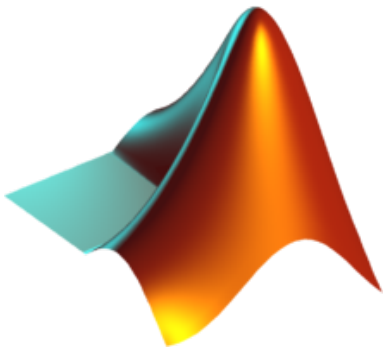
University of Liège
Faculty of Applied Sciences

Introduction to MATLAB

TUTORIAL

(English version)

This tutorial has been prepared by the Faculty of Applied Sciences (Engineering) in order to provide a basic knowledge of Matlab for engineering studies. It is an English translation of the document initially prepared in French for the course "PROJ0001 - Introduction aux méthodes numériques et projets"



Academic year 2020-2021

1 Introduction

This tutorial is intented for students who are following the MATLAB training 2018-2019. Figure 1 shows the MATLAB interface.

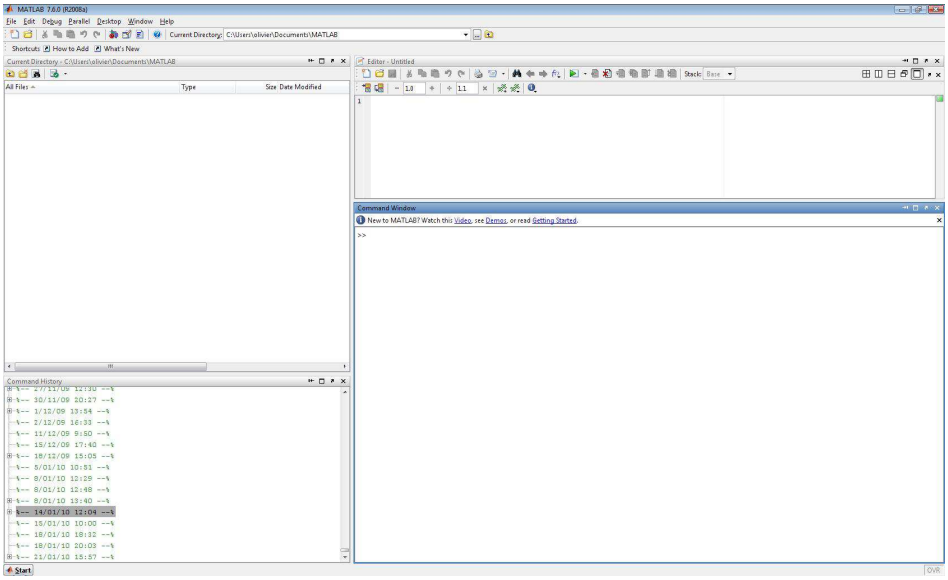


FIGURE 1 – MATLAB interface.

2 Command Window

The command window allows the user to enter commands after "`>>` ". In particular, we can use the following operators : additional operator "`+`", difference operator "`-`", multiplication operator "`*`", division operator "`/`" and power operator "`^`". The assignment operator allows to define variables. A variable is an information container, in the form of a scalar or a table, containing numbers, characters and so on. If the user does not wish to display the result of a command in the command window, the instruction must be finished by "`;`".

```
>> a = 2+2
a =
     4
```

The "`clc`" command enables to erase all of the command window.

3 Help

The user can access local help with the following commands : "`help`" (in the command window) and "`doc`" (in the help browser of MATLAB). Note that the first command gives an overview of information available in the second one.

```
>> help clc
 CLC    Clear command window.
    CLC clears the command window and homes the cursor.

    See also home.

    Reference page in Help browser
       doc clc

>> doc clc
```

Next, always locally, MATLAB integrates demonstrations which may be useful to review.

```
>> demo
```

Finally, we have a lot of resources available on the following website :
`http://www.mathworks.com/products/matlab/`

Exercise

1) Search for information in the help on the following functions "`format long`" & "`format short`" and test their operation.

4 Command History

The order history gives users an overview of commands entered during the current session and the last sessions. Therefore, it enables to reuse a command easily. In the command window, we can run through it with "↑" & "↓" buttons.

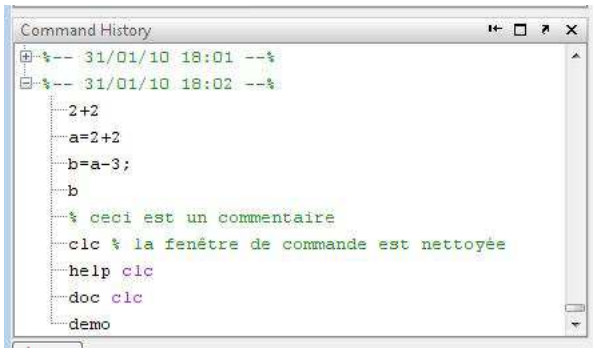


FIGURE 2 – Command history

5 Workspace

The workspace provides information to users on variables already defined. It has features for manipulating variables (create, edit, delete, ...). This information is also available thanks

to the commands "`who`" and "`whos`".

The variable "`ans`" corresponds to the most recent unassigned return value of MATLAB (here it is the result after commands described at Section 2).

```
>> who
Your variables are:
a    ans  b

>> whos
  Name      Size            Bytes  Class     Attributes
  a         1x1                 8  double
  ans       1x1                 8  double
  b         1x1                 8  double
```

The "`clear`" command is used to delete a variable or to erase all workspace if it is called with keyword "`all`".

```
>> clear a
>> who
Your variables are:
ans  b

>> clear all
>> who
```

6 Current Directory and Editor

The current directory is the directory where the user is working. The contents of the directory is posted and features for contents maintenance are given. You can use the "`dir`" command to display all the contents of the directory.

```
>> dir
.   ..
```

We create an "`hello.m`" file with command "`edit`".

```
>> edit hello
```

After the editor opens the file, we can write the following commands :

```
% Script affichant 'Hello World!'
text = 'Hello World!';
disp(text);
```

We save the file and return to the command window. The "`what`" command displays specific MATLAB files of the current directory.

```
>> dir
.         ..        hello.m

>> what
M-files in the current directory /home/sburton/matlab
hello
```

The MATLAB files of the current directory can be called from the command line directly. The execution of "`hello.m`" file is done by entering its name, without an "`.m`" extension in the command window. The commands saved in the file are then carried out, that is, text "`Hello World!`" is assigned to the "`text`" variable and it is displayed.

```
>> hello
Hello World!
```

So, we can observe :

```
>> who
Your variables are:
text

>> text
text =
Hello World!
```

Finally, call to `help hello` returns the first comment block of the file.

```
>> help hello
  Script affichant 'Hello World!'
```

It is also possible to open the editor and create a new file thanks to the "New Script" icon typically located at beginning of the second line of the MATLAB interface. Once the file is saved, it is possible to open it with the "`edit`" command or with a double-click on the file name present in the current directory.

7 Functions

The file "`hello.m`" is a script, that is, a program file with .m extension. In these files, you write series of commands, which you want to execute together. They operate on data in the workspace. Scripts do not accept inputs and do not return any outputs. By contrast, a function is also a program file with .m extension, that can accept inputs and return outputs. Internal variables are local to the function.

We create the "`byebye.m`" file

```
edit byebye
```

We enter the following commands

```
% Function displaying 'Bye bye World!'
function byebye
text = 'Bye bye World!';
disp(text);
end
```

We save the file and return to the command window. The file "`byebye.m`" execution returns

```
>> byebye
Bye bye World!
```

In this case, we observe :

```
>> who
Your variables are:
text

>> text
text =
Hello World!
```

Because the "`byebye`" function acts in a subenvironment of the work environment, the function execution didn't alter the "`text`" variable defined in the work environment.

Global variables, defined thanks to keyword "`global`", are a notable exception to this behavior in the sense that their scope is global and not local. There are also persistent variables, defined thanks to keyword "`persistent`", which retain their value between function calls where they are defined. For security reasons, we rarely use these kinds of variables (not recommended).

Functions can take several arguments as input and as output. For example, below is a function `polarToCartesian` which receives polar coordinates as input and returns cartesian coordinates as output.

```
% This Function returns cartesian coordinates of a point
% from its polar coordinates.
function [x y] = polarToCartesian(rho,theta)
    x = rho * cos(theta);
    y = rho * sin(theta);
end
```

We call it by the following command :

```
[x y] = polarToCartesian(rho,theta)
```

where input arguments, which are needed information for the smooth running of the function, are always located on the left side of the function name in square brackets. It is important to note that the function name must be the same as filename `.m`.

Many functions are natively integrated in MATLAB. For example, we can find the following functions :

— `pi` : returns the π value ;

— `sin` : returns the sine of an angle in radians ;
— `sqrt` : returns the square root of a number ;
— `ceil` : returns the higher round of a number.

MATLAB's help and its search options allow to find useful functions and to get information for their use.

Exercise

Consider the following algebraic equation :

$$ax^2 + bx + c = 0 \tag{1}$$

Create a function which receives arguments a, b and c as input and returns the two square roots as output.

8 Control structures

MATLAB control structures include :
— `if` : consists of a boolean expression followed by one or more statements ;
— `switch` : allows a variable to be tested for equality against a list of values ;
— `for` : executes a sequence of statements multiple times and abbreviates the code that manages the loop variable ;
— `while` : repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

MATLAB's help gives further information on these control structures.

Exercises

1) The exponential function can be represented by the following series

$$\exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!} \tag{2}$$

Write a MATLAB function that calculates this series limited to the $n+1$ first terms. You can use the "`factorial`" function and the "`^`" operator.

2) Same exercise as before but without using the "`factorial`" function and the "`^`" operator (only the following operators "`+`", "`*`" and "`/`" are allowed).

9 Vectors, matrices, cells and structures

MATLAB is a programming language which enables to manipulate scalars, vectors, matrices and tables. Therefore, we can do some operations with matrices and functions can take matrices as input.

There are different methods used to define vectors, test them :

```
>> a = [1 3 5 7 9]

>> b = 1:2:10

>> c = linspace(-pi,pi,5)

>> d = rand([1 5])
```

We can have access to elements of a vector by using linear indices (from "`1`" for the first one to "`end`" for the last one). The "`:`" operator represents a set of components of the vector. Familiarize yourself with vectors manipulation and commands below :

```
>> a(1)

>> a(:)

>> b([2 end])

>> c(3:end-1)

>> d(a(2))
```

We can also modify the elements of a vector created earlier :

```
>> a(1) = 10

>> b([end+1 10]) = [11 12]
```

The "`length`" function returns the vector length and the "`sort`" function enables to sort the vector values (second output argument identifies corresponding indices in the initial configuration).

```
>> length(a)

>> [c index] = sort(a)
```

Matrices can be defined thanks to following commands :

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

>> B = eye(3)

>> C = zeros(2,3)

>> D = diag([7 1 3])
```

MATLAB's help contains more details on these functions. We can have access to the matrix elements by using indexes of lines and columns, or linear indexes. Test following commands :

```
>> A(2,1)

>> C(:,end)

>> D(9)
```

We can obviously modify matrix elements in a similar way as for vector modification.

The "`size`" function returns matrix sizes and the "`find`" function enables to find linear indices corresponding to elements which respect a given equality or inequality.

```
>> size(A)

>> find(A > 2)

>> A(find(A > 2)) = 0
```

Notice three usual mistakes : Firstly, the difference between the matrix multiplication operator "`*`" and the multiplication operator element-by-element "`.*`", then the difference between the transposing operator "`.'`" and complex conjugate transposing operator "`'`", and finally the existence of an empty element "`[]`" easily identified due to its zero length. The `cell` command enables to create tables of cells containing variables of any size. It can also contain scalars, vectors or matrices. For example, the four matrices defined earlier can be integrated in one table of cells :

```
>> TAB = cell(2)

>> TAB{1,1} = A

>> TAB{1,2} = B

>> TAB{2,1} = C

>> TAB{2,2} = D
```

In the same way as for vectors and matrices, it is possible to gain access to an element thanks to indices. Note that cell indices are located within curly brackets.

```
>> TAB{1,2}(2,2)

ans =
     1

>> TAB{2,2}(1,1)

ans =
     7
```

Finally, a structure is a table in which we gain access to different elements thanks to field names. In the same way as for cells, the elements can be of different types. A basic structure uses the following syntax :

```
>> Struct.elem1 = a

>> Struct.elem2 = B

>> Struct.elem3 = 5
```

It is then possible to look for a particular element of a vector or a matrix via its indices. More complex structures can be defined thanks to the `struct` command.

Exercises

1)

- Create a row vector "`x`" of 5 successive numbers between 2 and 3 separed by equal intervals.
- Add 1 to the second element.
- Create a second row vector "`y`" with the same size as the first one but the elements are successive even numbers beginning with 4.
- Create a matrix "`A`", with"`x`" as its first row, the second row is fully completed by 1 and with"`y`" as its third row.

2) Consider the following two matrices

$$A = \begin{bmatrix} 4 & -5 \\ \sqrt{3} & \pi/4 \end{bmatrix},\ B = \begin{bmatrix} 2 & 3+i \\ -72/3 & 0.2 \end{bmatrix} \tag{3}$$

where i is the imaginary unit. Calculate the following elements :

$$A+B,\ AB,\ A^2,\ A^T,\ B^{-1},\ B^T A^T,\ A^2+B^2-AB \tag{4}$$

10 Solving systems of equations

To solve a linear system $Ax = b$, the left division operator "\" can be used :

```
x = A\b
```

This operation can be written for a square or rectangular matrix A. Further helpful functions to solve linear systems are "`lu`" for LU decomposition, "`det`" to compute the determinant of a matrix, "`inv`" for the inverse of a matrix, "`rank`" for the rank, "`cond`" for the condition number in the 2-norm, or "`rcond`" for an estimate of the reciprocal condition of A in 1-norm.

Exercise

1) Solve the following system of linear equations

$$\begin{aligned} 5x + 6y + 10z &= 4 \quad (5)\\ -3x \qquad + 14z &= 10 \quad (6)\\ -7y + 21z &= 0 \quad (7) \end{aligned}$$

Solution: $x = -1.4545,\ y = 1.2078,\ z = 0.4026.$

11 Plotting

MATLAB provides powerful features to generate plots and graphics. The simplest plot is made by using the function "`plot`". To display several curves overlaid on a single figure, we must activate the command "`hold on`" to keep the curves on the same figure and use the commande "`hold off`" to disable the hold. The function "`close`" closes the active figure while the command "`close all`" closes all the figures.

Further functions allow to define the dispay area ("`axis`"), a grid ("`grid`"), a title ("`title`") or labels for the different axes ("`xlabel`", "`ylabel`") :

```
t = 0:pi/20:2*pi;
y = sin(t);
plot(t,y)
axis ([0 2*pi -1.5 1.5])
grid
title('sine')
xlabel('time t')
ylabel('sine(t)')
```

To export the figures in a format which can then be used with a word processing software, there are several options. From the window that contains your plot, go to File -> save as and choose the right format (use preferably uncompressed images). You can also use the "`print`" and "`saveas(...)`" functions by specifying the saved file in the command window or in the editor. The commands below save the figure in the file sinus.pdf :

```
print -dpdf -r300 sinus
saveas(gcf,'sine.pdf')
```

where `-r300` specifies a resolution of 300dpi and `gcf` is an handle that is associated with `plot(...)`.

Make sure the font size, the curves and also points are large enough to be readable later (see Fig. 3 and Fig.4). The following commands allow you to change the default font size used in the graphs (title and axis labels) :

```
set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);
```

That is also applied to modify the default line width or axis size, as described below :

```
set(0,'defaultaxeslinewidth',2);
set(0,'defaultlinelinewidth',2);
```

The label font or the thickness of lines can still be separately modified using the `'fontsize'` and `'linewidth'` options respectively. Refer to the Help in the command window to modify other properties.

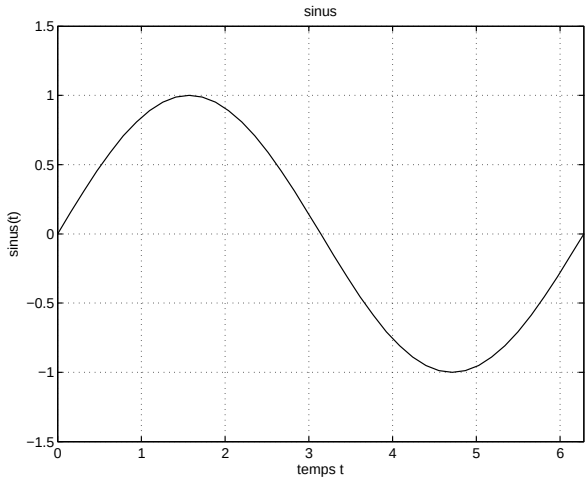


FIGURE 3 – Default font size and line width.

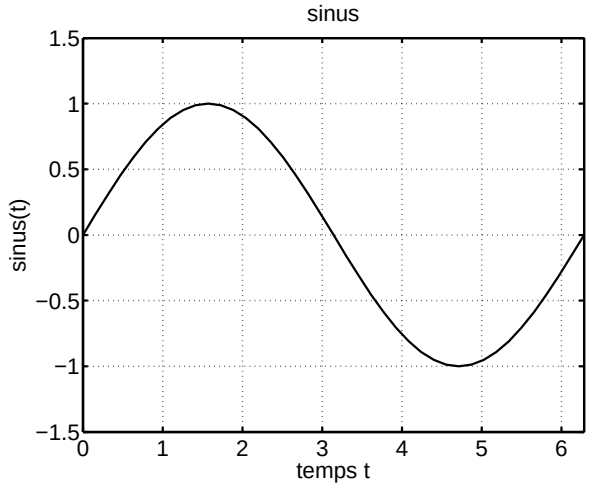


FIGURE 4 – Readability improved with a proper choice of the font size and the line width.

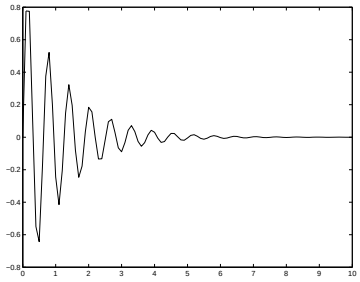
Exercises

1) Draw the function

$$y(x) = e^{-0.8x}\sin\omega x \tag{8}$$

for $\omega = 10$ rad/s et $x \in [0\ 10]$ s. Use the operator "`:`" to define the vector "`x`" with an increment of 0.1 s.

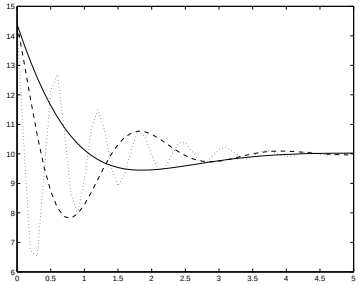
Solution:



2) We have the function

$$y(x) = 10 + 5e^{-x}\cos(\omega x + 0.5) \tag{9}$$

Write a script which can plot the function for $\omega = 1, 3, 10$ rad/s and $x \in [0\ \ 5]$ s. These three curves must appear in black, with a solid line for $\omega = 1$ rad/s, a dashed line for $\omega = 3$ rad/s and a dotted line for $\omega = 10$ rad/s.
Solution:



12 Input/Output

For the development of an interactive program, the screen output and the keyboard input can be realized using the functions "`disp`" and "`input`".

The variables are saved and loaded using the functions "`save`" and "`load`". For a more flexible management of the reading and the writing of a file, we can use the functions "`fopen`", "`fclose`", "`fprintf`", "`fscanf`", "`fgetl`" and "`fgets`".

The concatenation of character strings is made by means of the operator "`[]`". It can be useful to transform numbers into character strings and vice versa with the following functions "`num2str`" and "`str2num`". For a more flexible management of the character strings, we can use the functions "`sprintf`" and "`sscanf`".

13 Vectorial Functions

To improve the efficiency of a program, it is recommended to avoid loops "`for`" and "`while`" and to prefer the vectorial functions of MATLAB. These functions define operators on all the elements of a matrix or a vector in a single command.

For example, the functions "`prod`", "`sum`" and "`diff`" define operations of product, sum and difference for all the elements of a matrix or a vector. Here are other useful functions : "`max`", "`min`" and "`mean`". We often meet the functions "`sort`", "`find`", "`zeros`", "`ones`" too as well as the operator "`:`". The use of these functions is thus deeply recommended.

Exercise

1) Without using a loop, write a function which receives as argument a vector "`x`" from arbitrary dimension "`n`" and a scalar "`t`" and which calculates a scalar "`y`" as

$$y = (x(1) - t) * \ldots * (x(n) - t) \tag{10}$$

14 Stuck...

There is not only the Help which can help the user in case of a problem. At first, in case of misuse of MATLAB, an error message often appears. For example, the sine of a character "`a`" has no sense, an explicit error message is sent back.

```
>> sin('a')
??? Undefined function or method 'sin' for input arguments of
type 'char'.
```

Then, if there is syntactic errors in your code, the analyzer will underline in red the wrong part of the code. A contextual help to the resolution of these problems is supplied.

It is useful to mention the existence of « numbers » "`i`" (imaginary unit), "`Inf`" (infinity) and "`NaN`" (the indefinite number or « Not-a-Number »). It is also useful to mention the existence of the functions "`is*`" allowing to detect the state of an entity (for example the previous three numbers can be tested with the functions "`isreal`", "`isinf`" ou "`isnan`"). To end, it is important to be aware of the fact that the accuracy of calculation is not infinite (it is easily observable with the function "`eps`").

15 Debugging tools

There is a debugger integrated into MATLAB. This tool allows to stop a program during its execution to examine the value of the various variables and detect possible errors. For this purpose, breakpoints must be created in advance in the file(s) `.m`. To create a breakpoint, it is enough to press on F12 and a red point appears to the left of the current line. Equivalently, we can click the dash in the left margin of the editor, or use the debugging toolbar.

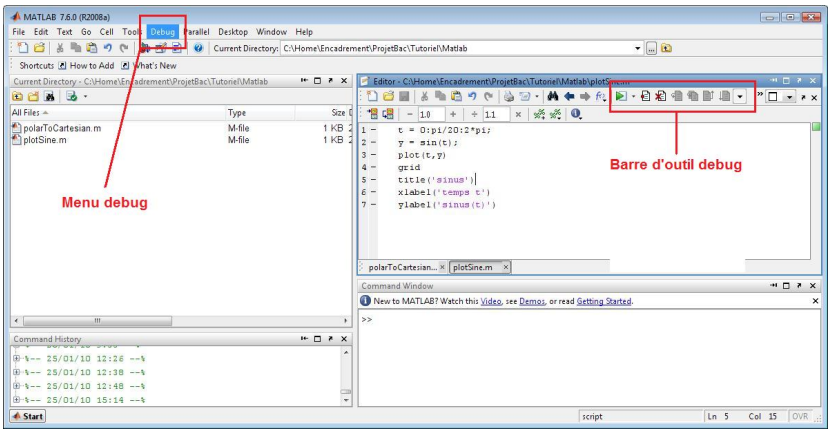


FIGURE 5 – Debugging tool : menu and toolbar

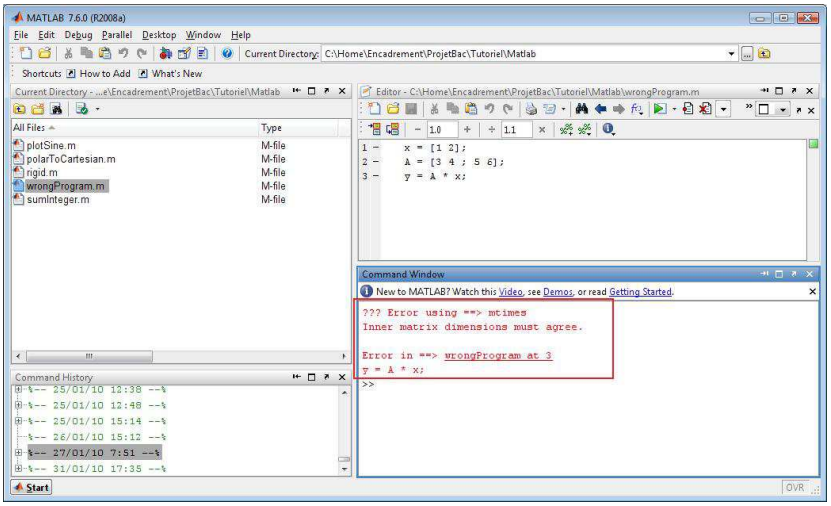


FIGURE 6 – Debugging tool during an error of execution

The execution of the program is then realised by pressing F5 or via the button Debug-> Run of the menu (see Fig. 5). The program stops at the breakpoint indicated by a green arrow. You can examine the contents of the variables of Workspace, move forward by a line, continue or stop the program. When a line calls a function, it is possible to go inside this function or to skip this function call and go directly to the following line.

Features of debugging are also accessible when a program produces an error. In the Fig. 6, the error message in red then indicates the line at which the error occurred. We can directly go to it by clicking the text "`wrongProgram at 3`".

16 Make codes readable

The MATLAB editor allows to shape the code automatically through "smart indenting". It allows to make the code more readable and easier to debug. Expressions such as if/end,

while/end or for/end are highlighted so as to see this part of the code as a single block (voir Fig. 7 and Fig. 8).

```
function no_indent
N=211;
M=211;
t=linspace(-5,100,N);
sigi=10;
for i=1:N-1
if (t(i)<0)
g(i) = 0;
else
g(i) = t(i)*exp(-t(i)/sigi);
end
end
gmax=max(g);
g=g/gmax;
plot(t(1:N-1),g);
xlabel('Time (s)')
axis tight
ylabel('Acceleration (m/s)');
end
```

FIGURE 7 – Code not formated with the smart indenting

```
function smart_indent
    N=211;
    M=211;
    t=linspace(-5,100,N);
    sigi=10;
    for i=1:N-1
        if (t(i)<0)
            g(i) = 0;
        else
            g(i) = t(i)*exp(-t(i)/sigi);
        end
    end
    gmax=max(g);
    g=g/gmax;
    plot(t(1:N-1),g);
    xlabel('Time (s)')
    axis tight
    ylabel('Acceleration (m/s)');
end
```

FIGURE 8 – Code formated with the smart indenting

There are several options for smart indenting, which you can find in the menu File -> Preferences -> Editor/Debugger -> Language, choose MATLAB as language, then modify the option Indenting. To use the smart indenting manually, select lines of code and press on CTRL+I.

The "\%" character allows to add comments which are not taken into account by the command interpreter. To increase the readability of your code, think of adding comments during important stages.

17 Tools of performances analysis

To test the performance of a code and possibly improve it, the user can take advantage of the profiler of MATLAB. It can be controlled via the "`profile`", or directly via its graphical interface (accessible by the command "`profile viewer`").

The profiler gives statistics for the number of calls and times spent during the execution of a code. This information allows to highlight the most computationally costly parts of the code. For the analysis of a very fast code, it can be useful to repeat this operation several times, to observe run times that are not negligible.

The functions "`tic`" and "`toc`" are also very practical to estimate the run time of a program section.

Exercise

Compare these two following programs :

```
tic;
for i=1:1000,
    for j=1:1000,
        x(i,j)=i+j;
    end
end
toc
```

and

```
tic;
x=zeros(1000);
for i=1:1000,
    for j=1:1000,
        x(i,j)=i+j;
    end
end
toc
```

How can you explain this result ?