

Lesson 1: INTRODUCTION TO MATLAB

2020

Giulia Comunale & Paolo Peruzzo

giulia.comunale@dicea.unipd.it

Download the student license of Matlab:

<https://www.csia.unipd.it/servizi/servizi-utenti-istituzionali/contratti-software-e-licenze/matlab>



The screenshot shows the CSIA Unipd website. The top navigation bar includes links for Unipd.it, Contatti, Accesso, CHIEDI@helpdesk, Webmail, Servizi Online, and SIT. The main header features the University of Padua logo, the text "SERVIZI INFORMATICI E TELEMATICI", and a search bar. Below the header is a navigation menu with "Chi siamo", "Servizi", "Attività", and "Progetti". The breadcrumb trail reads: Home / Servizi / Servizi per Utenti Istituzionali / Contratti Software e Licenze / MATLAB. The left sidebar contains a "Servizi" menu with items: Servizi per Utenti Istituzionali, Contratti Software e Licenze (with sub-items: Adobe, ArcGIS ESRI, DNA Star, LabVIEW e Multisim, MATLAB, and Microsoft Campus), and a "MATLAB" section with sub-items: MATLAB per tecnici informatici and Microsoft Campus. The main content area is titled "MATLAB" and contains the following text:

Il Centro Servizi Informatici ha firmato un accordo annuale per la Total Academic Headcount -licenza di tipo Campus e Student- che consente a tutto lo staff, ricercatori e studenti di utilizzare il software fornito da Mathworks. Il precedente contratto di manutenzione è stato congelato al 2016 e sarà ripristinato nel momento in cui si dovesse interrompere o giungere a termine la licenza TAH.

The Mathworks: Total Academic Headcount (TAH)

La licenza TAH consente a professori, studenti e altri membri dell'università di accedere al software attraverso computer di proprietà dell'università stessa, in tutta la struttura. I professori, gli altri membri dell'istituzione e gli studenti possono inoltre installare il software MathWorks sui propri computer personali. Le licenze TAH supportano tre configurazioni: Campus, Student e Accesso Simultaneo per le attrezzature di calcolo dell'università (riservata ai tecnici informatici).

Download the student license of Matlab:

<https://www.csia.unipd.it/servizi/servizi-utenti-istituzionali/contratti-software-e-licenze/matlab>

Download del software e codici - Campus e Student

- 1- Reperire il Codice di Attivazione sul sito <https://software.unipd.it/> (per accedere al sito è necessario utilizzare il proprio account personale @unipd.it oppure @studenti.unipd.it)
- 2- Creare un account Mathworks cliccando "Create Account" sul sito <https://it.mathworks.com/login> e utilizzando come e-mail address la propria casella @unipd.it oppure @studenti.unipd.it oppure @phd.unipd.it (si consiglia invece di non utilizzare la stessa password).
- 3- Una volta autenticati sul portale Mathworks cliccare sul nome del profilo in alto a destra, selezionare Collega licenza ed inserire il Codice di Attivazione di cui al punto 1.
- 4- Fare clic su "Scarica i tuoi prodotti ora" oppure accedere a mathworks.com/downloads
- 5- Fare clic sul pulsante "scarica" per ottenere la versione più recente.
- 6- Scegliere una piattaforma supportata e scaricare il programma di installazione.

Installazione del software - Campus e Student

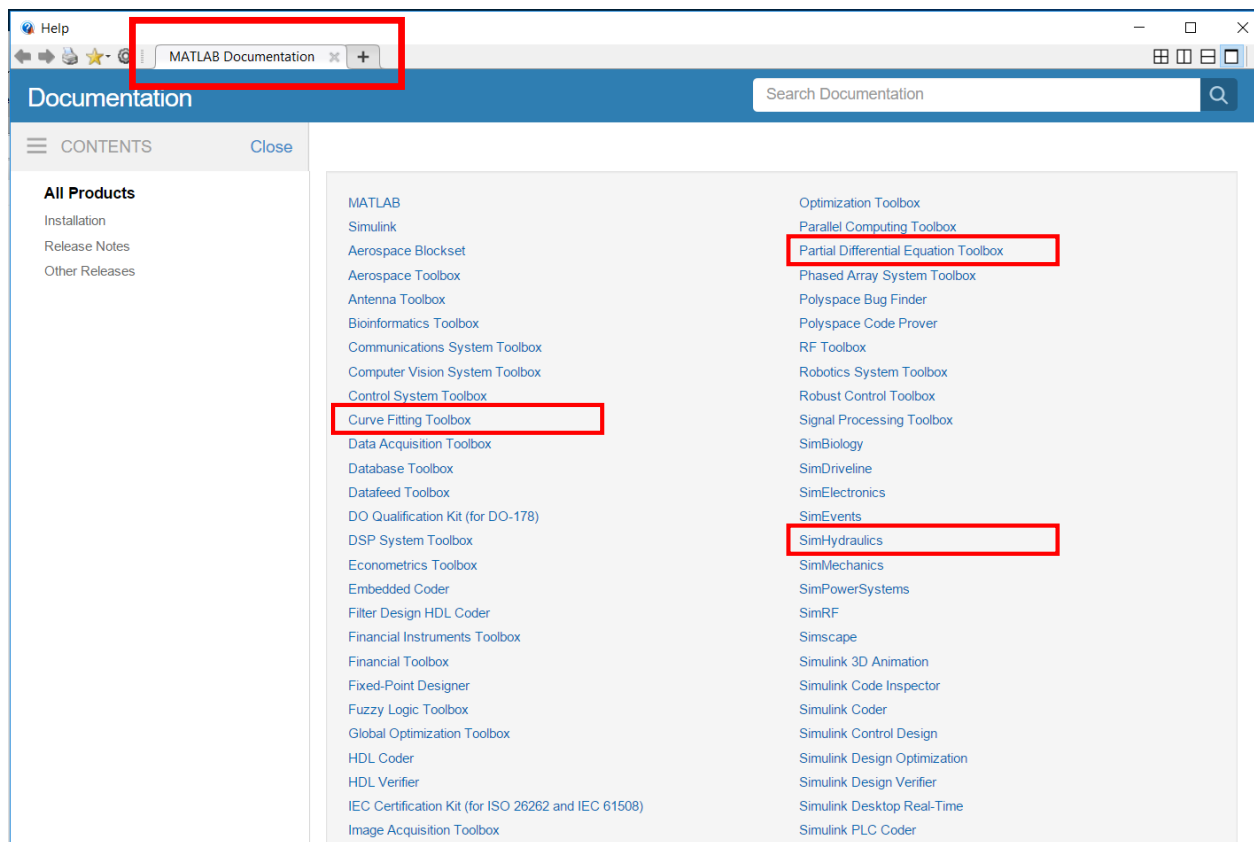
- 1- Avviare il programma di installazione.
- 2- Nel programma di installazione, selezionare Accedi con un account MathWorks e seguire le istruzioni online.
- 3- Quando richiesto, selezionare la licenza Academic – Total Headcount contrassegnata con Campus oppure Student.
- 4- Selezionare i prodotti che si desidera scaricare e installare.
- 5- Dopo aver scaricato e installato i prodotti desiderati, mantenere selezionata la casella Attiva MATLAB e fare clic su Avanti.
- 6- Quando viene richiesto di fornire un nome utente, verificare che il nome utente visualizzato sia corretto. Continuare con la procedura fino al completamento dell'attivazione.

MATLAB (**MAT**rix **LAB**oratory) is a numerical computing environment.

MATLAB allows:

- Matrix manipulations;
- Plotting of functions and data;
- Implementation of algorithms;
- Interfacing with programs written in other languages, including C, C++, Java, Fortran and Python;
- Specialized toolboxes for making things easier (**Matlab has several toolboxes**);

- Solve directly systems of equations with implemented functions;
- Solve Ordinary Differential Equations (ODEs);
- Solve Partial Differential Equations (PDEs);
- Elaborate data from file Text, Excel, etc.;
- Elaborate images, generate images, tables, 2D and 3D plot;
- Create functions.



<https://it.mathworks.com/help/matlab/>

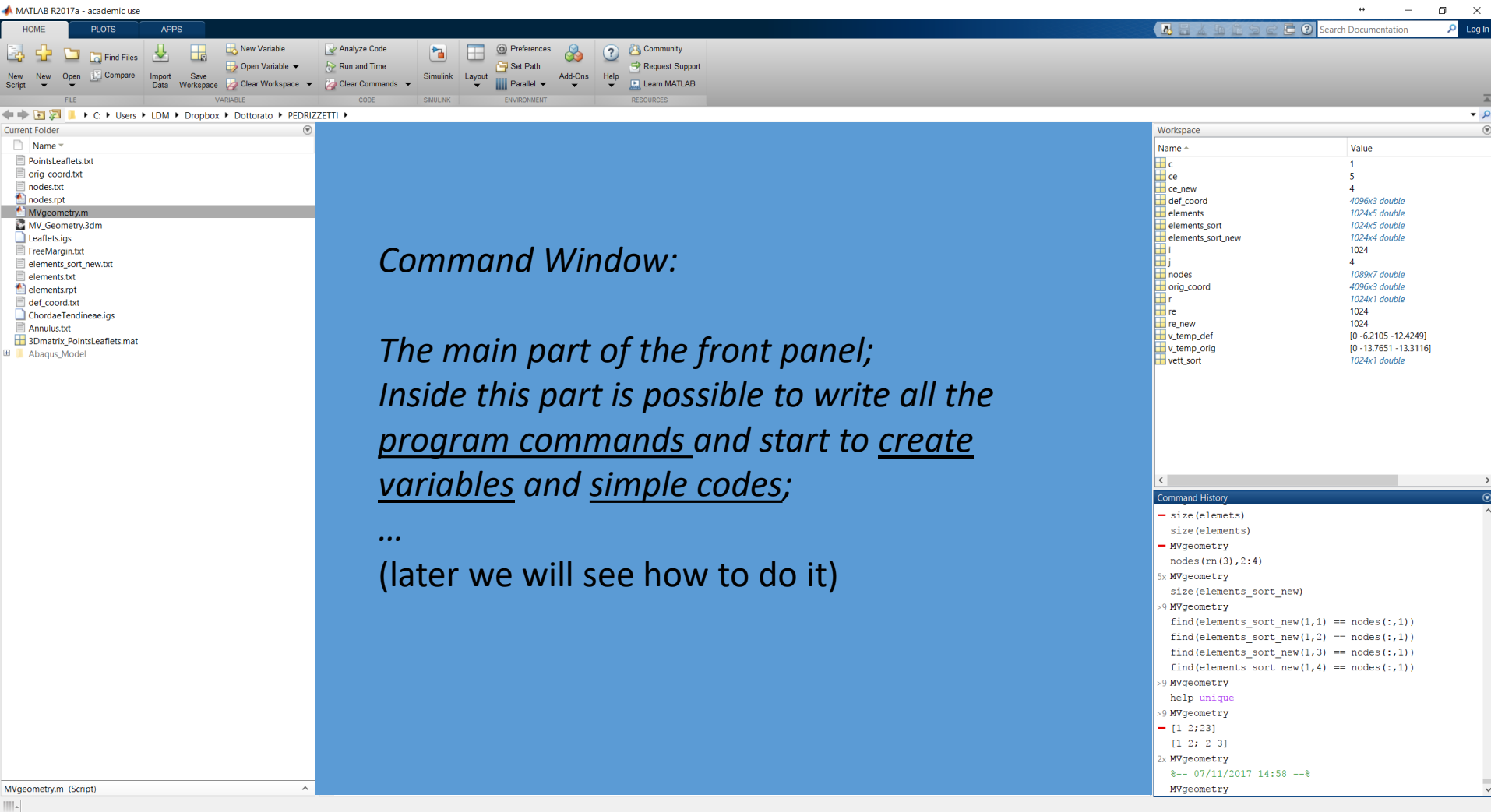
Control Bar

*Current
Folder*

*Command
Window*

Workspace

*Command
History*



The image shows the MATLAB R2017a - academic use interface. The main window is titled "MATLAB R2017a - academic use". The top menu bar includes HOME, PLOTS, and APPS. The top toolbar contains icons for New Script, New Open, Find Files, Import Data, Save Workspace, Open Variable, Analyze Code, Run and Time, Clear Commands, Simulink, Layout, Set Path, Add-Ons, Help, Request Support, and Learn MATLAB. The left sidebar shows the Current Folder and the Workspace. The Command Window is open, displaying the following code:

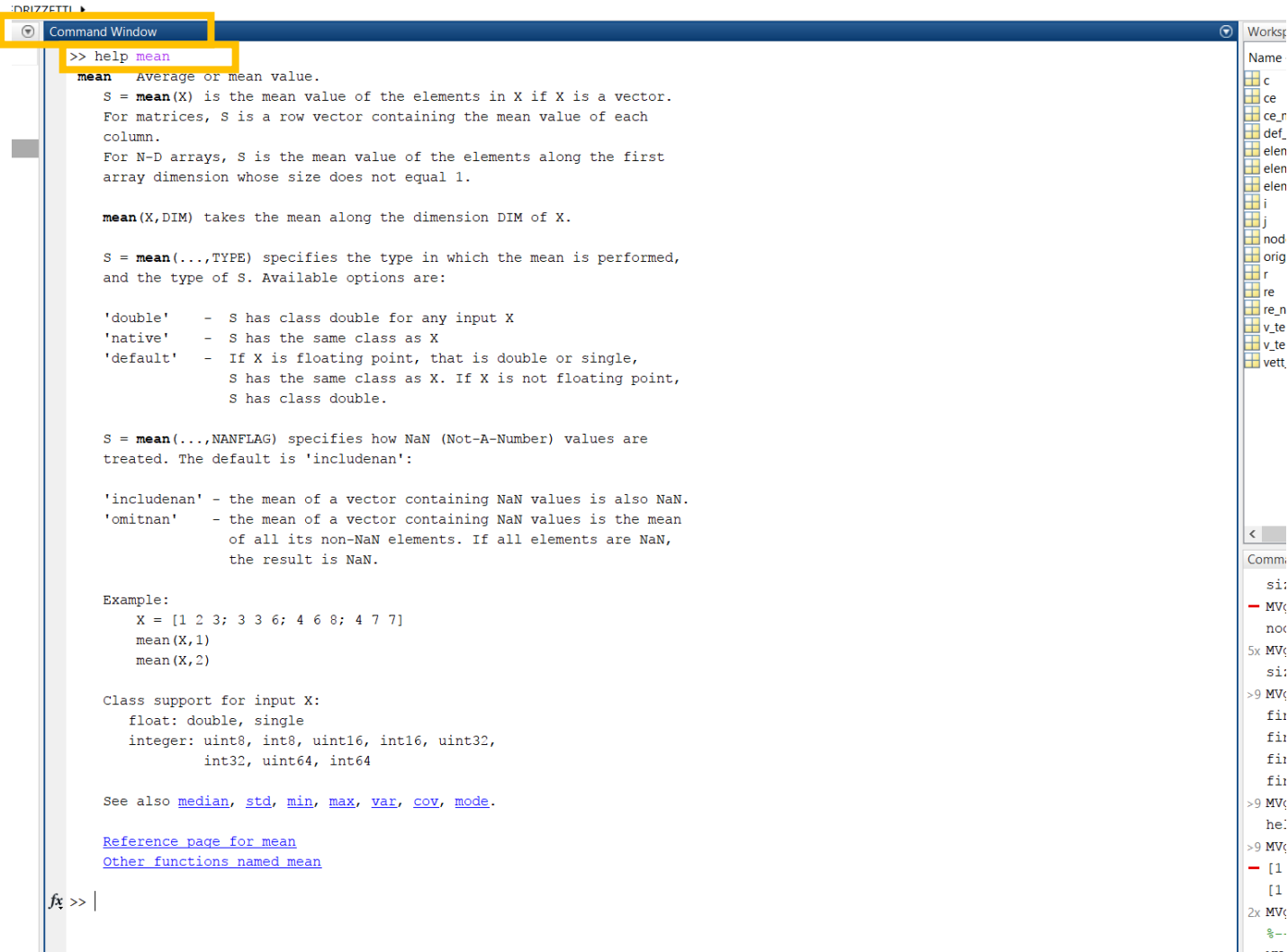
```

size(elements)
size(elements)
MVGeometry
nodes(rn(3),2:4)
5x MVGeometry
size(elements_sort_new)
>9 MVGeometry
find(elements_sort_new(1,1) == nodes(:,1))
find(elements_sort_new(1,2) == nodes(:,1))
find(elements_sort_new(1,3) == nodes(:,1))
find(elements_sort_new(1,4) == nodes(:,1))
>9 MVGeometry
help unique
>9 MVGeometry
[1 2:23]
[1 2: 2 3]
2x MVGeometry
%-- 07/11/2017 14:58 --%
MVGeometry
    
```

The Command Window output shows the following results:

Name	Value
c	1
ce	5
ce_new	4
def_coord	4096x3 double
elements	1024x5 double
elements_sort	1024x5 double
elements_sort_new	1024x4 double
i	1024
j	4
nodes	1089x7 double
orig_coord	4096x3 double
r	1024x1 double
re	1024
re_new	1024
v_temp_def	[0 -6.2105 -12.4249]
v_temp_orig	[0 -13.7651 -13.3116]
vett_sort	1024x1 double

...
(later we will see how to do it)



```
>> help mean

mean Average or mean value.

S = mean(X) is the mean value of the elements in X if X is a vector.
For matrices, S is a row vector containing the mean value of each
column.
For N-D arrays, S is the mean value of the elements along the first
array dimension whose size does not equal 1.

mean(X,DIM) takes the mean along the dimension DIM of X.

S = mean(...,TYPE) specifies the type in which the mean is performed,
and the type of S. Available options are:

'double' - S has class double for any input X
'native' - S has the same class as X
'default' - If X is floating point, that is double or single,
            S has the same class as X. If X is not floating point,
            S has class double.

S = mean(...,NANFLAG) specifies how NaN (Not-A-Number) values are
treated. The default is 'includenan':

'includenan' - the mean of a vector containing NaN values is also NaN.
'omitnan' - the mean of a vector containing NaN values is the mean
            of all its non-NaN elements. If all elements are NaN,
            the result is NaN.

Example:
    X = [1 2 3; 3 3 6; 4 6 8; 4 7 7]
    mean(X,1)
    mean(X,2)

Class support for input X:
    float: double, single
    integer: uint8, int8, uint16, int16, uint32,
            int32, uint64, int64

See also median, std, min, max, var, cov, mode.

Reference page for mean
Other functions named mean
```

Workspace:

Name
c
ce
ce_n
def
elen
elen
elen
i
j
nod
orig
r
re
re_n
v_te
v_te
vett

Command Window:

```
si:
- MV
noc
5x MV
si:
>9 MV
fir
fir
fir
fir
>9 MV
he:
>9 MV
[1
[1
2x MV
%--
MATLAB
```

Command Window

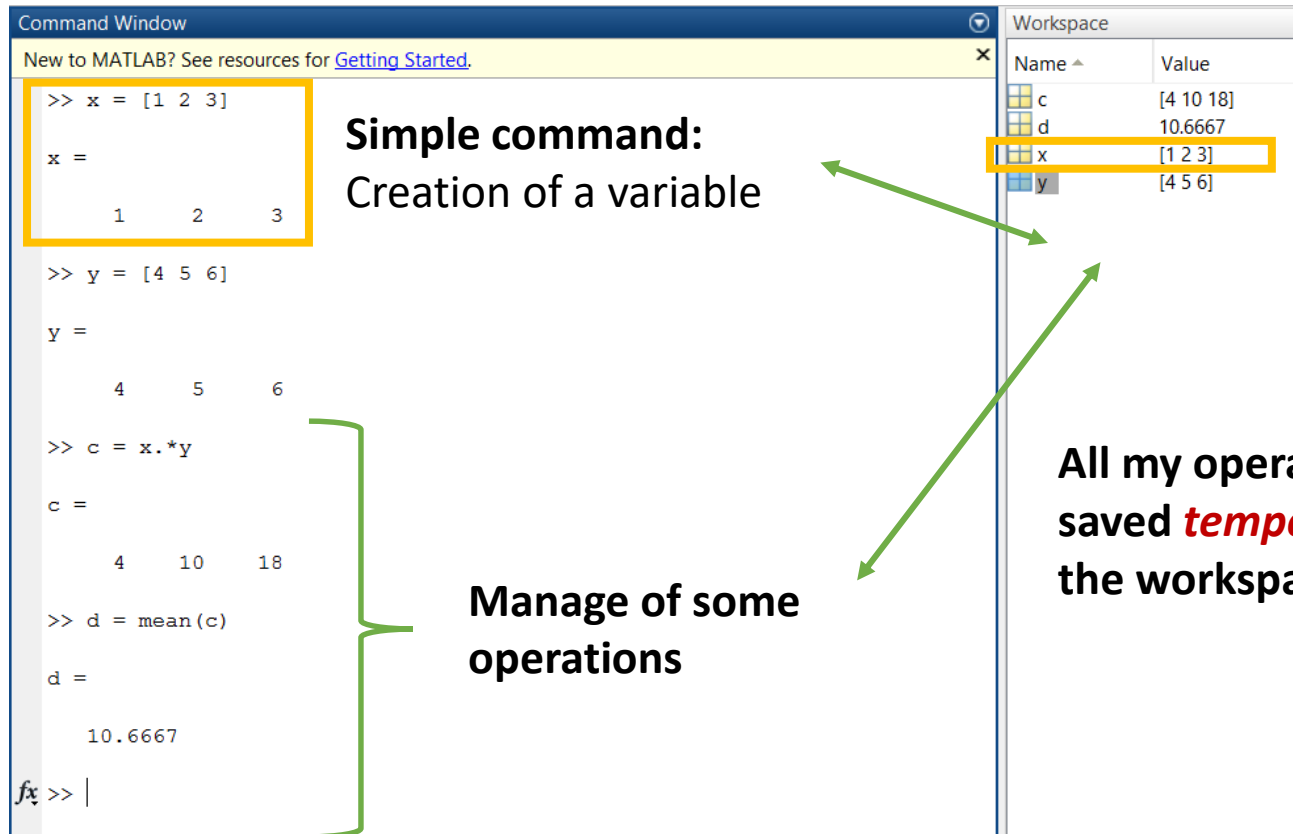
```
Error using load  
Unable to read file 'forza_Napoli.txt'. No such file or directory.  
  
Error in MVgeometry (line 110)  
load('forza_Napoli.txt')
```

J >> |

Work

Nam

c
ce
ce
de
el
el
el
i
j
nc
or
r
re
re.
v.
v.
ve



Command Window

New to MATLAB? See resources for [Getting Started.](#)

```
>> x = [1 2 3]
x =
     1     2     3

>> y = [4 5 6]
y =
     4     5     6

>> c = x.*y
c =
     4    10    18

>> d = mean(c)
d =
    10.6667

fx >> |
```

Simple command:
Creation of a variable

Manage of some operations

Workspace

Name	Value
c	[4 10 18]
d	10.6667
x	[1 2 3]
y	[4 5 6]

All my operators are saved **temporarily** in the workspace

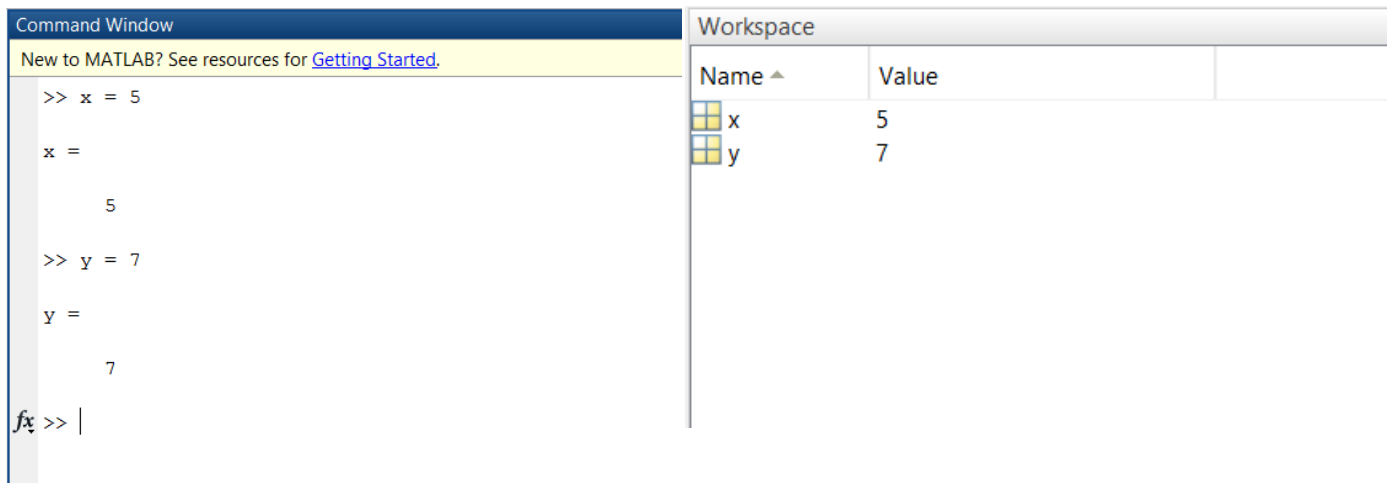
All variables are created with double precision unless specified and **they are matrices**.

- Example:

```
>> x = 5;
```

```
>> y = 7;
```

After these statements, the **variables** are 1x1 matrices with double precision



The screenshot shows the MATLAB Command Window and Workspace. The Command Window displays the commands and their outputs:

```
>> x = 5
x =
     5
>> y = 7
y =
     7
fx >> |
```

The Workspace window shows the variables created:

Name	Value
x	5
y	7

- a Row Vector (matrix [1x4])

$x = [1 \ 2 \ 5 \ 1]$

- a Column Vector (matrix [4x1])

$x = [1; 2; 5; 1]$

- a Matrix [3x3]

$A = [1 \ 2 \ 3; 5 \ 1 \ 4; 3 \ 2 \ -1]$

- Transpose

$a = [1 \ 2 \ 5 \ 1] \ [1 \times 4]$

$b = a' \quad [4 \times 1]$

```

Command Window
>> x = [1 2 5 1]

x =

     1     2     5     1

>> x = [1;2;5;2]

x =

     1
     2
     5
     2

>> A = [1 2 3;5 1 4;3 2 -1]

A =

     1     2     3
     5     1     4
     3     2    -1

>> a = [1 2 5 2]

a =

     1     2     5     2

>> b = a'

b =

     1
     2
     5
     2

fx >> |

```

- **t = 1:10** → all the values spaced 1 (default)
- **k = 2:-0.5:-1** → change in spacing
- **B = [1:4; 5:8]** → build a [2x4] matrix
- **zeros(M,N)** [MxN] matrix of zeros
x = zeros(1,3)
- **ones(M,N)** [MxN] matrix of ones
y = ones(1,3)

```
Command Window

>> t = 1:10

t =

     1     2     3     4     5     6     7     8     9    10

>> k = 2:-0.5:-1

k =

    2.0000    1.5000    1.0000    0.5000         0   -0.5000   -1.0000

>> B = [1:4; 5:8]

B =

     1     2     3     4
     5     6     7     8

>> x = zeros(1,3)

x =

     0     0     0

>> y = ones(1,3)

y =

     1     1     1

fx >> |
```

Given the Matrix A[3X3]:

A =

3	5	3
6	8	2
2	7	3

Two points with extremes = all the values from the first extreme to the last extreme

Two points without extremes = all the columns of my variable

```
>> A(6)
```

```
ans =
```

7

```
>> A(3,2)
```

```
ans =
```

7

```
>> A(2,:)
↓
```

```
ans =
```

6	8	2
---	---	---

```
>> A(1:2,2)
↓
```

```
ans =
```

5
8

>> A = [1 2 3; 4 5 6; 7 8 9]

>> B = [3 5 2; 5 2 8; 3 6 9]

A =

1	2	3
4	5	6
7	8	9

B =

3	5	2
5	2	8
3	6	9

Addition

>> X = A + B

X =

4	7	5
9	7	14
10	14	18

Subtraction

>> Y = A - B

Y =

-2	-3	1
-1	3	-2
4	2	0

Transpose

>> T = A'

T =

1	4	7
2	5	8
3	6	9


```
>> A = [1 2 3;4 5 6;7 8 9]
```

```
A =
```

```
1 2 3
4 5 6
7 8 9
```

Command Window

New to MATLAB? See resources for [Getting Started](#).

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
1 2 3
4 5 6
7 8 9
```

```
>> size(A)
```

```
ans =
```

```
3 3
```

Size of a Matrix

```
fx >> |
```

```
>> x = [1 2 3]
```

```
x =
```

```
1 2 3
```

```
>> size(x)
```

```
ans =
```

```
1 3
```

```
>> y = x'
```

```
y =
```

```
1
2
3
```

```
>> size(y)
```

```
ans =
```

```
3 1
```

Size of
a vector

$$A = [1 \ 2 \ 3; 5 \ 1 \ 4; 3 \ 2 \ 1]$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 1 & 4 \\ 3 & 2 & -1 \end{bmatrix}$$

$$\mathbf{x} = \mathbf{A}(1,:)$$

$$\mathbf{y} = \mathbf{A}(3,:)$$

$$\mathbf{x} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 3 & 2 & -1 \end{bmatrix}$$

$\mathbf{.*}$: element-by-element multiplication $\mathbf{b} = \mathbf{x} \mathbf{.*} \mathbf{y}$

$$\mathbf{b} = \begin{bmatrix} 3 & 4 & -3 \end{bmatrix}$$

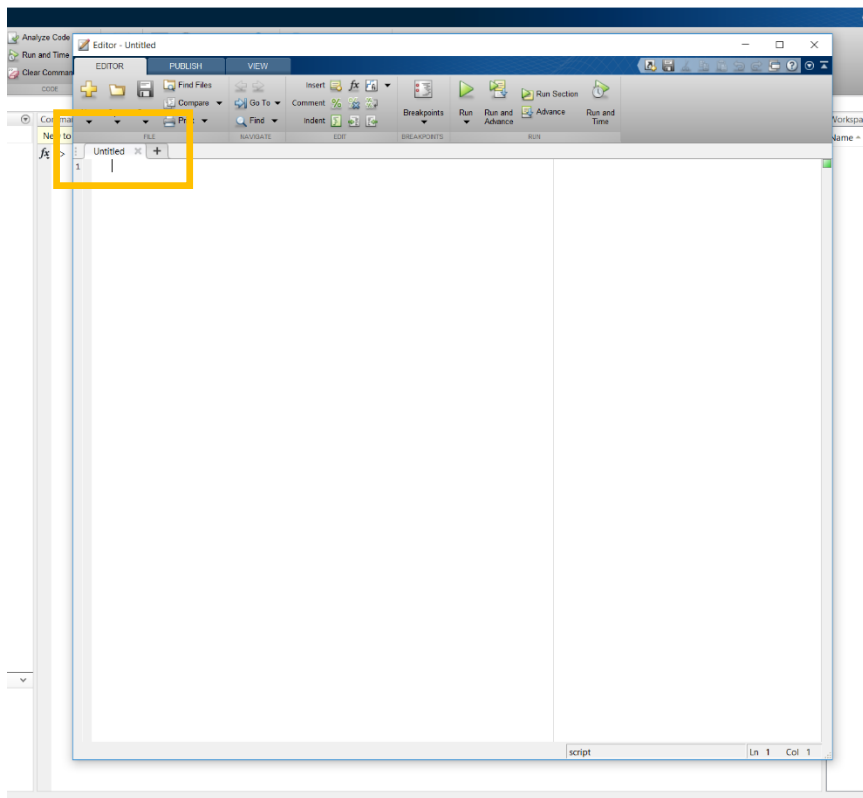
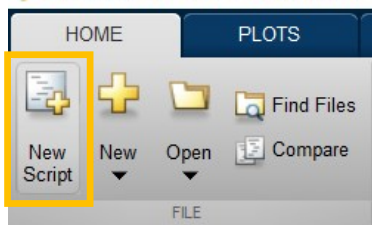
$\mathbf{./}$: element-by-element division $\mathbf{c} = \mathbf{x} \mathbf{./} \mathbf{y}$

$$\mathbf{c} = \begin{bmatrix} 0.33 & 1 & -3 \end{bmatrix}$$

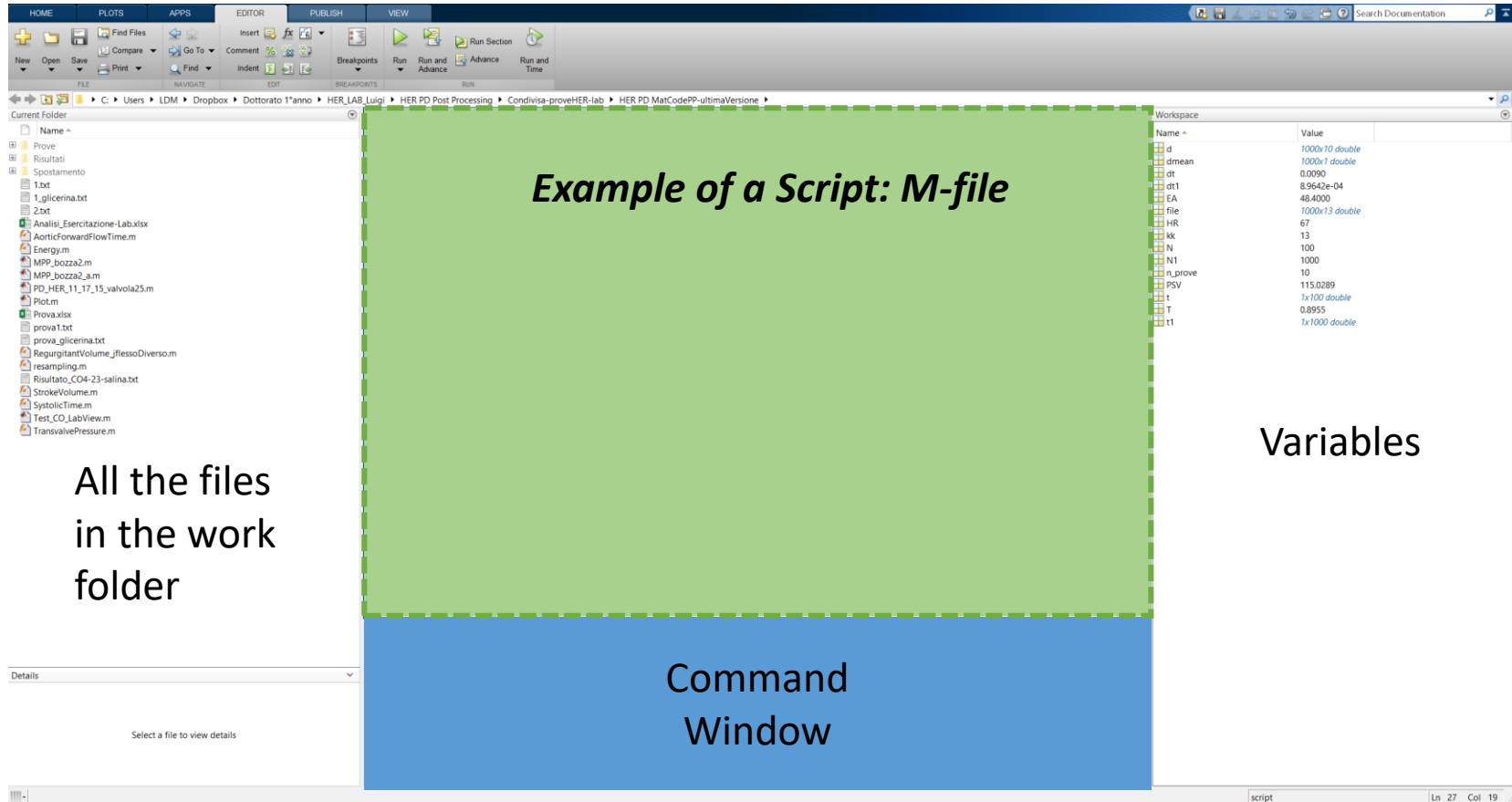
$\mathbf{.^}$: element-by-element power $\mathbf{d} = \mathbf{x} \mathbf{.^} 2$

$$\mathbf{d} = \begin{bmatrix} 1 & 4 & 9 \end{bmatrix}$$

MATLAB R2015b - academic use



- An m-file is a **text file** used by MATLAB with extension **“.m”**
- It can store a script or an individual function in the MATLAB language
- M files are used for executing algorithms, plotting graphs, and performing other mathematical operations.



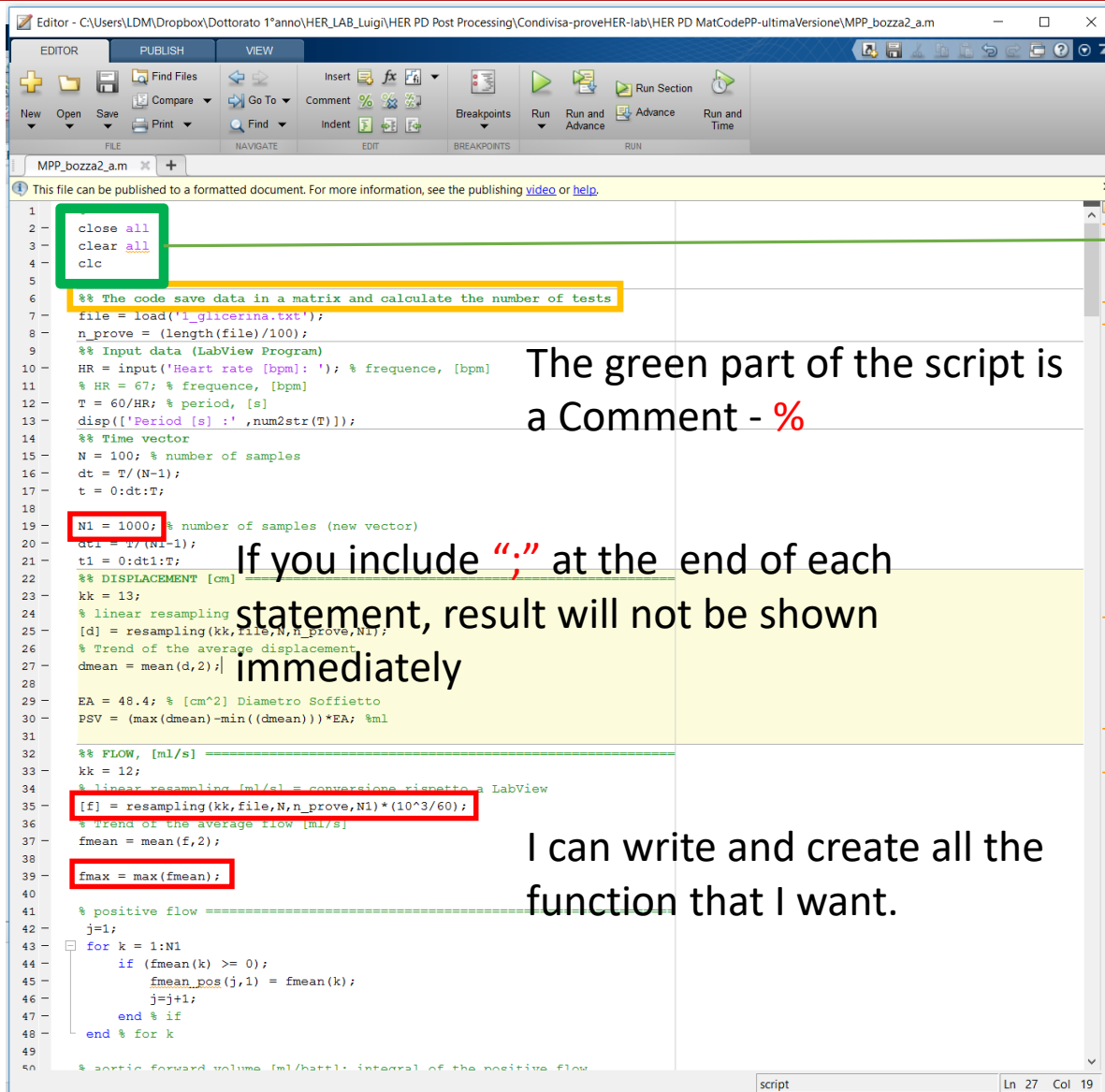
Example of a Script: M-file

Variables

Name	Value
d	1000x10 double
dmean	1000x1 double
dt	0.0090
dt1	8.9642e-04
EA	48.4000
file	1000x13 double
HR	67
kk	13
N	100
N1	1000
p.prove	10
PSV	115.0289
T	1x100 double
T	0.8955
t1	1x1000 double

Command Window

All the files in the work folder



```

1
2 close all
3 clear all
4 clc
5
6 %% The code save data in a matrix and calculate the number of tests
7 file = load('1_glicerina.txt');
8 n_prove = (length(file)/100);
9 %% Input data (LabView Program)
10 HR = input('Heart rate [bpm]: '); % frequency, [bpm]
11 % HR = 67; % frequency, [bpm]
12 T = 60/HR; % period, [s]
13 disp(['Period [s] :', num2str(T)]);
14 %% Time vector
15 N = 100; % number of samples
16 dt = T/(N-1);
17 t = 0:dt:T;
18
19 N1 = 1000; % number of samples (new vector)
20 dt1 = T/(N1-1);
21 t1 = 0:dt1:T;
22 %% DISPLACEMENT [cm]
23 kk = 13;
24 % linear resampling
25 [d] = resampling(kk, file, N, n_prove, N1);
26 % Trend of the average displacement
27 dmean = mean(d,2);
28
29 EA = 48.4; % [cm^2] Diametro Soffietto
30 PSV = (max(dmean)-min(dmean))*EA; %ml
31
32 %% FLOW, [ml/s]
33 kk = 12;
34 % linear resampling [ml/s] = conversione rianetto a LabView
35 [f] = resampling(kk, file, N, n_prove, N1) * (10^3/60);
36 % Trend of the average flow [ml/s]
37 fmean = mean(f,2);
38
39 fmax = max(fmean);
40
41 % positive flow =====
42 j=1;
43 for k = 1:N1
44     if (fmean(k) >= 0);
45         fmean_pos(j,1) = fmean(k);
46         j=j+1;
47     end % if
48 end % for k
49
50 %ortic forward volume (ml/batt1): integral of the positive flow

```

The green part of the script is
a Comment - %

If you include ";" at the end of each
statement, result will not be shown
immediately

I can write and create all the
function that I want.

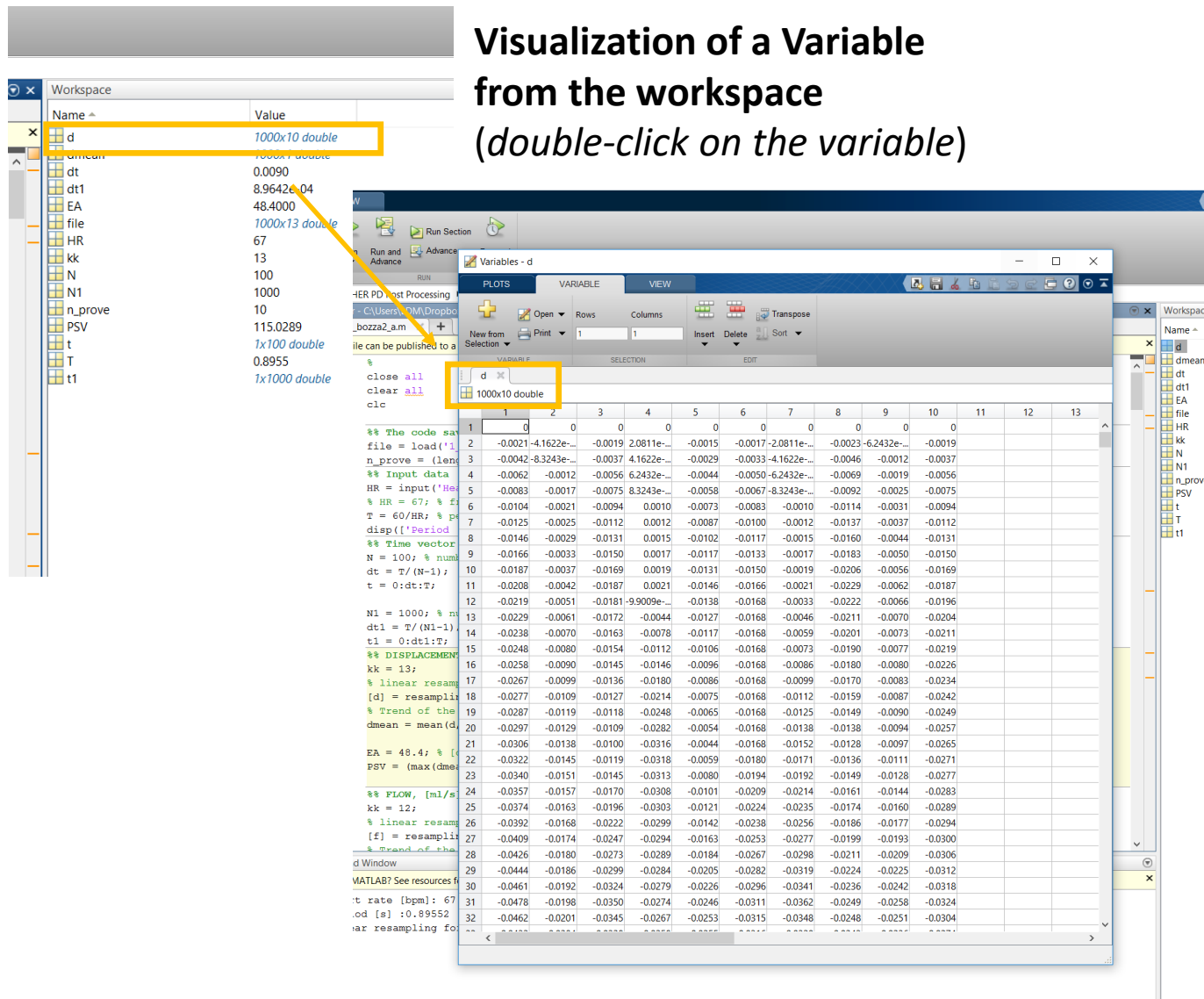
Commands:

Close all: closes all the
open figure windows

Clear all: removes all
variables from the
workspace

clc: Clears the command
window

Visualization of a Variable from the workspace (double-click on the variable)



The screenshot shows the MATLAB environment. On the left, the **Workspace** pane lists variables: `d` (1000x10 double), `dt` (0.0090), `dt1` (8.9642e-04), `EA` (48.4000), `file` (1000x13 double), `HR` (67), `kk` (13), `N` (100), `N1` (1000), `n_prove` (10), `PSV` (115.0289), `t` (1x100 double), `T` (0.8955), and `t1` (1x1000 double). The variable `d` is highlighted with a yellow box.

On the right, the **Variables - d** window displays the data for variable `d`. It shows a grid with 10 columns and 1000 rows. The first row contains values: 1, 0, 0, 0, 0, 0, 0, 0, 0, 0. The subsequent rows contain numerical values ranging from approximately -0.0462 to 0.0019.

In the background, the MATLAB script editor shows the following code:

```
%
close all
clear all
clc

%% The code say
file = load('1...');
n_prove = (length(file));
%% Input data
HR = input('HR = '); % HR = 67; % Period
T = 60/HR; % Period
disp(['Period = ', num2str(T), ' s'])
%% Time vector
N = 100; % number of samples
dt = T/(N-1);
t = 0:dt:T;

N1 = 1000; % number of samples
dt1 = T/(N1-1);
t1 = 0:dt1:T;

%% DISPLACEMENT
kk = 13;
% linear resampling
[d] = resample(file, N1, N);
% Trend of the displacement
dmean = mean(d, 2);

EA = 48.4; % Elastic modulus
PSV = (max(dmean));

%% FLOW, [ml/s]
kk = 12;
% linear resampling
[f] = resample(file, N1, N);
% Trend of the flow
fmean = mean(f, 2);

MATLAB See resources for more information
t rate [bpm]: 67
od [s]: 0.8955
var resampling factor: 10
```

Exercise 1: Matrix Operations

- Create a folder (in the desktop or in your Pendrive) : **matlab_introduction**;
- Run MatLab;
- **Open a new script**;
- Create a comment (%) with the phrase: **Lesson 1 – Exercise 1: Introduction To Matlab**
- Save the script (file.m) – **exercise_1.m**, inside the folder **matlab_introduction**;
- Use the command: **clear all, close all, clc**;
- *Use comments for describe the different operations*;
- Create two matrices **A [3x3]**, **B [5x3]**;
- Extract two vectors: **x [1x3]** from A; **y [3x1]** from B - (**$x = A(\text{first row})$** ; **$y = B(\text{first three row, second column})$**)
- Calculate the element-by-element multiplication between x and y; (call the result **a**)
- Calculate the element-by-element multiplication between x and y' (transpose), (call the result **a_1**);
- *Check the results, which is the difference between these two operations?*
- Calculate the element-by-element division between x and y' (transpose), (call the result **b**);
- Calculate the element-by-element power of x and y (call the results **q** = x^2 and **w** = y^2);
- Matrices Operations: Extract a sub-matrix **D [3x3]** from B: (**$D(\text{second, third and last row of B})$**)
- Calculate the addition between the matrix A and D (call the result **C**);
- Calculate the subtraction between A and D (call the result **E**);

A =		
1	7	5
4	6	10
-6	9	15

B =		
-3	8	20
12	8	2
0	2	1
3	4	5
6	12	-6

```
Editor - C:\Users\LDM\Google Drive\ENVIRONMENTAL FLUID MECHANICS - 2016_2017\Lesson1-MatLab-14_11_2016\matlab-introduction\lesson1.m
lesson1.m x +
1 % lesson 1: matrices operations
2 clc
3 clear all
4 close all
5
6 % Create two matrices A [3x3], B[5x3]
7 A = [1 7 5; 4 6 10; -6 9 15]
8 B = [-3 8 20; 12 8 2; 0 2 1; 3 4 5; 6 12 -6]
9
10 % Extract two vectors: x [1x3] from A; y [3x1] from B
11 x = A(1,:);
12 y = B(1:3,2);
13
14 % element-by-element multiplication a = x*y;
15 a = x.*y;
16
17 % element-by-element multiplication a = x*y;
18 a_1 = x.*y';
19
20 % element-by-element division b = x/y;
21 b = x./y';
22
23 % element-by-element power q = x^2, w = y^2,
24 q = x.^2;
25 w = y.^2;
26
27 % Matrices Operations: =====
28 % Extract a sub-matrix D [3x3] from B
29 D = [B(2,1:3);B(3,1:3);B(5,1:3)]
30
31 % addition C = A+B
32 C = A +D
33
34 % subtraction D = A-B;
35 E = A-D
36
37
38
```


- **plot(Y)**: plots the columns of Y versus their index.
- **plot(X,Y)**: plots vector Y versus vector X.
- **plot(X,Y,S)** : Various line types, plot symbols, and colors may be obtained with plot(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:

Color

b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

Marker

.	point
o	circle
x	x-mark
+	plus
*	star
s	square
d	diamond
v	triangle (down)
^	triangle (up)
<	triangle (left)
>	triangle (right)
p	pentagram
h	hexagram

Line style

-	solid
:	dotted
-.	dashdot
--	dashed
(none)	no line

Examples:

`plot(X,Y,'r+:')`: plots a red dotted line with a plus at each data point;

`plot(X,Y,'bd')`: plots blue diamond at each data point but does not draw any line.

- **plot(Y)**: plots the columns of Y versus their index.
- **plot(X,Y)**: plots vector Y versus vector X.
- **plot(X,Y,S)** : Various line types, plot symbols and colors may be obtained with plot(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:
- **plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)** combines the plots defined by the (X,Y,S) triples.

Example:

plot(X1,Y1,'y-',X2,Y2,'go') plots the two datasets, with a solid yellow line interpolating green circles at the data points.

- **plot(Y)**: plots the columns of Y versus their index.
- **plot(X,Y)**: plots vector Y versus vector X.
- **plot(X,Y,S)** : Various line types, plot symbols and colors may be obtained with plot(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:
- **plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)** combines the plots defined by the (X,Y,S) triples.
- The X,Y pairs, or X,Y,S triples, can be followed by parameter/value pairs to specify **additional properties of the lines**.

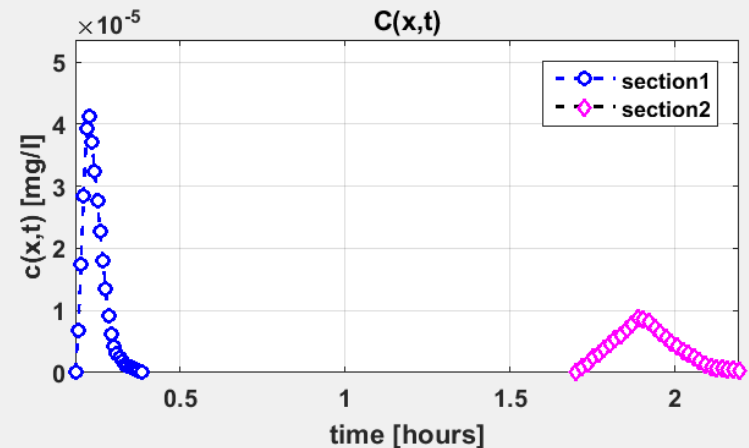
Examples:

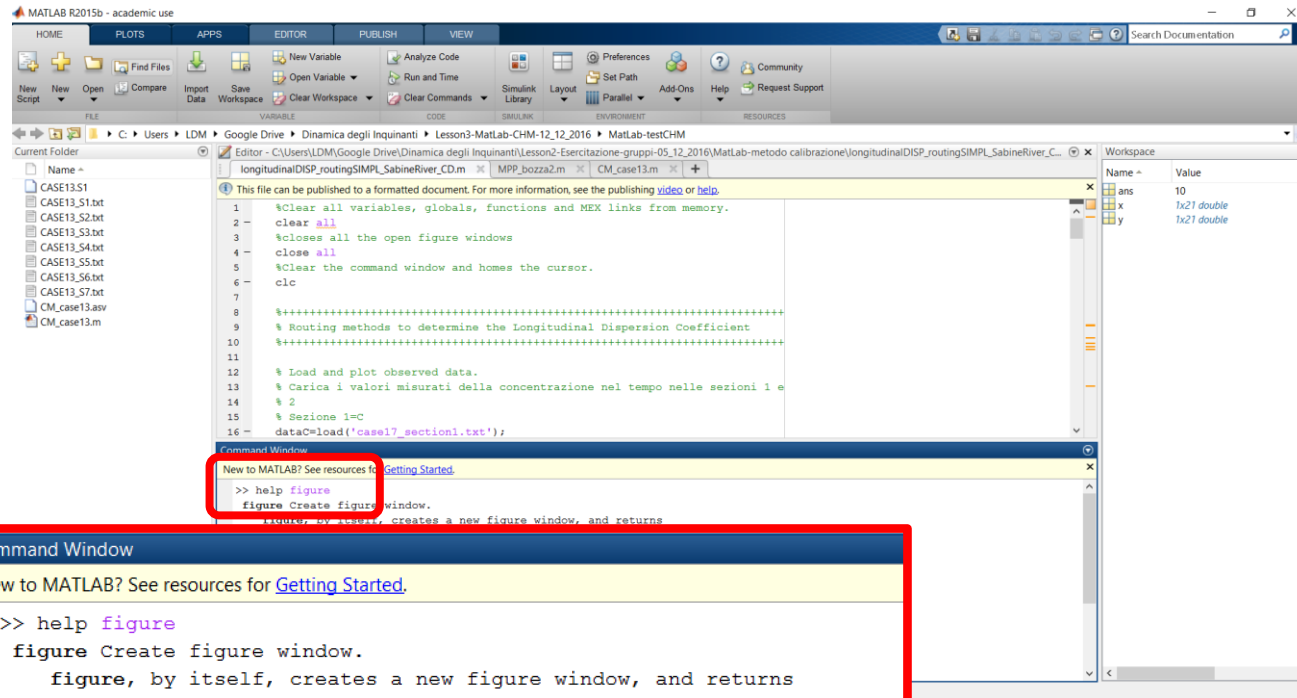
```
plot(x,y,'--rs','LineWidth',0.5,'MarkerEdgeColor','k',  
'MarkerFaceColor','g','MarkerSize',2)
```

```

46 - figure_1 = figure();
47 - hold on
48 - %Plot Observed data
49 - plot(t_dataC,C_dataC,'--b','linewidth',1.5,...
50 -     'Marker','o','MarkerSize',6,...
51 -     'MarkerEdgeColor',[0 0 1],...
52 -     'MarkerFaceColor',[1 1 1]);
53 - plot(t_dataD,C_dataD,'--k','linewidth',1.5,...
54 -     'Marker','d','MarkerSize',6,...
55 -     'MarkerEdgeColor',[1 0 1],...
56 -     'MarkerFaceColor',[1 1 1]);
57 - title('C(x,t)','FontSize',12,'FontWeight','bold')
58 - axis([min(t_dataC) max(t_dataD) 0 max(C_dataC)])
59 - xlabel('time [hours]','FontSize',12,'FontWeight','bold');
60 - ylabel('c(x,t) [mg/l]','FontSize',12,'FontWeight','bold');
61 - set(gca,'FontSize',12,'FontWeight','bold');
62 - legend('section1','section2')
63 - box on
64 - grid on
65 - % save image
66 - set(gcf, 'PaperUnits', 'centimeters');
67 - set(gcf, 'PaperSize', [10 10]);|
68 - set(gcf, 'PaperPositionMode','manual');
69 - set(gcf, 'PaperUnits','normalized');
70 - set(gcf, 'PaperPosition',[0.1 0.1 2 2]);
71 - fig = 'figure_1';
72 - print('-dpng',figure_1,fig);
73

```





Command Window

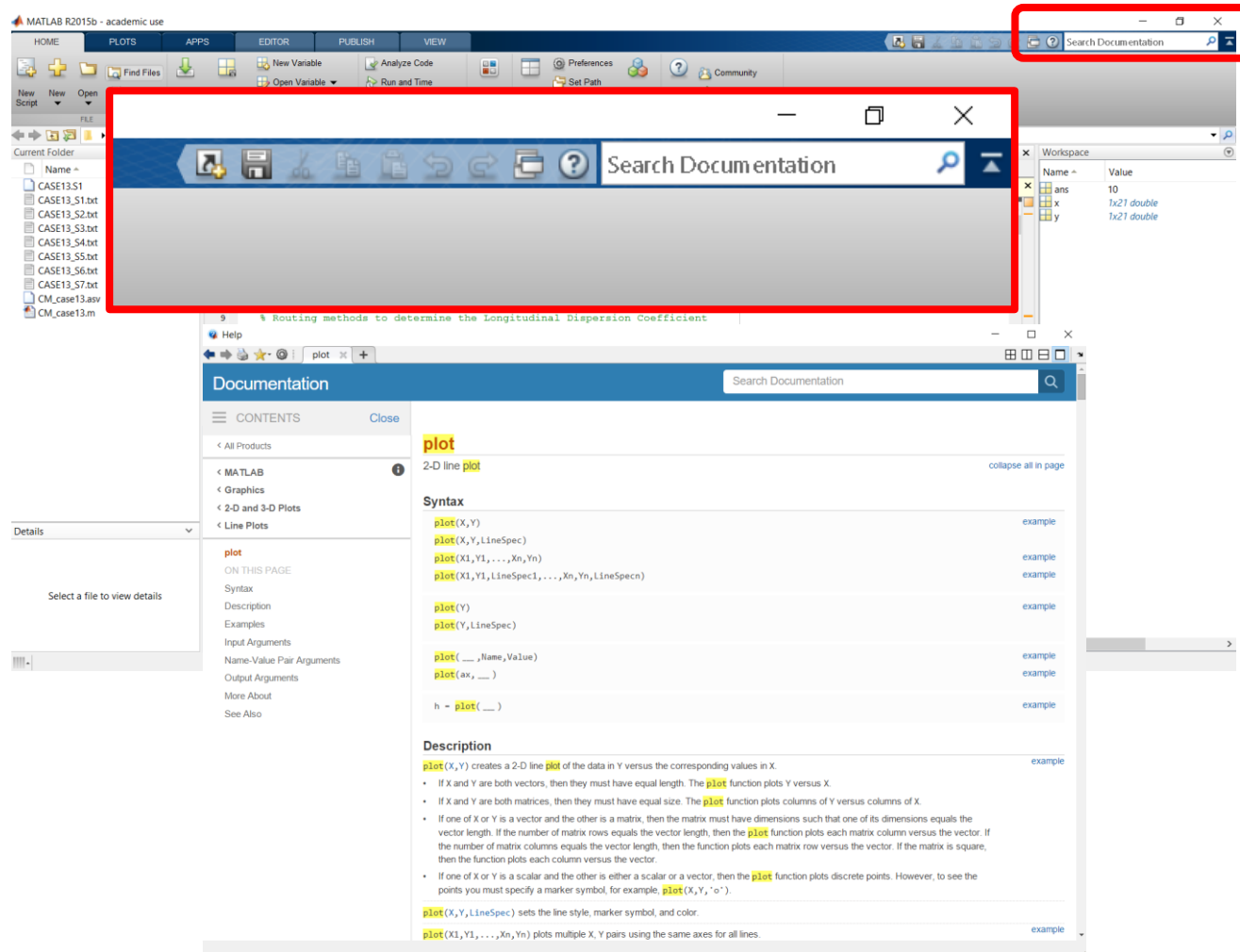
New to MATLAB? See resources for [Getting Started](#).

```
>> help figure
figure Create figure window.
figure, by itself, creates a new figure window, and returns
its handle.

figure(H) makes H the current figure, forces it to become visible,
and raises it above all other figures on the screen. If Figure H
does not exist, and H is an integer, a new figure is created with
handle H.

GCF returns the handle to the current figure.

Execute GET(H) to see a list of figure properties and
```



The image shows the MATLAB R2015b - academic use interface. The top window is the 'Search Documentation' window, which is highlighted with a red border. Below it is the 'Documentation' window, also highlighted with a red border, showing the 'plot' function documentation. The 'plot' function documentation includes a 'Syntax' section with various function calls and a 'Description' section with detailed information about the function's usage.

Search Documentation

Documentation

plot

2-D line plot

Syntax

`plot(X,Y)` example

`plot(X,Y,LineStyle)` example

`plot(X1,Y1,...,Xn,Yn)` example

`plot(X1,Y1,LineStyle1,...,Xn,Yn,LineStylen)` example

`plot(Y)` example

`plot(Y,LineStyle)` example

`plot(___,Name,Value)` example

`plot(ax, ___)` example

`h = plot(___)` example

Description

`plot(X,Y)` creates a 2-D line plot of the data in Y versus the corresponding values in X.

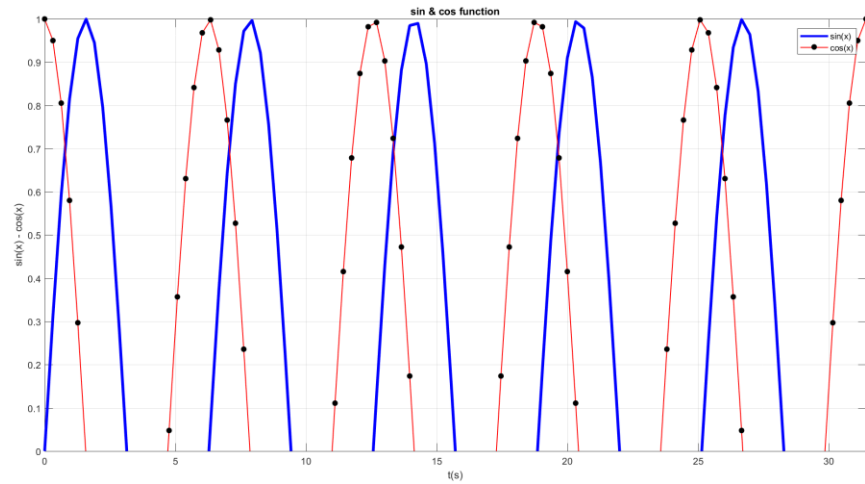
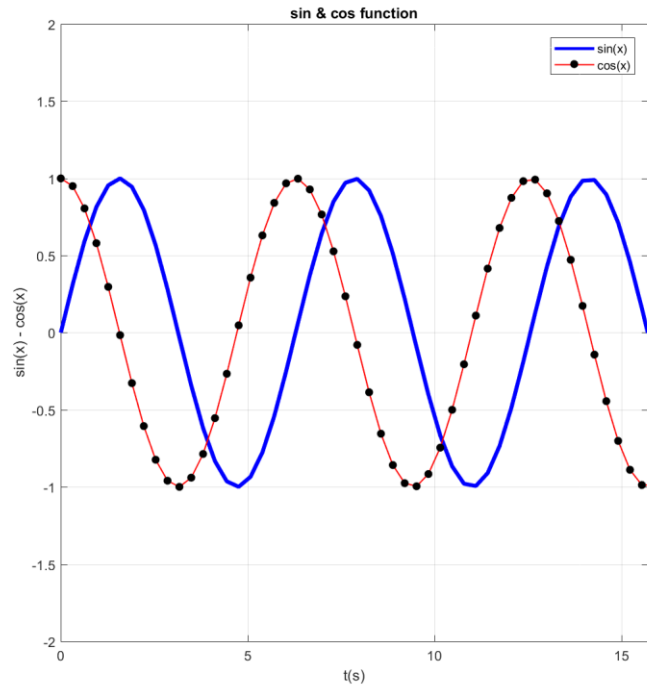
- If X and Y are both vectors, then they must have equal length. The `plot` function plots Y versus X.
- If X and Y are both matrices, then they must have equal size. The `plot` function plots columns of Y versus columns of X.
- If one of X or Y is a vector and the other is a matrix, then the matrix must have dimensions such that one of its dimensions equals the vector length. If the number of matrix rows equals the vector length, then the `plot` function plots each matrix column versus the vector. If the number of matrix columns equals the vector length, then the function plots each matrix row versus the vector. If the matrix is square, then the function plots each column versus the vector.
- If one of X or Y is a scalar and the other is either a scalar or a vector, then the `plot` function plots discrete points. However, to see the points you must specify a marker symbol, for example, `plot(X,Y,'o')`.

`plot(X,Y,LineStyle)` sets the line style, marker symbol, and color.

`plot(X1,Y1,...,Xn,Yn)` plots multiple X, Y pairs using the same axes for all lines.

- Create a new script;
- Create a comment with the phrase: **Lesson 1 – Exercise 2: Introduction To Matlab**
- Save the script (file.m) – exercise_2.m, inside the folder matlab_introduction;
- Use the command: clear all, close all, clc;
- Use the help function (in the command window) to understand how to use the function *linspace*;
- Create a vector (x) of 100 samples between 0 and 10π . (“linspace”);
- Calculate $y = \sin(x)$ of the vector;
- Create a figure window (figure_1 = figure());
- Plot x vs y;
- Insert the title (sin function);
- Insert the xlabel(t[s]);
- Insert the ylabel(sin(x));
- Add the grid and the external box to the figure; (to understand how to insert box and grid use the help function in the command window);
- Calculate $z = \cos(x)$ of the vector;
- Use the help function (in the command window) to understand how to use the function *hold* for include different plots in the same figure windows;

- Create a new figure window (`figure_2 = figure()`)
- Plot together x vs y & x vs z ;
- Insert a new title (sin and cos functions)
- Insert the new ylabel(sin(x) & cos(x));
- Insert the xlabel ($t(s)$);
- Add the grid and the external box to the figure;
- Add the legend to the different curves;
- **For plot sin(x):** Set the color (blue), line style (solid line) and line width = 3;
- **For plot cos(x):** Set the color (red), line style (solid line), line width (1), marker (o), marker size (10), ; Marker Face and edge Color (black).
- Set the axis between **0 to 5π** along x direction and **-2 to 2** along y direction
- Save the image 2: `figure_2` with ***PaperSize 10x10***
- Create a figure 3 equal to the previous one but setting the axis along y direction between 0 to 2 and 0 to 10π along x direction.
- Save the image 3: `figure_3` with ***PaperSize 20x10***



Syntax:

for index = **values**

Statements

End

This syntax that describes a for cycle executes a group of statements in a loop for a specified number of times.

values has one of the following forms:

- ***initVal : endVal*** — Increments the index variable from *initVal* to *endVal* by 1, and repeats execution of statements until index is greater than *endVal*.
- ***initVal :step: endVal*** — Increments index by the value *step* on each iteration, or decrements index when *step* is negative.

Syntax:

for index = values

Statements

End

This syntax that describes a for cycle executes a group of statements in a loop for a specified number of times.

`s = 10;`

`for c = 1:s`

`for r = 1:s`

`H(r,1) = 1/(r+r-1);`

`end`

`end`

```
s = 10;
% H = zeros(s);
%
% for c = 1:s
    for r = 1:s
        H(r,1) = 1/(r+r-1);
    end
% end
.
```

10x1 double

	1	2	3	4
1	1			
2	0.3333			
3	0.2000			
4	0.1429			
5	0.1111			
6	0.0909			
7	0.0769			
8	0.0667			
9	0.0588			
10	0.0526			
11				
12				

Syntax:

for index = values

Statements

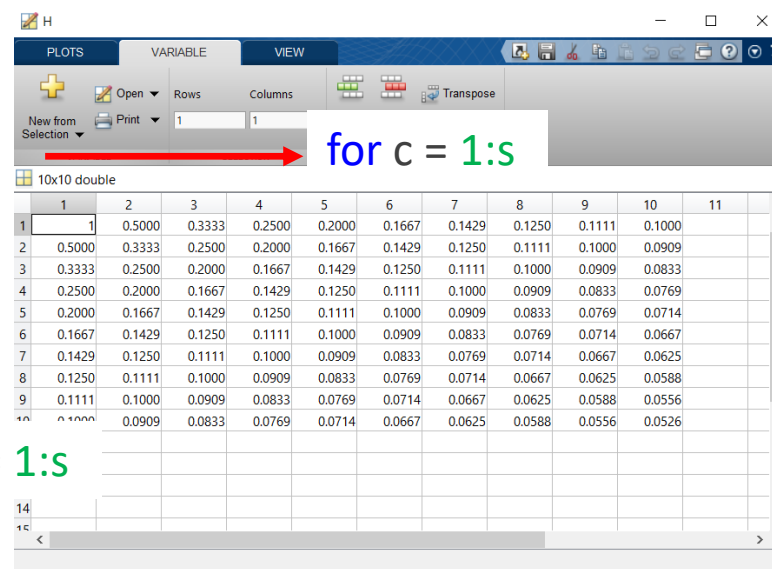
End

This syntax that describes a for cycle executes a group of statements in a loop for a specified number of times.

`s = 10;`

```
for c = 1:s
    for r = 1:s
        H(r,c) = 1/(r+c-1);
    end
end
```

`for r = 1:s`



Syntax

```
if expression
  statements
elseif expression
  statements
else
  statements
end
```

Executes a group of **statements** when the **expression** is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

The **elseif** and **else** blocks are optional. The statements execute only if previous expressions in the *if...end* block are false.

An *if* block can include multiple *elseif* blocks.

Assign to a matrix values that
depend on their indices

Syntax

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

```
for c = 1:ncols
    for r = 1:nrows
        if r == c
            A(r,c) = 2;
        elseif abs(r-c) == 1
            A(r,c) = -1;
        else
            A(r,c) = 0;
        end
    end
end
```

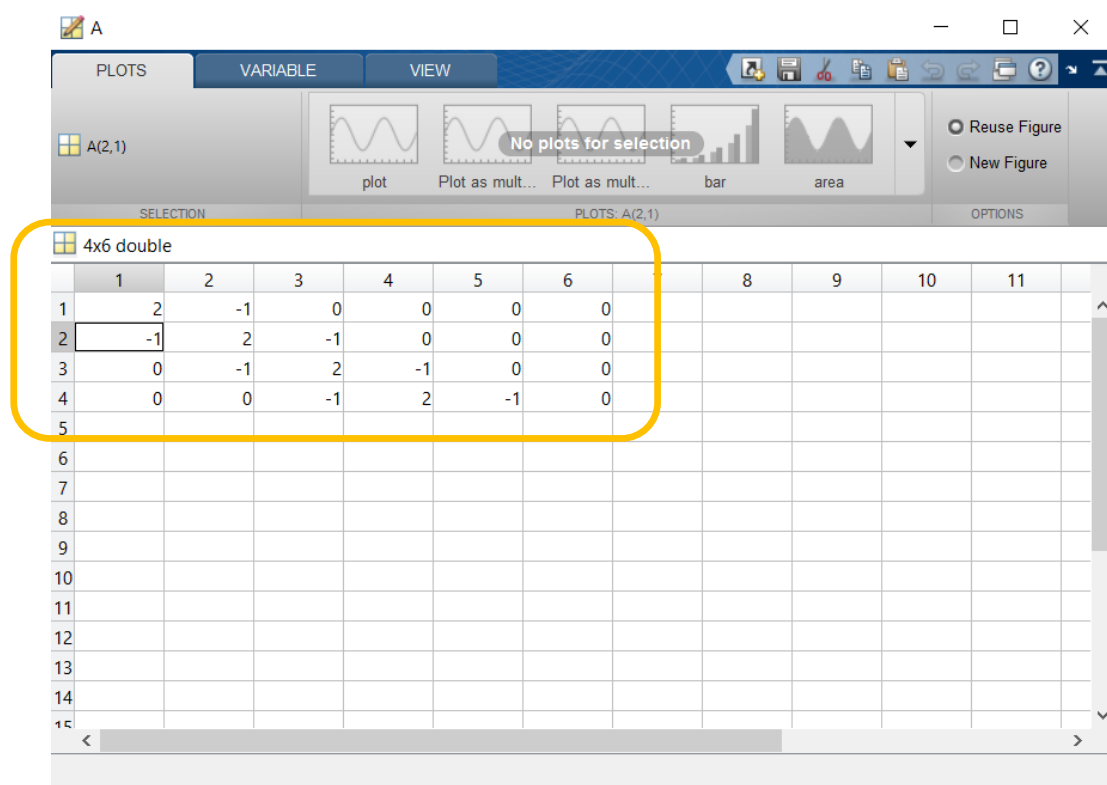
Loop to define each element of matrix A.
Assign 2 on the main diagonal, -1 on the
adjacent diagonals, and 0 everywhere
else.

Assign to a matrix values that
depend on their indices

Syntax

```

if expression
    statements
elseif expression
    statements
else
    statements
end
  
```



... through the main and adjacent
element a new value. Assign 2 on the
main diagonal, -1 on the adjacent
diagonals, and 0 everywhere else.

function LOAD, file .txt

Load data from a folder into workspace.

S = load(FILENAME) loads the variables from a MAT-file into a structure array (matrix)

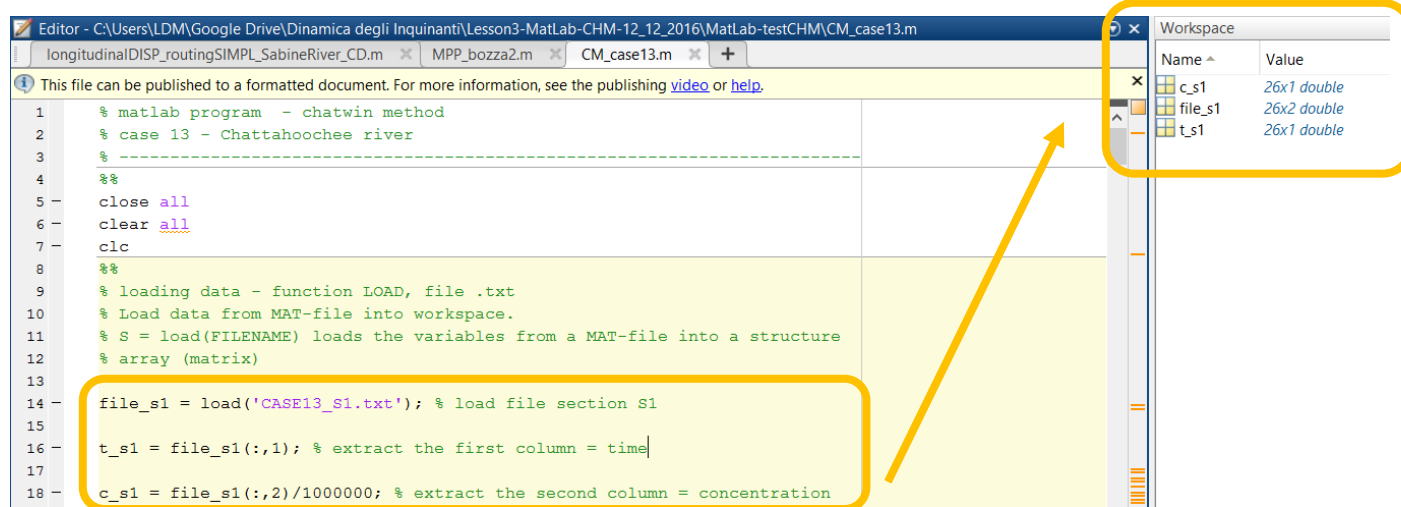
CASE13_S1.txt - Blocco note

File Modifica Formato Visualizza ?

4.000	0.00
4.200	0.97
4.400	3.98
4.600	9.21
4.800	11.46
5.000	10.23
5.200	7.23
5.400	4.64
5.600	2.86
5.800	1.75
6.000	0.88
6.200	0.54
6.400	0.37
6.600	0.28
6.800	0.22
7.000	0.18
7.200	0.16
7.400	0.14
7.600	0.13
7.800	0.11
8.000	0.08
8.200	0.07
8.400	0.05
8.600	0.03
8.800	0.01
9.000	0.00

time

C



The screenshot shows the MATLAB Editor with a script file named 'CM_case13.m'. The script contains comments and code for loading data from 'CASE13_S1.txt'. A yellow box highlights the code lines 14-18, which load the file and extract time and concentration vectors. An orange arrow points from this code block to the MATLAB Workspace window on the right. The Workspace window shows three variables: 'c_s1' (26x1 double), 'file_s1' (26x2 double), and 't_s1' (26x1 double).

```
1 % matlab program - chatwin method
2 % case 13 - Chattahoochee river
3 % -----
4 %%
5 close all
6 clear all
7 clc
8 %%
9 % loading data - function LOAD, file .txt
10 % Load data from MAT-file into workspace.
11 % S = load(FILENAME) loads the variables from a MAT-file into a structure
12 % array (matrix)
13
14 file_s1 = load('CASE13_S1.txt'); % load file section S1
15
16 t_s1 = file_s1(:,1); % extract the first column = time
17
18 c_s1 = file_s1(:,2)/1000000; % extract the second column = concentration
```

Loading of the entire matrix (*file .txt*) and extraction of the vectors of the time (*first column*) and concentration (*second column*)

function *importdata* (filename)
A = importdata (FILENAME)

Editor - test.m

test.m x +

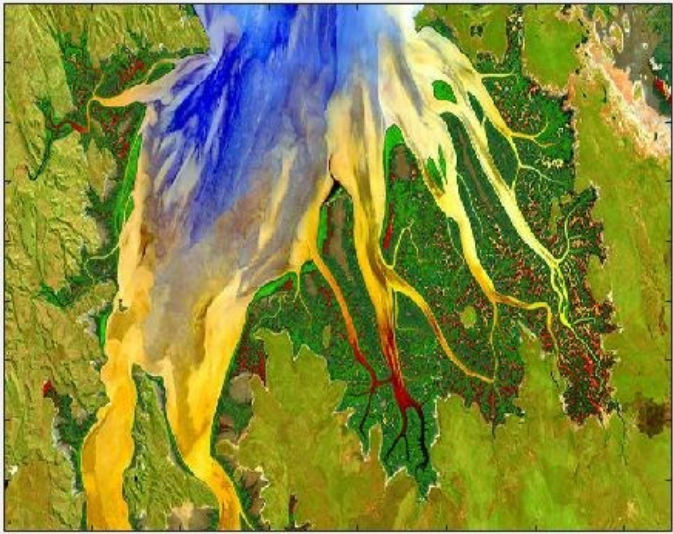
This file can be opened as a Live Script. For more information, see [Creating Live Scripts](#).

```
2 % A = importdata(filename) loads data into array A.  
3  
4 clear all  
5 close all  
6 clc  
7  
8 A = importdata('Cambridge Gulf_Australia.jpg');  
9 image(A)  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31
```

Variables - A.colheaders

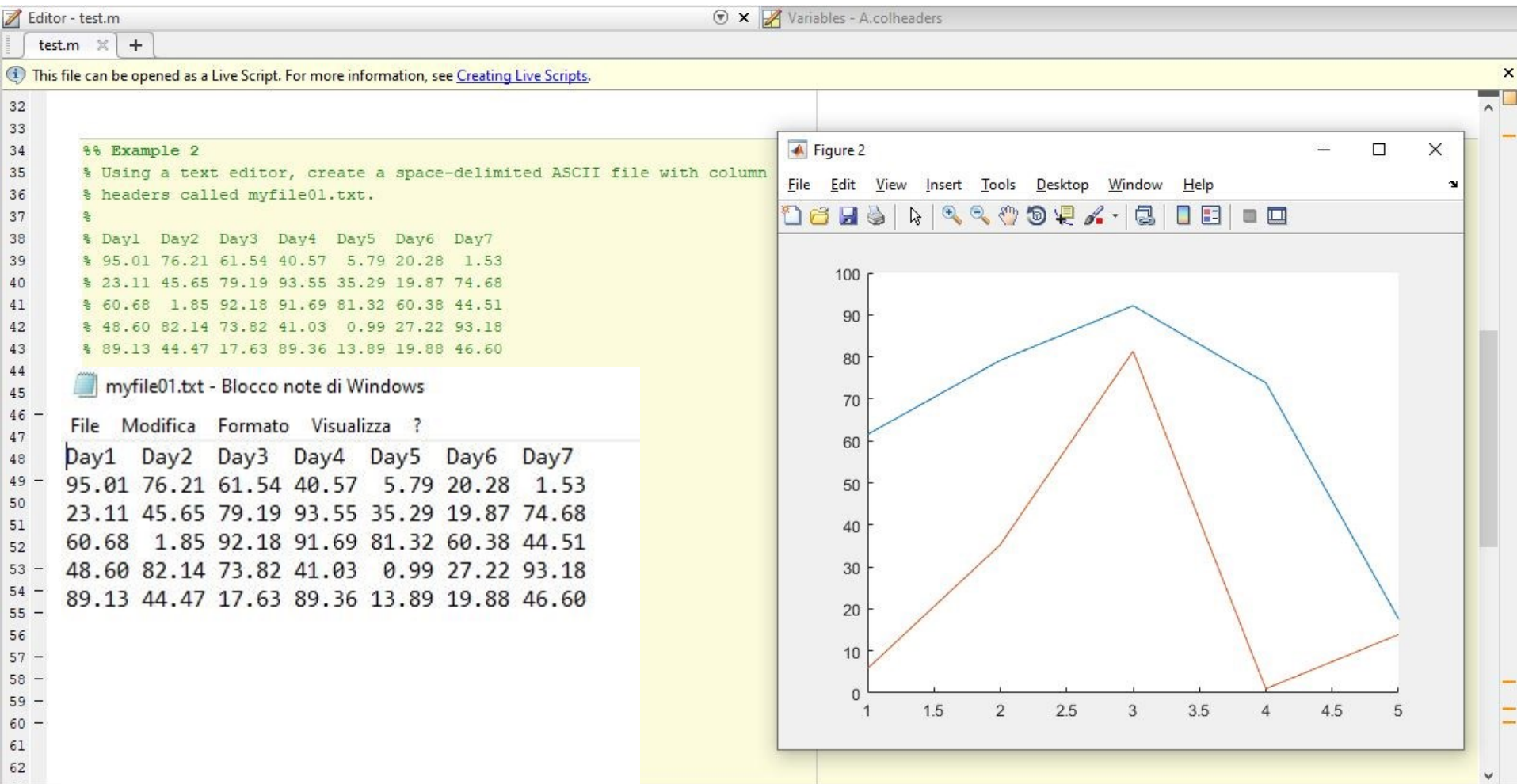
Figure 1

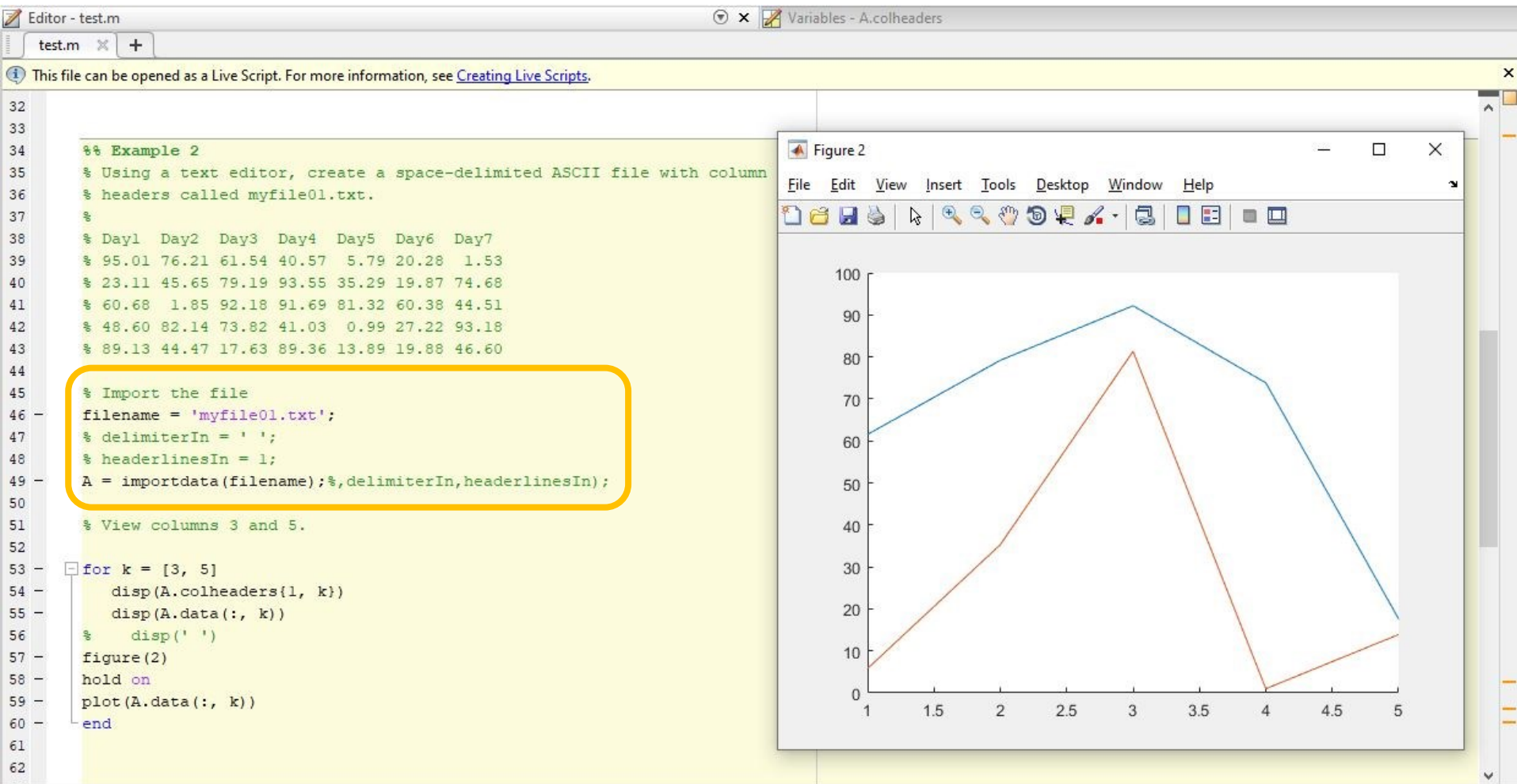
File Edit View Insert Tools Desktop Window Help

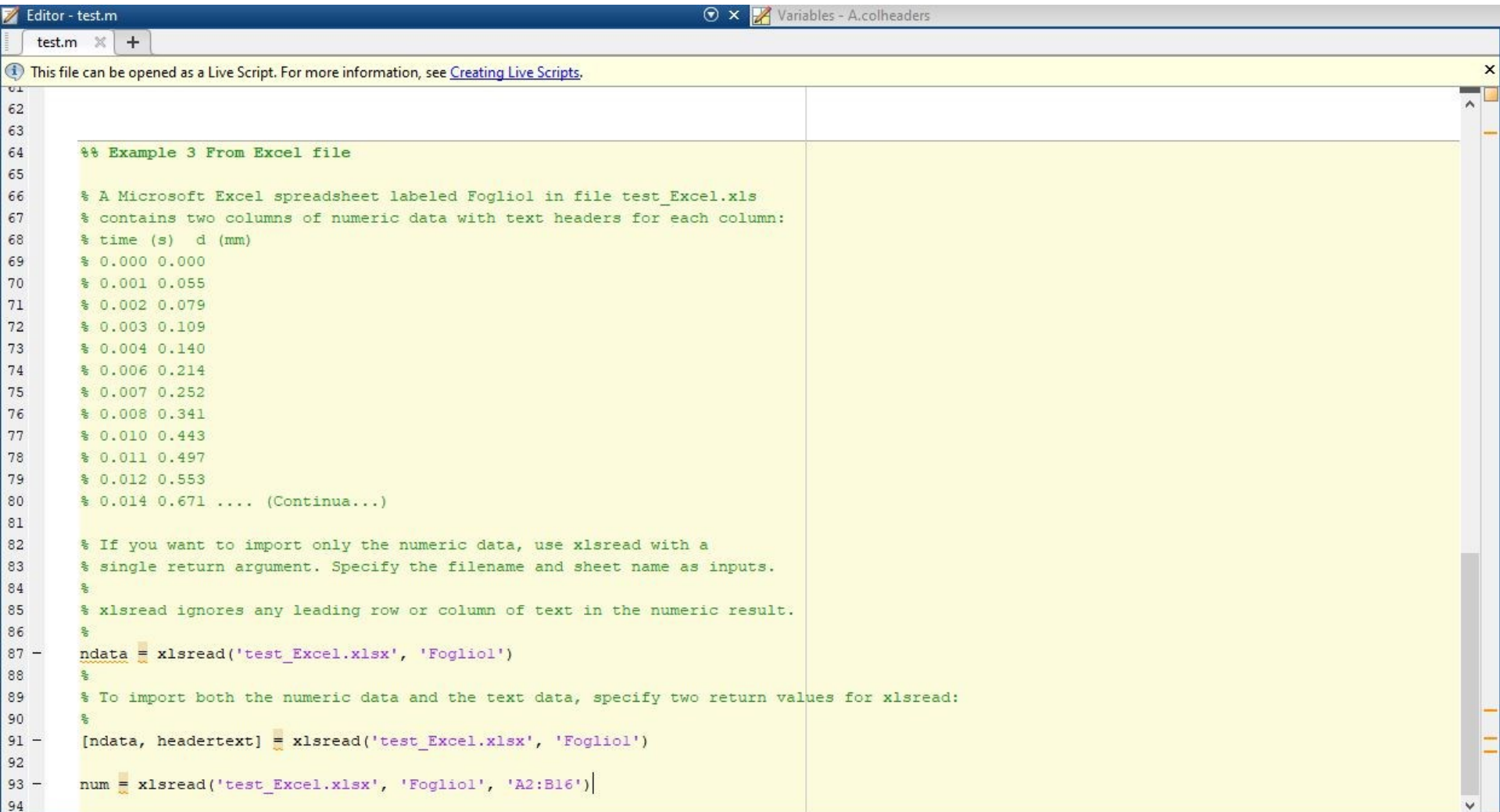


50
100
150
200
250
300
350
400

100 200 300 400 500 600 700







Editor - test.m

test.m x +

This file can be opened as a Live Script. For more information, see [Creating Live Scripts](#).

```
61  
62  
63  
64 %% Example 3 From Excel file  
65  
66 % A Microsoft Excel spreadsheet labeled Fogliol in file test_Excel.xls  
67 % contains two columns of numeric data with text headers for each column:  
68 % time (s) d (mm)  
69 % 0.000 0.000  
70 % 0.001 0.055  
71 % 0.002 0.079  
72 % 0.003 0.109  
73 % 0.004 0.140  
74 % 0.006 0.214  
75 % 0.007 0.252  
76 % 0.008 0.341  
77 % 0.010 0.443  
78 % 0.011 0.497  
79 % 0.012 0.553  
80 % 0.014 0.671 .... (Continua...)  
81  
82 % If you want to import only the numeric data, use xlsread with a  
83 % single return argument. Specify the filename and sheet name as inputs.  
84 %  
85 % xlsread ignores any leading row or column of text in the numeric result.  
86 %  
87 - ndata = xlsread('test_Excel.xlsx', 'Fogliol')  
88 %  
89 % To import both the numeric data and the text data, specify two return values for xlsread:  
90 %  
91 - [ndata, headertext] = xlsread('test_Excel.xlsx', 'Fogliol')  
92  
93 - num = xlsread('test_Excel.xlsx', 'Fogliol', 'A2:B16')  
94
```