y); % plot sine on figure 1

figure(2) % go to figure '2'

% Multiple plots:
subplot(2,3,[4]); % creates a 2x3 (row x column) grid and set the 4th cell
    as entire subplot drawing area
% the gird 2x3 cells are indexed as follows:
% +-----+-----+-----+
% |  1  |  2  |  3  |
% +-----+-----+-----+
% |  4  |  5  |  6  |
% +-----+-----+-----+
plot(x,y); % plot sine in the '4'th cell

subplot(2,3,[3,6]); % creates second subplot and set the 3rd and 6th cells
    as drawing area of subplot
plot(x,y2,'r'); % plot cosine in subplot on figure '2'

axis([0.5 10 -1 1]); % resize the axis of the current subplot
```

, you should get two figures: one with sine and second as in Fig. 7.

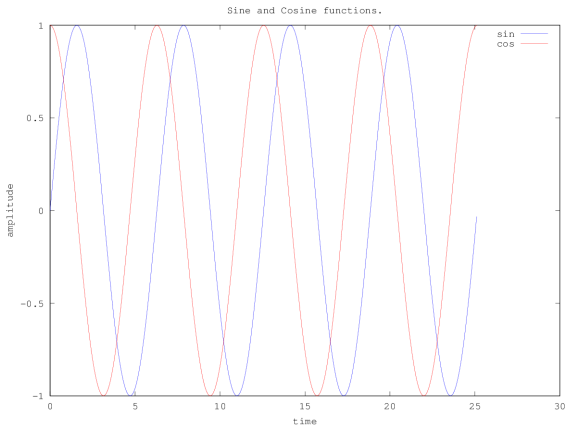

Figure 6: Sin() and cos() functions example.

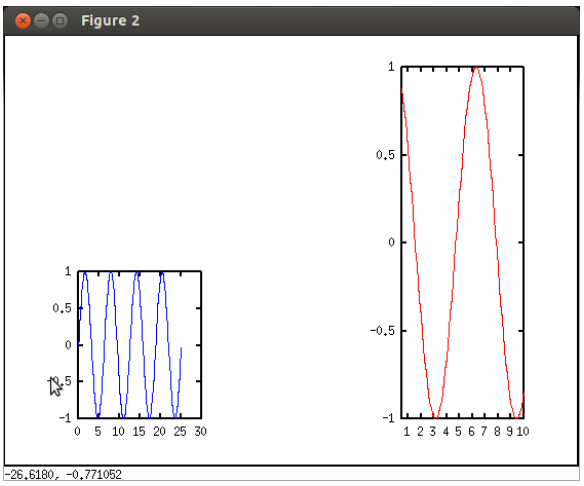

Figure 7: Subplots example.