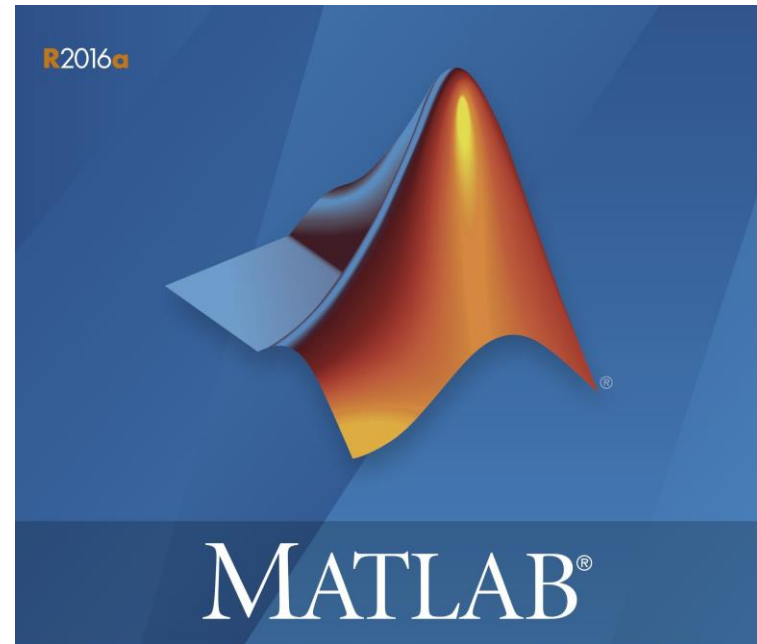


MATLAB

a short primer



Ovidiu Ivanov
'Gheorghe Asachi' Technical University,
Electrical Engineering Faculty
Iasi, Romania
www.tuiasi.ro



Part Two

Getting input data for our programs and saving results

How to define, save and load data
Some useful statements of general purpose

Defining data for your programs

- **MATLAB stores numerical and text data into variables**
 - You can define a variable with a name and a value
 - You can compute values for variables using numbers, mathematical operators, other variables, mathematical functions
 - You can import variables or values into variables from external files
- In MATLAB, you don't need to declare the variables and their types before using them
- MATLAB sets the type according to what you write
- You can define a variable anywhere in your program
- A variable is accessible if you can see it in the Workspace

Most common variable types

- integer
- double
- char
- string
- array of integer or float – vector or matrix
- cell and cell arrays

Variable types in MATLAB

CONTENTS

Close

< Documentation Home

< MATLAB

< Language Fundamentals

< Data Types

Numeric Types

Characters and Strings

Dates and Time

Categorical Arrays

Tables

Timetables

Structures

Cell Arrays

Function Handles

Map Containers

Time Series

Data Type Identification

Data Type Conversion

Numeric Types

Integer and floating-point data

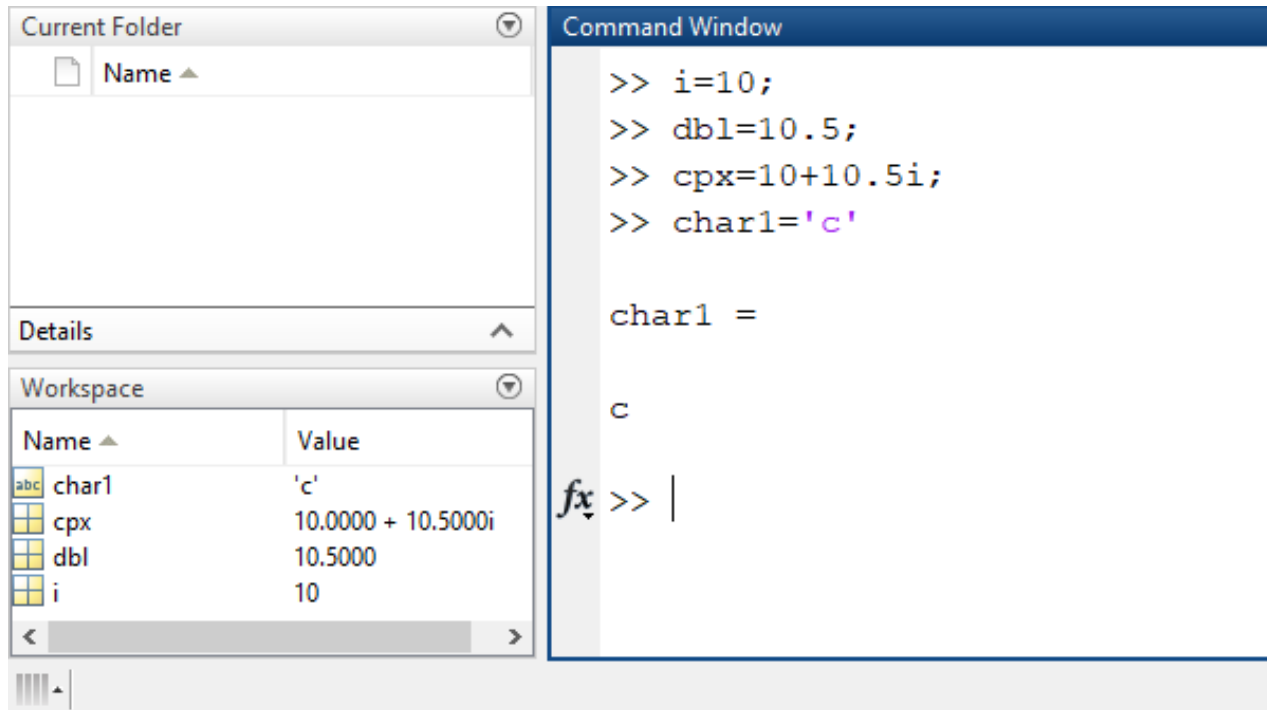
Numeric classes in MATLAB® include signed and unsigned integers, and single-precision and double-precision floating-p cannot change the default type and precision.) You can choose to store any number, or array of numbers, as integers or double precision.

All numeric types support basic array operations, such as subscripting, reshaping, and mathematical operations.

Functions

<code>double</code>	Double-precision arrays
<code>single</code>	Single-precision arrays
<code>int8</code>	8-bit signed integer arrays
<code>int16</code>	16-bit signed integer arrays
<code>int32</code>	32-bit signed integer arrays
<code>int64</code>	64-bit signed integer arrays
<code>uint8</code>	8-bit unsigned integer arrays
<code>uint16</code>	16-bit unsigned integer arrays
<code>uint32</code>	32-bit unsigned integer arrays
<code>uint64</code>	64-bit unsigned integer arrays
<code>cast</code>	Cast variable to different data type
<code>typecast</code>	Convert data types without changing underlying data
<code>isinteger</code>	Determine if input is integer array
<code>isfloat</code>	Determine if input is floating-point array

Defining integer, double, char and complex variables



- You can use small or capital letters, numbers and underscore
- The variable name must start with a letter
- If you don't terminate a line with semicolon, the variable name and value are displayed on the screen.

Defining vectors and matrices

- Arrays are defined between square brackets.
- In vectors and matrices, columns are separated with space or comma, and rows are separated by semicolons
- The maximum number of dimensions is not fixed

Command Window

```
>> M1=[1 2 3; 4 5 6];  
>> M2=[1,2,3;4,5,6];  
>> M1-M2
```

ans =

```
0     0     0  
0     0     0
```

fx >> |

Addressing array elements

- You can address individual elements using round brackets
- You can address chunks of arrays using square brackets and Excel-like ranges

Command Window

```
>> M2=[1,2,3;4,5,6]
```

```
M2 =
```

1	2	3
4	5	6

```
>> n1=M2(2,2)
```

```
n1 =
```

5

```
>> v1=M2(2,:)
```

```
v1 =
```

4	5	6
---	---	---

fx >> |

Elementary operations with numbers

- addition $+$
 - subtraction $-$
 - multiplication $*$
 - division $/$
 - power $^$
-
- Parentheses and operation precedence are observed

Elementary operations with arrays

- all of the above
- MATLAB will treat any variables as matrices and it will multiply them accordingly.
- If you want to multiply or divide matrices or vectors element by element, you should use `.*` or `./`

Command Window

```
>> M1
```

```
M1 =
```

```
    1    2    3
    4    5    6
```

```
>> M2
```

```
M2 =
```

```
    1    2    3
    4    5    6
```

```
>> M1*M2
```

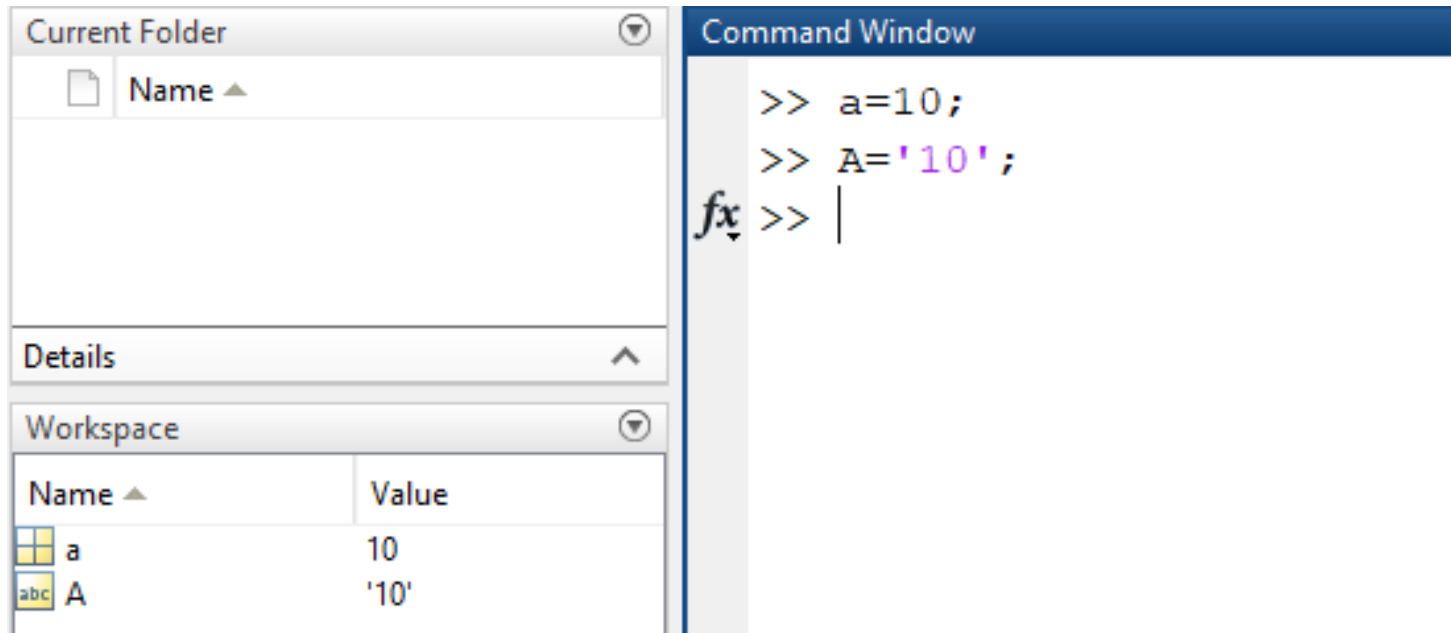
```
Error using *
Inner matrix dimensions must agree.
```

```
>> M1.*M2
```

```
ans =
```

```
    1    4    9
   16   25   36
```

The difference between numbers and strings



- Notice the different symbol from the Workspace
 - conversion from one to another: `num2str` or `str2num`
- **MATLAB is case sensitive**, variables `a` and `A` are distinct

Defining cell variables and arrays

```
Command Window
>> Cell11=[{'Club'} {'Points'};{'Man C'} {59};{'Man U'} {47}]

Cell11 =

    'Club'    'Points'
    'Man C'   [    59]
    'Man U'   [    47]

fx >> |
```

- You define cell arrays as you define matrices, but putting individual elements between curly brackets
- Cells can store any type of element
- Excel or database imports with more than one type of values will be automatically stored by MATLAB into cell arrays

Converting between cells and cell arrays and numbers or strings

Command Window

```
>> Cell1=[{'Club'} {'Points'};{'Man C'} {59};{'Man U'} {47}]
```

```
Cell1 =
```

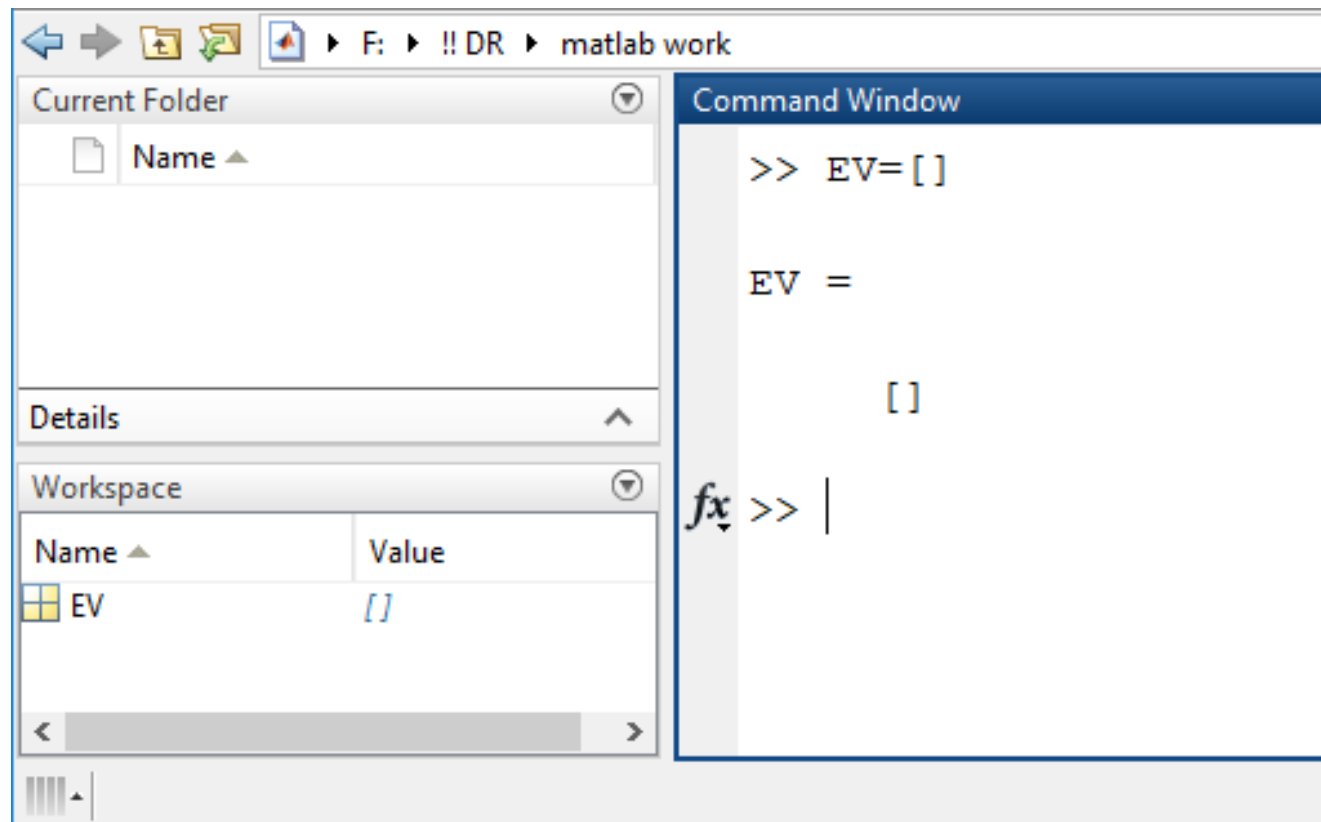
'Club'	'Points'
'Man C'	[59]
'Man U'	[47]

```
fx >> |
```

- Cell to numbers: **cell2mat**
- Numbers to cells: **num2cell** or **mat2cell**
- Cell to string: **char**
- Strings to cell: **cellstr**

The empty variable

- You can define an empty variable to which you can assign values later.

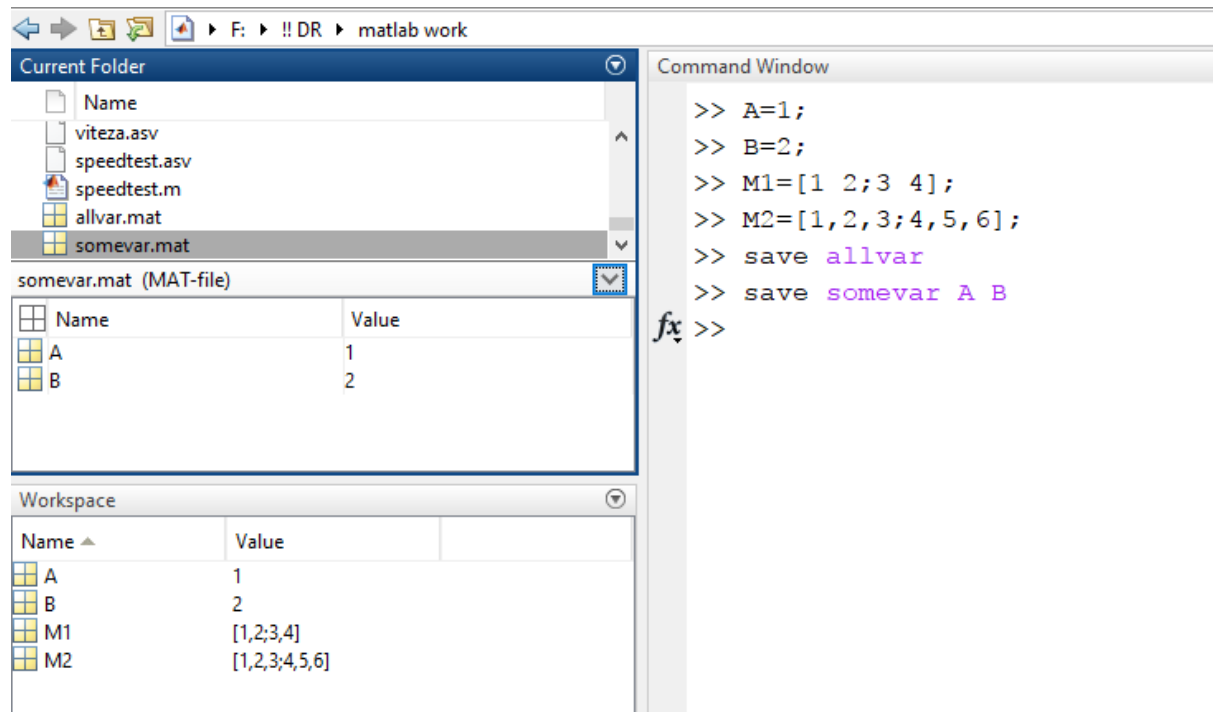


Saving variables for later use – the save statement

- MATLAB can save variables into .mat files
- You can save all the variables from the workspace or just some of them

```
save filename
```

```
save filename var1 var 2 var 3
```



The image shows a screenshot of the MATLAB interface. The 'Current Folder' pane on the left displays a list of files: viteza.asv, speedtest.asv, speedtest.m, allvar.mat, and somevar.mat. The 'somevar.mat' file is selected, and its contents are shown in a table below it:

Name	Value
A	1
B	2

The 'Workspace' pane at the bottom shows the current workspace variables:

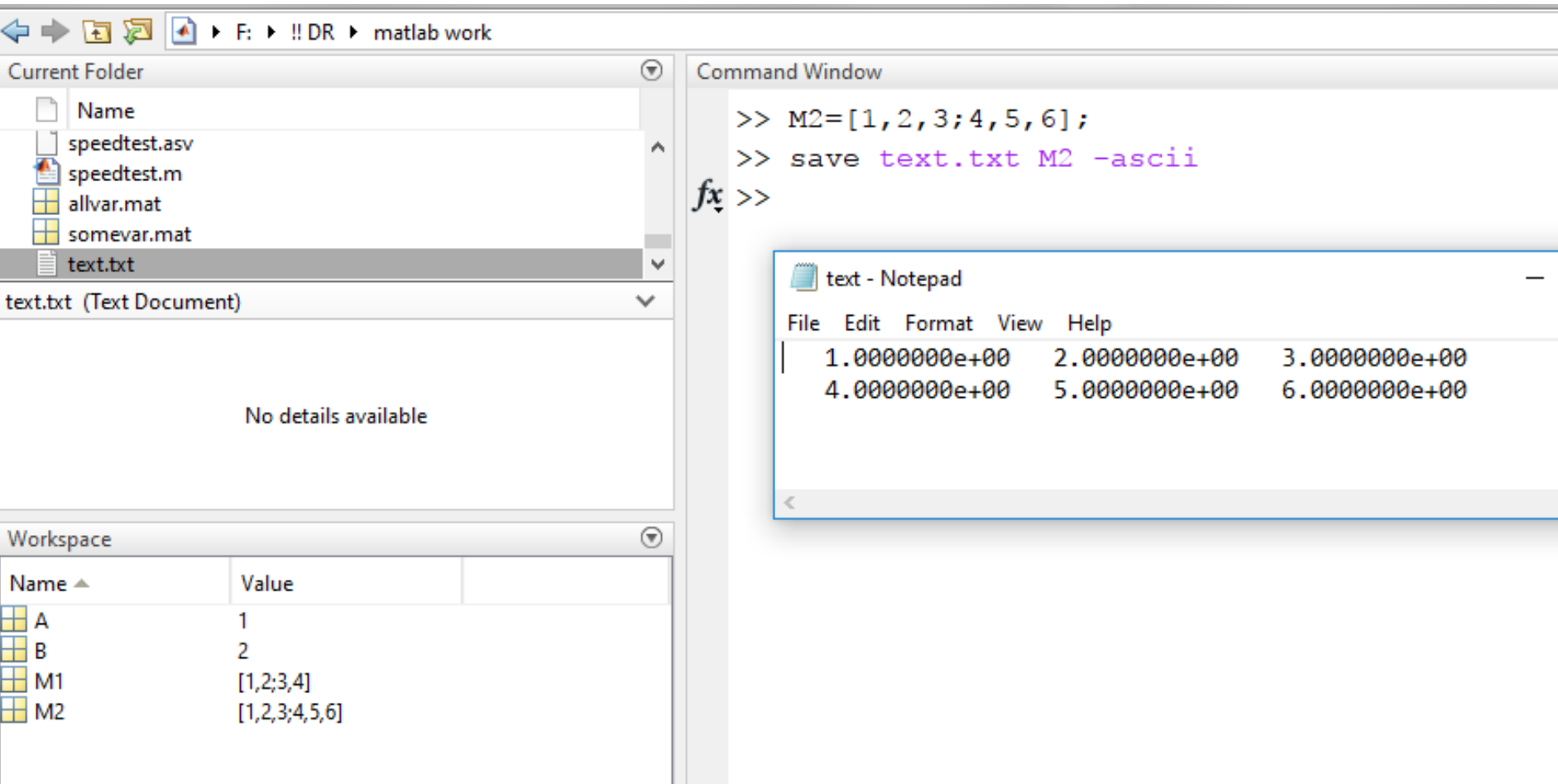
Name	Value
A	1
B	2
M1	[1,2;3,4]
M2	[1,2,3;4,5,6]

The 'Command Window' on the right shows the following commands being executed:

```
>> A=1;  
>> B=2;  
>> M1=[1 2;3 4];  
>> M2=[1,2,3;4,5,6];  
>> save allvar  
>> save somevar A B  
fx >>
```

Saving variables as text files

- You can save only one variable from the Workspace as a text file



The image displays the MATLAB environment with three main panels: Current Folder, Command Window, and Workspace.

Current Folder: Shows a list of files including `speedtest.asv`, `speedtest.m`, `allvar.mat`, `somevar.mat`, and `text.txt`. The `text.txt` file is selected, and its details are shown as "No details available".

Command Window: Contains the following MATLAB commands:

```
>> M2=[1,2,3;4,5,6];  
>> save text.txt M2 -ascii  
fx >>
```

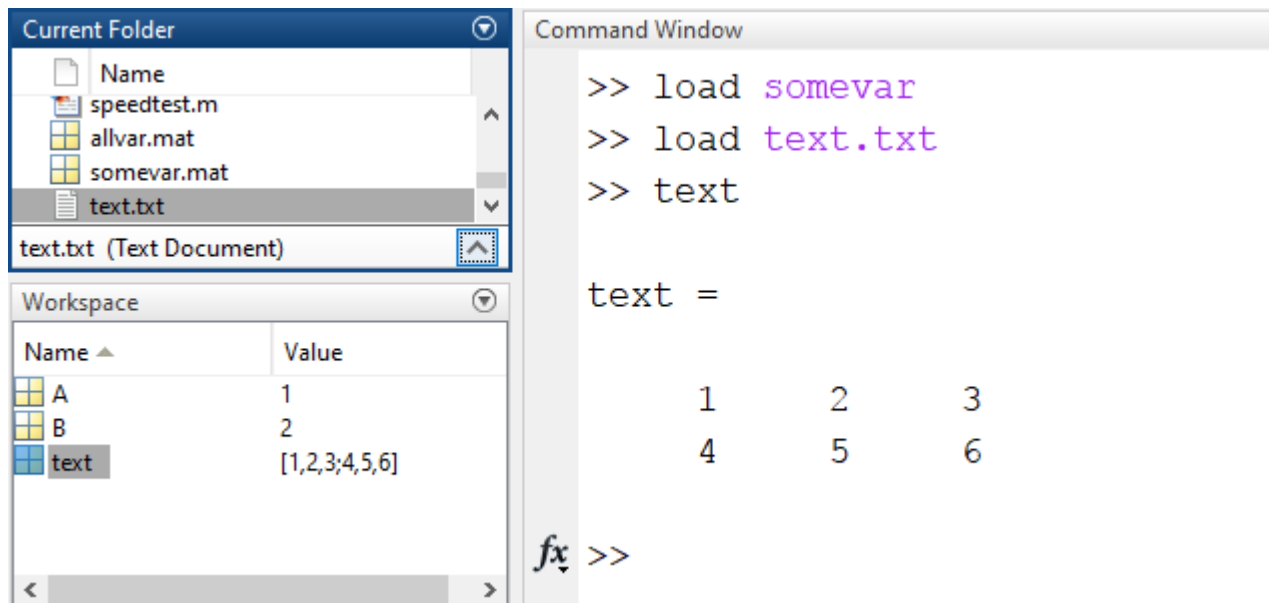
Workspace: Displays a table of variables in the workspace:

Name	Value
A	1
B	2
M1	[1,2;3,4]
M2	[1,2,3;4,5,6]

text - Notepad: A window showing the contents of `text.txt`, which contains the ASCII representation of the variable `M2`:

```
1.0000000e+00 2.0000000e+00 3.0000000e+00  
4.0000000e+00 5.0000000e+00 6.0000000e+00
```

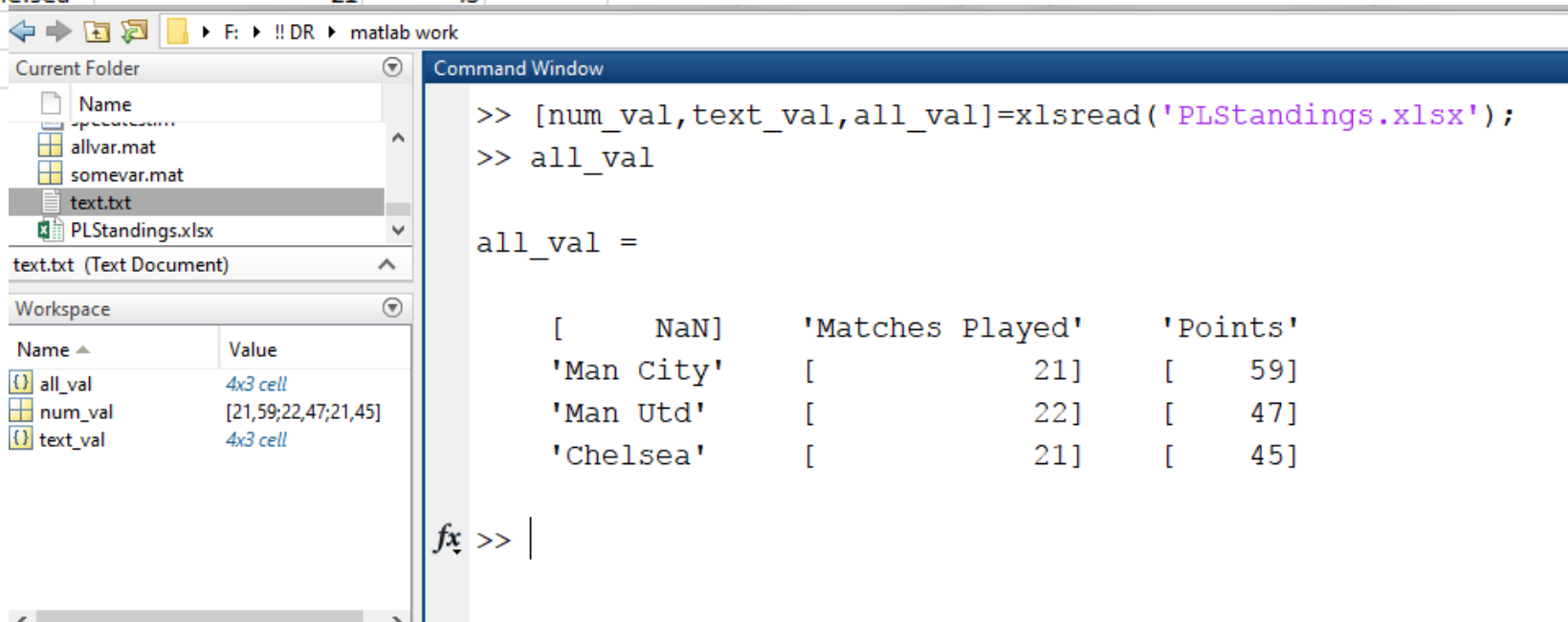
Loading variables from .mat and .txt files – the load statement



- MATLAB can read tab delimited and space delimited text files and import them into matrices
- For .mat file, you can skip the extension. For others, you cannot.

Reading from Excel files – the xlsread statement

	A	B	C	D	E
1			Matches Played	Points	
2		Man City	21	59	
3		Man Utd	22	47	
4		Chelsea	21	45	



```
>> [num_val,text_val,all_val]=xlsread('PLStandings.xlsx');  
>> all_val  
  
all_val =  
  
[      NaN]    'Matches Played'    'Points'  
'Man City'    [          21]    [      59]  
'Man Utd'    [          22]    [      47]  
'Chelsea'    [          21]    [      45]  
  
fx >> |
```

Current Folder

- Name
- allvar.mat
- somevar.mat
- text.txt
- PLStandings.xlsx

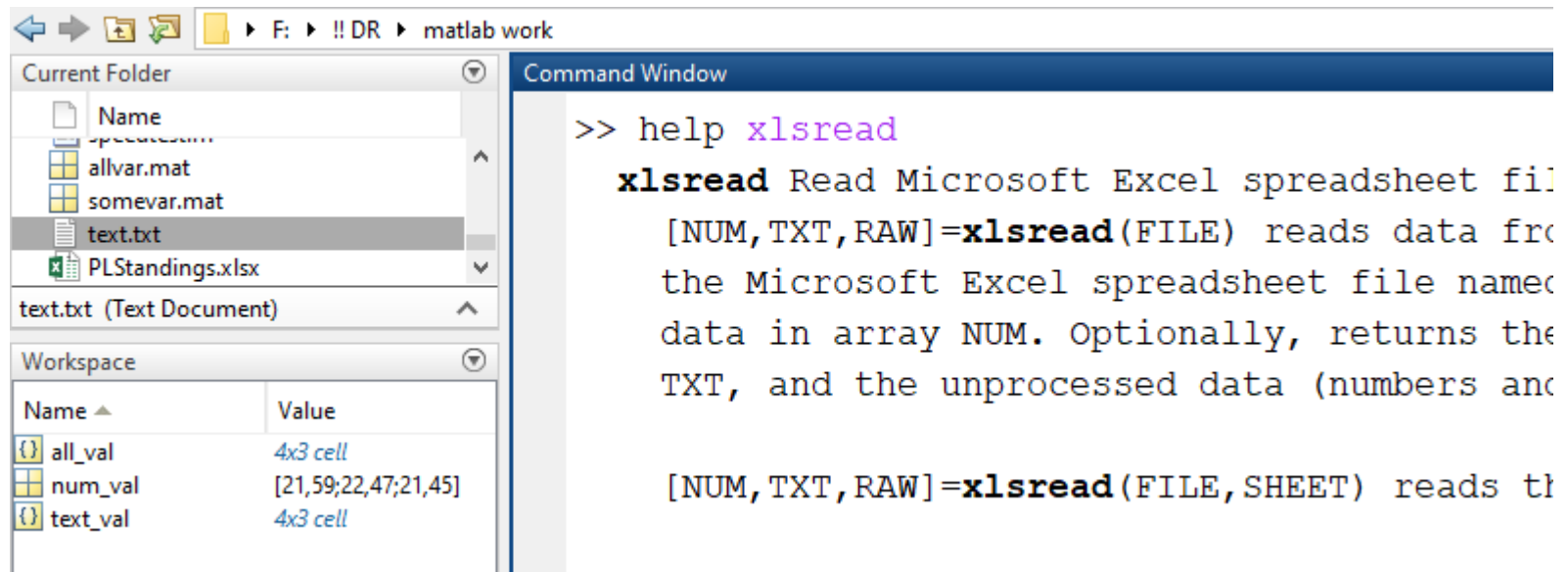
text.txt (Text Document)

Workspace

Name	Value
all_val	4x3 cell
num_val	[21,59;22,47;21,45]
text_val	4x3 cell

- MATLAB can also write into Excel files, with **xlswrite**

Getting help



- Write **help** followed by the function name
- You can make **xlsread** to read from a specific sheet and cell range

Getting extended help

The screenshot displays the MATLAB R2016a environment with the documentation window open for the `xlsread` function. The documentation window on the left shows the 'Syntax' and 'Description' sections. The 'Syntax' section lists various ways to call `xlsread`, such as `num = xlsread(filename)` and `[num,txt,raw] = xlsread(filename, sheet, range)`. The 'Description' section explains that `xlsread` reads data from a specified worksheet and range. The MATLAB interface in the background shows the 'Current Folder' with files like `allvar.mat` and `somevar.mat`, and the 'Workspace' with variables `all_val`, `num_val`, and `text_val`. The 'Command Window' on the right contains example code for reading data from a worksheet and requesting numeric, text, and unprocessed data.

Documentation Window:

Documentation

Search Documentation

CONTENTS Close

< All Products

< MATLAB

< Data Import and Export

< Standard File Formats

< Spreadsheets

< MATLAB

< Functions

xlsread

ON THIS PAGE

Syntax

Description

Examples

Input Arguments

Output Arguments

Limitations

More About

See Also

Syntax

```
num = xlsread(filename)
num = xlsread(filename, sheet)
num = xlsread(filename, x1Range)
num = xlsread(filename, sheet, range)
num = xlsread(filename, sheet, range, processFcn)
[num,txt,raw] = xlsread(filename, sheet, range)
[num,txt,raw,custom] = xlsread(filename, sheet, range, processFcn)
```

Description

`num = xlsread(filename)` reads the numeric data in the first worksheet of the spreadsheet file `filename` into the array `num`.

`num = xlsread(filename, sheet)` reads the numeric data from the worksheet `sheet` into the array `num`.

`num = xlsread(filename, x1Range)` reads the numeric data from the range `x1Range` into the array `num`.

`num = xlsread(filename, sheet, range)` reads the numeric data from the range `range` in the worksheet `sheet` into the array `num`.

`num = xlsread(filename, sheet, range, processFcn)` reads the numeric data from the range `range` in the worksheet `sheet` into the array `num`, and applies the function `processFcn` to the data.

`[num,txt,raw] = xlsread(filename, sheet, range)` reads the numeric data from the range `range` in the worksheet `sheet` into the array `num`, the text data into the cell array `txt`, and the unprocessed data into the cell array `raw`.

`[num,txt,raw,custom] = xlsread(filename, sheet, range, processFcn)` reads the numeric data from the range `range` in the worksheet `sheet` into the array `num`, the text data into the cell array `txt`, the unprocessed data into the cell array `raw`, and the data processed by `processFcn` into the cell array `custom`.

The `xlsread` function returns the text fields in cell array `txt`, both the numeric and text data in cell array `raw`, and the second output from `processFcn` in array `custom`. The `xlsread` function does not change the data stored in the spreadsheet. This syntax is supported only on Windows computers with Excel software.

Command Window:

```
% Read from a named worksheet:
B = xlsread('myExample.xls', 'MySheet')

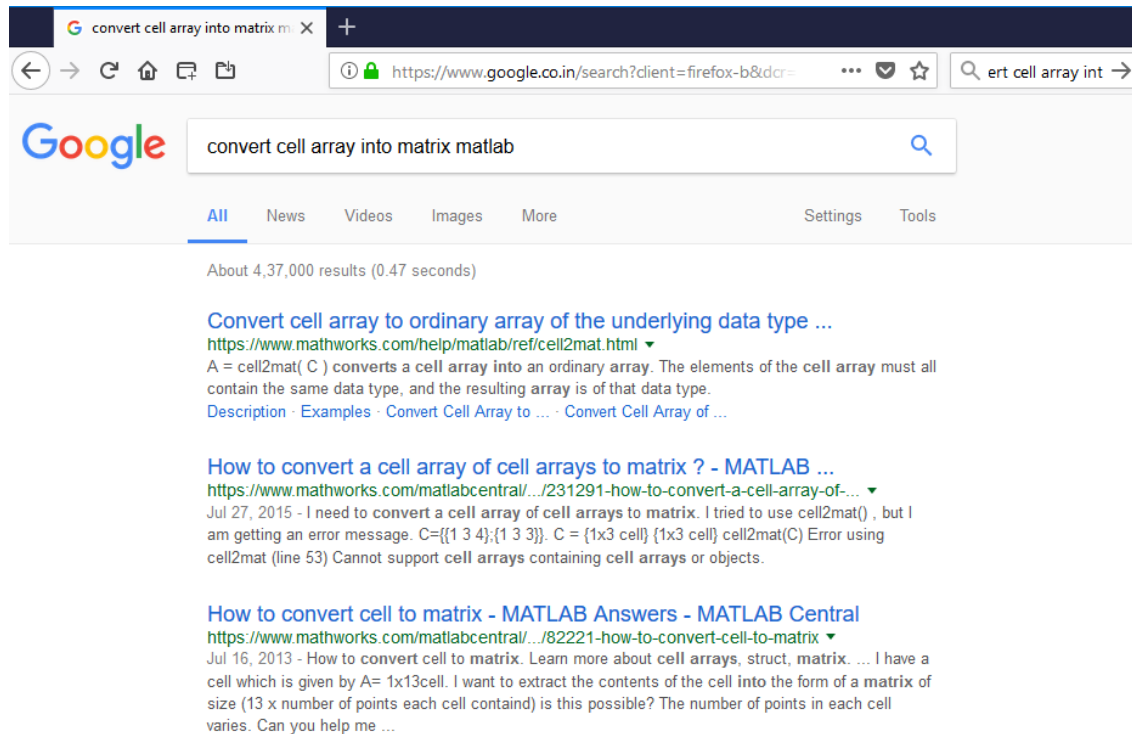
% Request the numeric data, text, and a copy of the unprocessed
% data from the first worksheet:
[ndata, text, alldata] = xlsread('myExample.xls')

See also xlswrite, xlsinfo, dlmread, importdata, textscan.

Reference page for xlsread
```

- Scroll down use the offline documentation reference pages

Getting extended help



- Google and www.mathworks.com are there to help you with code and examples !

Other useful commands

- **clc** will clear the Command Window screen, but not the memory (Workspace)
- **clear** will clear the Workspace, but not the screen
- **tic** and **toc** will time the execution of your code
- **exit** will exit MATLAB
- **rand** will generate a random number in (0,1)
- **pause** stops execution until a key is pressed
- **break** exits a loop

Useful keyboard shortcuts

- **Ctrl+C** breaks the execution of a running program
- **F5** runs a program
- **F10** runs a program step by step
- **Shift+F5** exits debugging mode
- **Ctrl+I** will align the selected code in the Editor window
- Use the arrow keys to access the history feature of the Command Window

- **How to define, save and load data**
 - The most used types of variables
 - The load and save commands
 - xlsread / xlswrite
- **Some useful functions and statements of general purpose**
 - Getting help
 - Clearing your screen and memory

End of Part Two - Review

- **Feel free to contact me with questions !**

Ovidiu Ivanov

Lecturer, PhD

The 'Gheorghe Asachi' Technical University

The Electrical Engineering Faculty

www.tuiasi.ro

oviduiuivanov@tuiasi.ro



THANK YOU !