1 Introduction to MATLAB

This is an introduction to the MATLAB language for the course in *Machine Learning*. After a brief introduction of the basic operations on vector, matrix and cells, we will show some of the operators offered to program in the MATLAB language and some plot options.

1.1 Basics

To start a script in MATLAB we should, at first:

```
clear           %clear the workspace
clc             %clear the command window
close all       %close all the previously opened figures
```

If you are not sure of the usage of a function (but you know its name) you use:

```
help function_name
```

otherwise you can ask to *Matlab Central*.[1]

MATLAB does not require to initialize variables:

```
a = 15;
b = 5;
```

If you want to print the value of a variable on the screen you should not use the semi-column at the end of the line:

```
c = a + b
```

Any result coming from an operation in MATLAB which is not assigned to any variable is associated by default to the variable ans. The command whos returns the names of all the variables present in the current workspace as well as their properties. Any scalar

[1] http://it.mathworks.com/matlabcentral/?refresh=true.

number in MATLAB is considered a 1×1 matrix; any vector is considered as an $n \times 1$ (column) or $1 \times n$ (row) matrix.

There are reserved variables names, for instance `i` is the imaginary unit:

```
i
j
pi
eps %machine epsilon
Inf %infinity
NaN %not a number
```

If you want to check if you are using a reserved word in MATLAB you should check it with `iskeyword()`.

1.2 Conditional and loop operators

Each operators begins with a keyword and ends with `end`. Here we report the syntax for the most common ones

1.2.1 if

```
if %condition
        %instruction 1
        %instruction 2
        %...
elseif
        %instruction 3
        %instruction 4
        %...
else
        %instruction 5
        %instruction 6
        %...
end
```

Remark 1. *The condition should return a* single *logical value.*

1.2.2 while

```
while %condizione
        %instruction 1
        %instruction 2
        %...
end
```

There is no `do ... while` operator.

1.2.3 for

```
for ii = %vector
        %instruction 1
        %instruction 2
end
```

It iterates over all the elements of the `vector`. If we provide a matrix to the iterator `ii`, it will be evaluated column by column.

Remark 2. *In MATLAB , it is a good choice not to change the value of the index `ii` used for the loop inside a for loop itself.*

1.2.4 switch

```
switch %variable
        case %value1
                %instruction 1
                %instruction 2
        case %value1
                %instruction 3
                %instruction 4
        otherwise
                %instruction 5
                %instruction 6
end
```

It is possible to use `variables` of integer and string type for the `switch` clause. Each set of instructions in a `case` is mutually exclusive with the other ones (i.e., no `break` is required).

1.3 Vectors and Matrix

In MATLAB , we use the square brackets (concatenation operation) to define vectors, dividing numbers with spaces (or equivalently with commas), if we want to concatenate them orizzontally (row vector) and by semicolumns, if we want to concatenate them vertically (column vector):

```
row_vec = [10, 11, 12, 13, 14];
column_vec = [10; 11; 12; 13 ; 14]
```

Indexing in MATLAB is performed with round brackets:

```
ii = 4;
row_vec(ii)
idx = logical([0 1 0 1 1]);
row_vec(idx)
```

Remark 3. *Round brackets are used also for functions, thus:*

```
foo(12);
```

is either accessing the 12*-th element of the vector* `foo`*, if you defined it as a vector and it calls the function* `foo` *with input* 12 *if you defined it or if it exists a built in function with the same name. Common mistake is the use of the function* `sigma`*.*

Remark 4. *Indexing in* MATLAB *starts from* 1*.*

Vectors dimension in MATLAB is dynamic. The reserved word `end`, when is used to index a vector, returns the last index of the vector. It is possible to use the concatenation operation (square brackets) to merge vectors:

```
row_vec = [row_vec, row_vec]
row_vec(end+1) = 5,
```

We can not access elements exceeding vector dimension, but we can assign them values (and fill the empty positions of the vector with zeros):

```
row_vec = [4 7 2];
row_vec(10) %Error: index out of matrix dimensions
row_vec(10) = 8;
```

In order to declare a column vector we can either transpose a row vector (with a single quotation mark) or use the vertical concatenation operator `[... ; ...]`:

```
column_vec = row_vec';
column_vec = [4; 5; 6];
whos
```

It is possible to define evenly spaced vectors by specifying starting point, step and ending point:

```
val_ini = 9;
step = 15;
val_end = 223;

v = [val_ini : step : val_end]
%oppure
v = val_ini : step : val_end
v = val_ini : val_end  %(default step 1)
```

or similarly one can use the function `linspace`;

```
val_ini = 9;
n_points = 20;
val_end = 223;

linspace(val_ini,val_end,n_points);
```

It is possible to select subset of elements from vectors by using integer indexes (specific elements or slices) or with with logical indexing:

```
row_vec = 5:15;
indexes = [1, 8, 3];
sub_vec1 = row_vec(indexes)
indexes = 1:4;
sub_vec2 = row_vec(indexes)
log_indexes = logical([1 0 0 0 1 0 0 1 0 0 0 0];
sub_vec3 = row_vec(log_indexes);
```

If we use logical indexes we are required it to be of the same length of the original vector.

A matrix is the proper sequence of horizontal and vertical concatenation operations:

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

It is possible either to index the matrix elements with row and column index or by a single index, as if it was a vector (column by column):

```
A(1,2)
A(3)
```

The use of the colon is used to specify all the existing indexes, thus in matrix is used to access it as if it was a vector:

```
aa = A(:)
```

There are some built in function to define a matrix with particular structure:

```
A = zeros(5); %5x5 matrix of zeros
A = zeros(3,4); %3x4 matrix of zeros
A = ones(5); %5x5 matrix of ones
A = ones(3,4); %3x4 matrix of ones
A = eye(5); %identity matrix of order 5
A = rand(4); %4x4 matrix with uniform random numbers in [0,1]
```

Similarly to vectors, we can select slices of a matrix or replace them by indexing:

```
a = A(:,3) %selects third column
b = A(2,:) %selects second row
A(:, 3) = 2 %replaces third column with the value 2
A(3:end,3:end) = 1 %replaces the right-lower part of the matrix
    with ones
A(:,3) = [1:5] %replaces the third column of the matrix with [1
    2 3 4 5]
               % (if the matrix has 5 rows)
```

Matrix (and vector) summation and subtraction are performed element by element, i.e., only if the matrix have the same dimensions. The same operators performed between a matrix and a scalar value will replicate the scalar to fit the matrix dimensions:

```
a = [1 2 3]
b = [1 2 4]
c = a + b
d = c + 3
```

There are two different kind of product and exponentiation. The first kind is performed element by element:

```
a .* b %a and b have same dimensions
a .^ 2
```

while the second kind performs the matricial operations:

```
a * b %a #columns = b #rows
a .^ 2
```

Mathematical operators are performed element by element. Some of the most used built in functions related to vectors and mathematical functions in MATLAB are:

```

length(v) %length of vector
numel(M) %number of element in a matrix
size(M) %vectof of the matrix dimensions
find(p) % indexes of a vector satisfying p

round() % round of a number
ceil(x) % round toward positive infinity
floor(x) % round toward negative infinity
fix(x) % round toward 0

max(v) % maximum
min(v) % minimum
mean(v) % average value of a vector
mod(m,n) % module n operation

sin(x) %sine
cos(x) %cosine
exp(x) %base e exponential
log(x) %base e logarithm
```

Strings are vectors of char and are defined by single quotation mark `'text'`. To display a string we can either use `disp` or `fprintf()` (which has syntax similar to the one in C language):

```
disp('Hello world');
fprintf('%s\n','It is me');
```

Another structure to store generic data is the cell. This structure allows one to store heterogeneous data:

```
cell_vec = cell(3,1); %instantiate a cell vector of dimension 3
   x1
cell_vec{1} = 1;
cell_vec{2} = 'hello';
cell_vec{3} = zeros(4);
```

You can use the curly brackets to access elements contained in a cell, while round brackets to select a subset of the cell vectors. Similarly one can define a matrix of cells. Cells are less flexible than regular matrix since there is no easy way to perform operations over all the cell matrix and requires a bigger space to store the same data.[2]

1.4 Logical operators

We have the possibility to define logical conditions

```
a = 0;
b = 1;

a && b %and
a & b %bit by bit and
a || b %or
a | b %bit by bit or
~a %not bit by bit

a == b %equal
a ~= b %not equal
a > b %greater
a < b %smaller
```

Operation does not evaluate the second element of the logical formula if it is not necessary to infer its result. The bit by bit operations return a vector of logical indexes, by performing the logical operation element by element.

1.5 Data: loading and data visualization

To load data we have different options depending on the tool we want to use and the format of the file we want to load. At first, it is possible to import file with the *Import Tool* (Home tab → Import Data). Other options are available by relying on built in functions. In the case it is a standard file saved by MATLAB , whose extension is `.mat`, we can use the load function:

```
load('iris_dataset.mat');
load iris_dataset
```

[2] For a "complex" way of working on cells see `cellfun()`.

This instantiate in the workspace all the variables stored in the file. If we want to load a specific variable we have to specify its name:

```
load('iris_dataset.mat','irisInputs');
```

To save files we can use the `save` function, which has similar syntax:

```
save('results.mat');
save('results_reduced.mat','means','std_devs');
```

If we want to read from a `.csv` or from a table we can use:

```
init_row = 1;
init_col = 1;
end_row = 20;
end_col =
delimiter = ' ';
dlmread(filename,delimiter,[init_row, init_col, end_row,
   end_col])
```

If you do not specify the delimiter, the function tries to detect the delimiter, while if you do not specify rows and columns it reads the whole file.

At last, if the data to import are more structured and are stored in a plain text file, it is possible to use `textscan` (which has similar syntax to C language):

```
pippo = fopen('iris.txt','r');
dati = textscan(pippo, '%f%f%f%f%s', 'Delimiter', ',');
fclose(pippo);
```

Once you loaded a data the first operation one should perform is its visualization, if the dimensionality of the considered dataset allows it. To open a new graphical figure we use the command:

```
figure();
```

If we want to plot in the previously defined figure we should use the command `plot()`:

```
plot(y) %plots (1,y_1) ... (n,y_n)
plot(x,y) %plots (x_1,y_1) ... (x_n,y_n)
```

Remark 5. *A new plot on the same figure will delete the previously plotted data. To keep all the previous plot you can use `hold on`. To start again to erase previous plots you can use `hold off`.*

It is possible to specify the line style and color:

```
plot(x,y)          % blue continuous line
plot(x,y,'r')      % red continuous line
plot(x,y,'b--')    % blue dashed line
plot(x,y,'g-.')    % green dot-dashed line
plot(x,y,'c.')     % cyan dots
plot(x,x.^2,'-o') % blue continuous line with o marks
plot(x,x,'rx')     % red crosses
```

It is possible to add some labels and a title to the figure:

```
title('Nice plot');
legend('line name');
xlabel('time');
ylabel('space');
```

In the case we want to plot a 3D function $z = f(x, y)$ we can use the function `plot3()` or the function `surf()`, which have similar usage as `plot()`. Clearly, we have to specify the tuples $(x_i, y_i, z_i)_i$ for each point we want to plot:

```
[x, y] = meshgrid(0:0.1:1,0:0.1:1);
z = x.^2 - y.^2;
figure();
plot3(x,y,z);
figure();
surf(x,y,z);
```

If you want to plot data points you can either use a scatterplot or a its multivariate version, if data have more than two dimensions:

```
scatter(x,y)
```

which will plot circles in the points identified by tuples (x_i, y_i) or

```
gplotmatrix(X);
```

which plot a matrix of scatter plots for each couple of variables in the off-diagonal elements and depicts the empirical distribution of the variables on the diagonal.

If you want to plot an histogram you should use:

```
histogram(data,n_bins);
```

where `n_bins` is the number of bins you want to divide your data with.

Exercise 1.1

Write a MATLAB script to declare a vector x of 8001 evenly spaced points over $[-3; 5]$. Evaluate the function:

$$f(x) = 3\sin(4x) + x^2$$

over the considered points. Plot the graph of the function $f(x)$ in $[-3; 5]$ with a red dashed line.

Plot red circle over all the points of `f(x)` s.t. their x-coordinates are integer numbers.

Exercise 1.2

Given the matrix `A = [2 7 9 7 ; 3 1 5 6 ; 8 1 2 5]` provide the MATLAB commands able to:

1. assign the even-numbered columns of A to a matrix called B
2. assign the odd-numbered rows of A to a matrix called C
3. convert A into a 4-by-3 array
4. compute the reciprocal of each element of A
5. compute the square-root of each element of A

Exercise 1.3

Given `x = [3 15 9 12 -1 0 -12 9 6 1]`, provide the command(s) that will:

1. set the values of x that are positive to zero
2. set values that are multiples of 3 to 3 (rem will help here)
3. multiply the values of x that are even by 5
4. extract the values of x that are greater than 10 into a vector called y
5. set the values in x that are less than the mean to zero
6. set the values in x that are above the mean to their difference from the mean

Answers

Answer of exercise 1.1

```
clear
clc
close all

x = -3:0.001:5;
y = 3 * sin(x) + x.^2;

figure();
plot(x,y,'r--');

x_tilde = x(1:1000:8001);
y_tilde = y(1:1000:8001);
hold on;
plot(x_tilde,y_tilde,'ro');
```

Answer of exercise 1.2

1.

```
B = A(:, 2:2:4);
```

2.

```
C = A(1:2:3, :);
```

3.

```
A = reshape(A, 4, 3);
```

4.

```
1./A
```

5.

```
sqrt(A)
```

Answer of exercise 1.2

1.

```
x(x > 0) = 0;
```

2.

```
x( rem(x,3) == 0) = 3;
```

3.

```
x( rem(x,2) == 0) = x( rem(x,2) == 0) * 5;
```

4.

```
y = x( x > 10);
```

5.

```
x( x < mean(x)) = 0;
```

6.

```
x( x > mean(x)) = x( x > mean(x)) - mean(x);
```