Introduction to Matlab

Dr. Robin Roche

Université de Technologie de Belfort-Montbéliard

Contents

1 Getting help **2**

2 User interface **2**

3 Files **3**

4 Functions **3**

5 Variables **4**

6 Basic functionalities **5**
6.1 Inputing data . . . 5
6.2 Saving and importing data . . . 5
6.3 Reading and writing variables . . . 6
6.4 Operations on scalar values . . . 6
6.5 Operations on matrices and vectors . . . 7
6.6 Logical tests . . . 8
6.7 Loops . . . 9

7 Plotting data **9**

8 Artifical neural networks **10**

9 Miscellaneous **11**
9.1 Sorting data . . . 11
9.2 Generating lists of binary combinations . . . 12
9.3 Other functions . . . 12

10 Programming skills **12**
10.1 Troubleshooting . . . 12
10.2 Interacting with users . . . 13
10.3 Tips . . . 13
10.4 Speeding up your code . . . 15
10.5 Documenting code . . . 15
10.6 Other resources . . . 15

Matlab is a numerical computing environment, with its own high level programming language, that runs on Windows, Mac, and Linux. It is widely used in academia, research institutions and industry. A free alternative to Matlab is GNU Octave, which uses a similar language, but has a minimal user interface and slightly less functionalities.

This tutorial is a guide designed to help students who are not familiar with Matlab, so that they are able to use it rapidly.

1 Getting help

When needing some help, try the following:

1. Read this tutorial,
2. **Use Matlab's help file**, using the `help` function (example: `help lsqcurvefit`), by pressing F1, or by browsing Matlab's online documentation,
3. Run a Google search with your problem, you are probably not the first one to face it.

2 User interface

Matlab's user interface (UI) is organized as shown in Fig. 1. Parts of the UI may be organized differently (e.g., depending on the version); you may reorganize it manually by dragging parts where you want them to be.

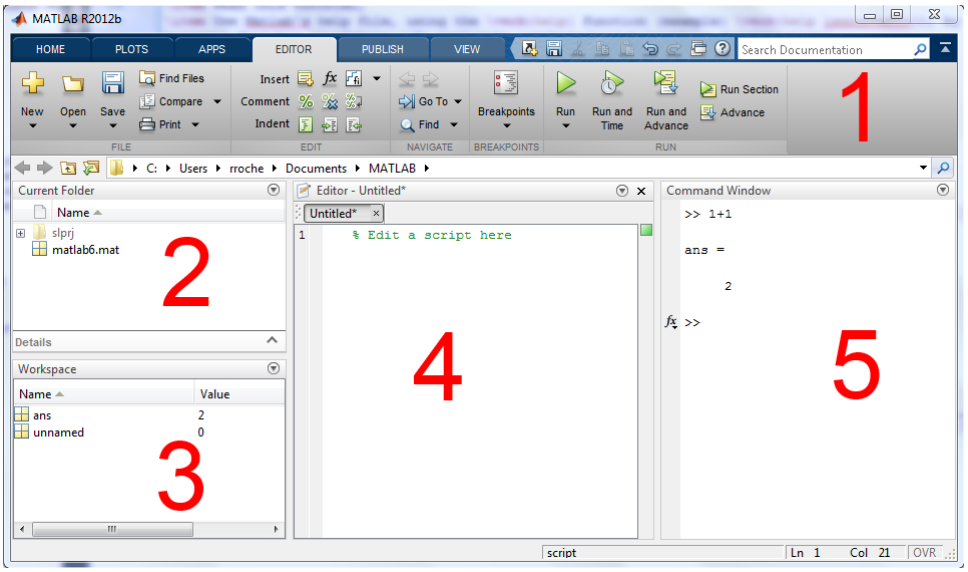

Figure 1: Screenshot of Matlab's user interface

Each area of the interface has a specific role (see numbering on Fig. 1):

1. *Toolbars* enable easy access to basic functionalities.
2. The *Current folder* displays the files in the current directory. Files outside of this folder are not accessible directly.
3. The *Workspace* lists the existing variables, and some basic characteristics (min., max., etc.).
4. The *Editor* is where you can edit `.m` files.

5. The *Command window* is where you can type basic, one line instructions and see their output, as well as the output of scripts and functions. You may clear the content of the window with `clc`.

3 Files

Matlab uses the following types of files:

- `*.m` files for scripts and functions,
- `*.asv` for auto-save files (removed after a file is saved),
- `*.mat` files for storing variables and data,
- `*.mdl` files for Simulink models.

The two types of `.m` files can be differentiated as follows:

- A script is to be run directly, takes no input and does not return anything directly,
- A function may take inputs, has specific format, and can return one or several outputs.

File names can contain letters and numbers, but cannot start with a number. They should also not contains hyphens (`-`), but underscores (`_`) are allowed. Some terms, such as `load`, `save`, `sim`, and others, also correspond to built-in functions and should not be used as file names.

4 Functions

Matlab function files are structured as follows:

```
function [ output_args ] = Untitled2( input_args )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
...
end
```

Here is an example (`myfunc.m`) that simply returns the sum of two values:

```
function c = myfunc(a,b)
    c=a+b;
end
```

The function output is as follows:

```
>> myfunc(1,2)
ans =
     3
```

Note that the file name should be exactly the same as the function name, and that variables in a function are not accessible outside of this function (and also not in the workspace).

5 Variables

Variables do not need to be declared, and there is no need to define their type (integer, float, string, etc.).

Variable names can contain letters and numbers, but cannot start with a number. Case is important, and the same variable name with two different case letters are considered as two different variables. Some terms, such as `load`, `save`, `sim`, and others, also correspond to built-in functions and should not be used as variable names. Several other names are used for specific constants or values: `i`, `j`, or `1i` for the complex number, `NaN` for an undefined value, `pi` for Π, `eps` for the smallest value Matlab can handle, `ans` is the last output, and `inf` for ∞ (among others).
By default, variables are only 'visible' in the function or file they are defined. Only variables defined in the top script are also visible in the workspace. Global variables can be defined, but their use is discouraged, except if no other solution is feasible.

In Matlab, almost everything is considered as a matrix: a scalar value is a 1×1 matrix, a vector is a $1 \times n$ matrix, etc. The main variable types are: logical (boolean), numeric (integer, float, complex, etc.), characters or strings, and cell arrays. All variables types can only include a specific type of data, except cell arrays, that can contain data of varying types and sizes.

To assign a value to a variable, use the `=` operator:

```
>> a=5
a =
     5
>> a=5;
```

For all statements, adding a semicolon at the end of a line of instructions does not output anything (except errors). To display the content of a variable, just input its name:

```
>> a
a =
     5
```

To determine the type of a variable, Matlab includes several specific functions (see full list here):

```
>> isnumeric(a)
ans =
     1
>> ischar(a)
ans =
     0
```

To convert a variable from a type to another, other specific functions are defined (see full list here). For example, to convert a character or string to a numeric value:

```
>> b='5'
b =
5
>> ischar(b)
ans =
     1
>> b2=str2num(b)
b2 =
     5
>> isnumeric(b2)
ans =
     1
```

6 Basic functionalities

6.1 Inputing data

Data is typically stored as matrices. All matrices indices start at 1, not 0, i.e., `A(0)` does not exist.

To create a vector of regularly spaced values, use the instruction `a:b:c`, where `a` is the first value, `b` is the interval between two consecutive values, and `c` the maximum value. An example is as follows:

```
>> x = 0:3:10
x =
     0     3     6     9
```

You may also input data directly, and split an instruction on several lines with '`...`':

```
>> d=[1 4 5; 3 12 9]
d =
     1     4     5
     3    12     9
>> A=[1 2 3 ...
4 5 6]
A =
     1     2     3     4     5     6
```

Initialize a matrix with '0', '1', and 'NaN' with the functions `zeros`, `ones` and `NaN`:

```
>> y=ones(2,3)
y =
     1     1     1
     1     1     1
>> z=zeros(1,3)
z =
     0     0     0
>> a=NaN(1,3)
a =
   NaN   NaN   NaN
```

To obtain the size of a matrix, use the `size` function, or the `length` function that returns the size of the longest dimension:

```
>> size(y)
ans =
     2     3
>> length(y)
ans =
     3
```

6.2 Saving and importing data

Variables from the workspace can be saved to a `.mat` file, e.g., to store data or send it to someone else. The first solution is to select the variables to save, right-click on one of them, and select 'Save As'. Another solution is to use the `save` function, that saves all variables in the workspace to a specified `.mat` file:

```
>> save('backupfile')
```

Data can be imported through several means:

- Direct input of data, as shown earlier,
- Loading variables in a `.mat` file,
- Importing data from a `.csv` or `.xls` (or `.xlsx`) file.

To load the content of a `.mat` file, use the `load` function, that loads the variables in the workspace:

```
>> load('backupfile')
```

Matlab also include a very powerful tool to import data, e.g. from `.xls`, `.csv` or `.txt` files. This tool is accessible from the 'Home' tab, under 'Import Data'. This tool is able to load data with different formats, to separate data to several variables, remove text or specific lines, etc.

6.3 Reading and writing variables

To read a value or several ones in a variable:

```
>> x=1:6
x =
     1     2     3     4     5     6
>> x(3)
ans =
     3
>> x(3:5)
ans =
     3     4     5
```

A similar approach is used to write values in variables:

```
>> x(4)=6;
>> x
x =
     1     2     3     6     5     6
>> x(1:3)=10;
>> x
x =
    10    10    10     6     5     6
```

Parts of matrices can also be extracted. For example, the second column of a matrix `z` can be obtained as follows:

```
y=z(:,2);
```

Similarly, to overwrite the second column of a matrix `z` and the third line of a matrix `z2`:

```
z(:,2)=10;
z2(3,:)=10;
```

6.4 Operations on scalar values

All basic operations on scalar values are available:

```
>> c=1;
>> d=2;
>> c+d
ans =
     3
>> c*d
```

```
ans =
     2
>> c-d
ans =
    -1
>> c/d
ans =
    0.5000
```

Multiple other basic operators are also available:

```
>> min(c,d)
ans =
     1
>> max(c,d)
ans =
     2
>> mod(11,3)
ans =
     2
```

Other types of common operators include:

- Complex numbers: `real`, `imag`, `abs`, `angle`, `conj`,
- Trigonometry: `cos`, `sin`, `tan`, `acos`, `cosh`, etc.,
- Exponents and logarithms: `exp`, `log`, `log10`, `sqrt`, etc.

6.5 Operations on matrices and vectors

To compute the transpose of a matrix:

```
>> a=[1 2 3 4];
>> a2=a'
a2 =
     1
     2
     3
     4
```

Most operations can also be done on vectors and matrices. For example, to compute the sum of two vectors:

```
>> b=[1 1 1 1];
>> c=a+b
c =
     2     3     4     5
```

Be careful with nonlinear operators, such as products, exponential functions, etc. By default, Matlab will use a matrix product if you use the `*` operator. If you want to compute a scalar product (i.e., each element of a vector multiplied by the element of the other vector at the same position), add a `.` before the operator, as show in the example below. The same applies for other nonlinear operators.

```
>> d=a2*b
d =
     1     1     1     1
     2     2     2     2
     3     3     3     3
```

```
     4     4     4     4
>> a.*b
ans =
     1     2     3     4
>> d/b
ans =
     1
     2
     3
     4
>> b./a
ans =
    1.0000    0.5000    0.3333    0.2500
```

To solve systems of linear equations $Ax = B$ for x, use (note the order):

```
x = A\B
```

Matrices can also be concatenated as follows:

```
>> [a b]
ans =
     1     2     3     4     1     1     1     1
>> [a' b']
ans =
     1     1
     2     1
     3     1
     4     1
```

To sum the elements of a vector:

```
>> sum(a)
ans =
    10
```

To find the minimum (or maximum using `max`) value of a vector and its index (i.e., location in the vector):

```
>> c = [2 7 3 0 4];
>> [minVal, minIndex] = min(c)
minVal =
     0

minIndex =
     4
```

6.6 Logical tests

Logical tests are commonly used, e.g., to determine if a statement is true or not. Tests can be done on boolean values (`true` or `false`, or `1` or `0`, respectively), numbers, as well as characters or strings. The following operators are used:

- `a==b` to test if both values are equal,
- `a~=b` to test if both values are different,
- `a<=b` to test if `a` is smaller or equal to `b`.

```
>> r=5;
>> r>=0
ans =
     1
>> r==5
ans =
     1
>> r~=5
ans =
     0
```

Logical AND and OR operations are also possible, with the following instructions:

```
>> p=1;
>> q=0;
>> p&&q
ans =
     0
>> p||q
ans =
     1
```

6.7 Loops

Two main type of loops are available in Matlab: `for` and `while` loops. The following code snippets have same output shown below, and are therefore equivalent:

Source code

```
for i=1:3
    i
end
```

```
i=1
while(i<3)
    i=i+1
end
```

Output

```
i =
     1
i =
     2
i =
     3
```

To choose between `for` and `while` loops, follow this simple rule: if you now how many iterations are needed (e.g., a sum from 1 to N), use a `for` loop. Otherwise, use a `while` loop, e.g., to check whether a criterion is met.

7 Plotting data

Regular plots are simple to obtain in Matlab. The easiest way is to select the variable to plot in the workspace, and click on desired plot type in the 'Plots' type of the top toolbar. Obviously, more complex types of plots are achievable using specific commands. Learn more using Matlab's help file on this topic.

An X-Y plot can be obtained with the `plot` function, as shown in the example below (see output in Fig. 2). It is also possible to add labels to the x and y axes (`xlabel`, `ylabel`), a title (`title`), as well as to specify the limits on both axes (`xlim`, `ylim`). A new figure is created with the `figure` instruction, otherwise a newer figure replaces a previous one.

```
figure
x = 0:0.5:2*pi;
```

```
y = sin(x);
plot(x,y)
title('Graph title')
xlabel('Title of the x-axis')
ylabel('Title of the y-axis')
xlim([0 pi])
ylim([0 1.1])
```

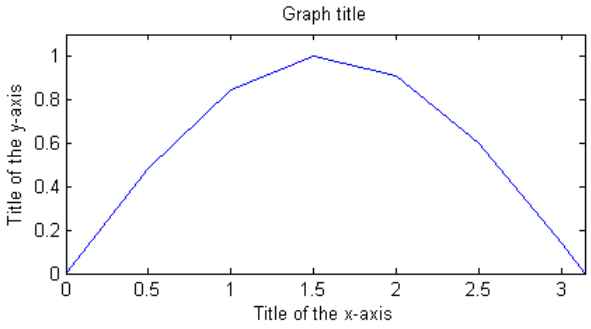

Figure 2: Output of the first set of instructions

Several lines can be plot on the same graph (see output in Fig. 3), and their style can be customized:

```
figure
x = 0:0.5:2*pi;
y = sin(x);
y2 = cos(x);
plot(x,y,'r--o',x,y2)
```

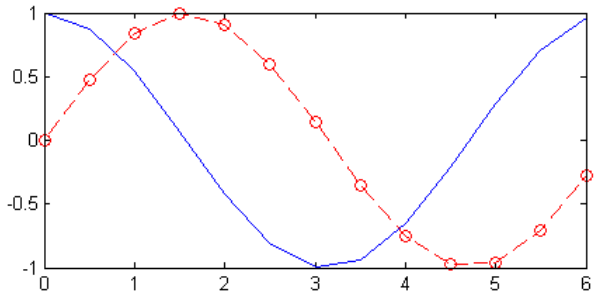

Figure 3: Output of the second set of instructions

It is also possible to superimpose several plots, using the `hold on` instruction after the first plot. It is then necessary to change the color of the lines to distinguish them:

```
plot(x,y1,'b')
hold on
plot(x,y2,'r')
```

8 Artifical neural networks

The Neural Network Toolbox enables designing and using artificial neural networks (ANN). Among other applications, an ANN can be used to "learn" an unknown function, e.g., to forecast electric load. A network must be *trained* using known input/output data (i.e., training data), so it can later be used to predict future data based on new inputs (i.e., test data).

The user interface can be launched by typing `nntool`. However, using this tool requires inputing the input and output (target) vectors manually each time you run the tool. It is thus more interesting to create, train and use networks through commands:

```
// Create the input matrix, with one input vector per line
input(1,:) = vectorInput1;
input(2,:) = vectorInput2;
input(3,:) = vectorInput3;

// Create a feed-forward network, with nbNeurons neurons in the middle layer
net = newff(input, knownOutput, nbNeurons);

// Train the system, using training data
net = train(net, training_input, training_knownOutput);

// Test the ANN, using test input data
testOutput = sim(net, testInput);
```

9 Miscellaneous

9.1 Sorting data

The `sort` function can be used to sort data, in ascending or descending order. In the following example, data is sorted in ascending order according to the second column, and data data is stored as a cell array.

Source code

```
% Input unsorted data
unsortedData = {...
    'a', 90;...
    'b', 60;...
    'c', 150;...
    'd', 250;...
    'e', 80;...
    'f', 220;...
    'g', 300}
dataColumn1 = char(unsortedData(:,1))';
dataColumn2 = cell2mat(unsortedData(:,2))';

% Sort data
[sortedDataColumn2, sortedIndex] = sort(dataColumn2);
sortedDataColumn1 = dataColumn1(sortedIndex);

% Output sorted data
disp('Sorted data:')
for i=1:length(unsortedData)
    fprintf('%s \t %d \n',sortedDataColumn1(i),...
     sortedDataColumn2(i))
end
```

Output

```
unsortedData =
    'a'    [ 90]
    'b'    [ 60]
    'c'    [150]
    'd'    [250]
    'e'    [ 80]
    'f'    [220]
    'g'    [300]

Sorted data:
b   60
e   80
a   90
c   150
f   220
d   250
g   300
```

9.2 Generating lists of binary combinations

To generate a list of all combinations of binary values of a given length, you may simply convert decimal numbers to binary, as follows (here, for 4 binary variables):

Source code

```
dec2bin(0:2^4-1)-'0'
```

Output

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

9.3 Other functions

Many specific functions are available in Matlab. The following ones may be helpful:

- `cumsum`: returns the cumulative sum of a vector,
- `unique`: returns the unique values in an array (i.e., remove duplicates),
- `repmat`: replicates and tiles an array,
- `reshape`: reshapes an array, e.g., to transform a vector into a matrix,
- `rand` or `randn`: returns a random number between 0 and 1 (uniform or normal distribution),
- `randi`: returns a random integer between 0 and a given value (uniform distribution),
- `lsqcurvefit`: fits a curve in least-squares sense,
- `stairs`: draws a stairstep graph,
- `linprog`: optimizes a function given a set of constraints using linear programming,
- `tic` and `toc`: returns the execution duration of a set of instructions.

Check out Matlab's help for more details. More functions will be added as needed.

10 Programming skills

10.1 Troubleshooting

There is no general method to debug your code. However, it is good practice to follow these steps:

1. Properly read and understand the error message, it usually tells you where the error is and what is wrong: variables do not have the proper size, a variable does not exist (i.e., usually, you did not type its name correctly), etc.

2. Add comments in your code (i.e., print something in the console where you think there is a problem) and re-run it. This may help you trace where the problem happens.

Clearing all variables is also useful, especially when errors mentioning variables with the wrong size appear: use the `clear all` instruction.

To stop an infinite loop, select the *Command Window* and use `Ctrl + C`. This should "brutally" stop the loop.

10.2 Interacting with users

The main way to interact with users is to display information, using the `display` (or `disp`) or `fprintf` functions. Display can only include text or a variable, but not both or several variables.

```
>> display('This is a test')
This is a test
>> disp(p)
     1
```

The `fprintf` function enables displaying variables along with text. Check out this page for more details:

```
>> i=2.000234;
>> fprintf('Value of i = %.4f\n',i)
Value of i = 2.0002
```

As a general rule, it is discouraged to ask the user to input data, expect if really necessary. It is better to input data directly in the script, as it only requires inputing data once.

10.3 Tips

10.3.1 Sums

The `sum` function described earlier may be used for summing the elements of a vector. However, for other types of sums, a different approach must be used, with a `for` loop and a temporary summing variable. For example, to compute $\sum_{i=1}^{3} i^2$:

```
tempSum = 0;
for i=1:3
    tempSum = tempSum + i^2
end
```

10.3.2 Filling a matrix

To fill an empty matrix, two embedded loops may be used. If is also possible to check whether an element is on the matrix diagonal by comparing the indices of both loops.

Source code

```
n = 3;
for i=1:n
    for j=1:n
        if(i==j)
            A(i,j) = 1;
        else
            A(i,j) = 0;
        end
    end
end
A
```

Output

```
A =
     1     0     0
     0     1     0
     0     0     1
```

10.3.3 Vector to matrix conversion

In some cases, it can be useful to transform a matrix into a vector. This may be done by adding the second line at the end of the first line, and then the third line, etc. The process can also be reverted, as shown below.

```
n = 3;
m = 4;

% Create the initial vector
A = 1:(n*m)

% Reshape vector into a matrix (short version)
B = reshape(A,n,m)'

% Reshape vector into a matrix (long version)
for i=1:m
    minIndex = (i-1)*n+1;
    maxIndex = i*n;
    B2(i,:) = A(minIndex:maxIndex);
end
B2

% Reshape matrix into a vector (short version)
C = reshape(B',1,n*m)

% Reshape matrix into a vector (long version)
for i=1:m
    for j=1:n
        C2((i-1)*n+j) = B2(i,j);
    end
end
C2
```

The output is as follows:

```
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

B2 =
     1     2     3
     4     5     6
     7     8     9
    10    11    12

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

C2 =
     1     2     3     4     5     6     7     8     9    10    11    12
```

10.4 Speeding up your code

In some cases, you may have to run long loops, e.g., to test a large number of possible solutions. This can result in a very long script run time. To try to shorten this run time:

- Avoid as much as possible copying or creating new variables,
- Reduce the number of loops and combine several ones when possible,
- Use Matlab's built-in functions, when they exist,
- Use vector or matrix operations instead of element-by-element operations,
- Do not print anything in the console,
- Pre-allocate matrices, i.e., initialize them,
- Use `mlint`, Matlab's code analyzer.

Pascal Getreuer's document may also be useful.

10.5 Documenting code

It is of the utmost importance to properly document your code, e.g., simply by adding comments (after a `%` sign) every few lines. These comments enable your code to be understood by other people (including the person grading it), and may also be helpful if you reuse this code much later.

10.6 Other resources

Below is a list of examples of other resources on Matlab:

- MathWorks' tutorials
- TutorialsPoint's tutorial
- Introduction to Matlab for engineering students by David Houcque
- Control tutorials for Matlab & Simulink
- A beginner's guide to Matlab by Christos Xenophontos
- MIT's course on Introduction to Matlab

In addition to these, and of Matlab's documentation of course, I also encourage you to take a look at videos you may find on Youtube.