FACULTY OF ECONOMICS AND BUSINESS ADMINISTRATION

A Brief Introduction to MATLAB

Gerdie Everaert

GHENT UNIVERSITY

What is MATLAB?

- The name MATLAB stands for **matrix laboratory**
- MATLAB is a numerical computing environment and programming language. Its basic element is a **matrix**
- This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar language

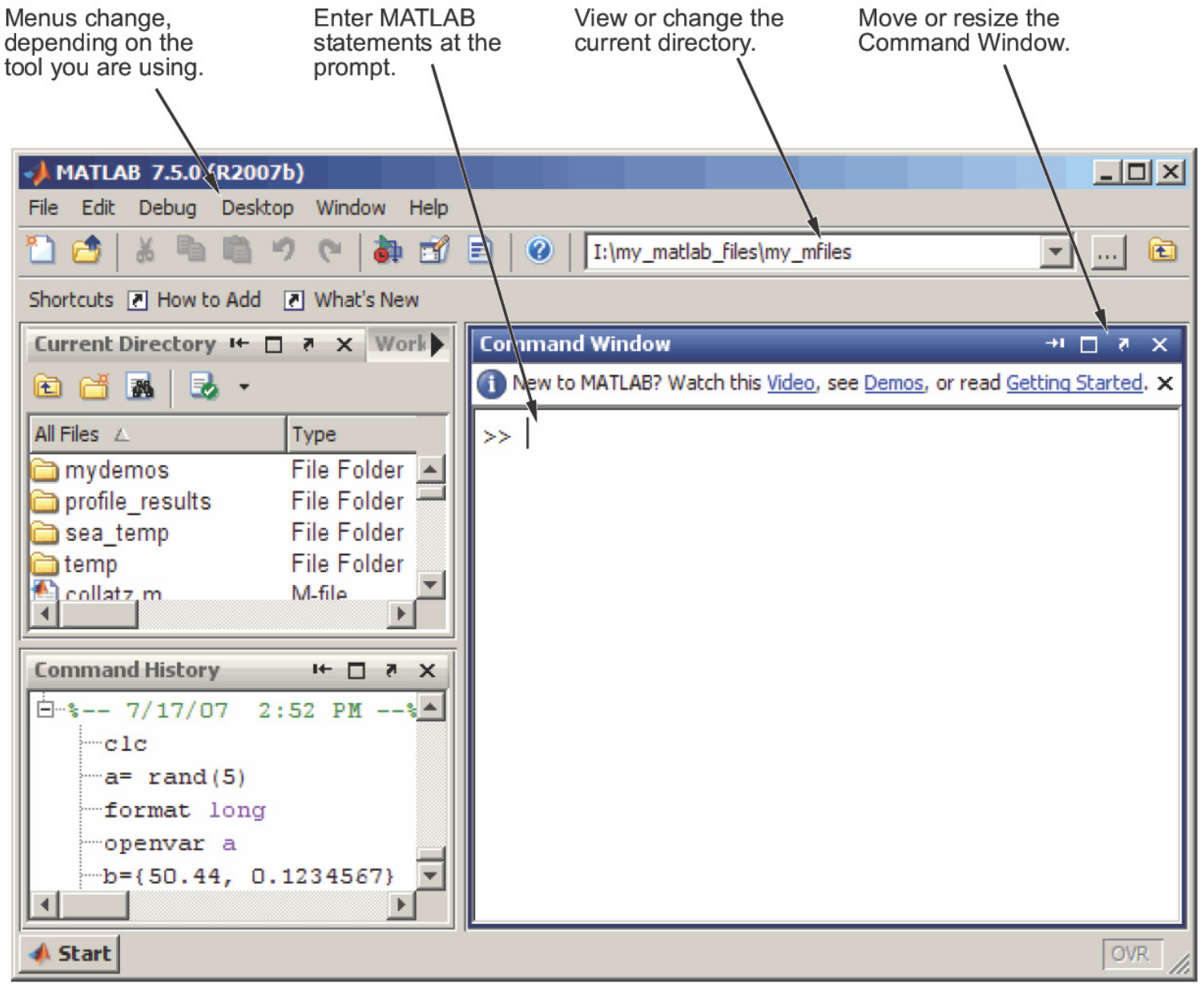
Why MATLAB?

- **Flexible tool** for data analysis, estimation, simulation, ...
  - Allows you to **implement state-of-the-art methodology** (which is not yet available in easy-to-use econometric software like EViews, SPSS, Microfit, STATA, ...)
  - You have perfect **control** over the implemented methodology
  - Has a number of add-on application-specific solutions called **toolboxes** which implement specialized methodology
  - Program files implementing a specific methodology can easily be **shared** with others
- Comparable to GAUSS, R, ...
- MATLAB is available on **Athena**

Getting started

- Starting MATLAB
  - The MATLAB desktop opens
- Typing in the Command Window
  - Try typing 3^2 - (5+4)/2 + 6 x 3 and then press ENTER
    MATLAB will answer **ans = 22.5**
  - Try typing test = 3^2 - (5+4)/2 + 6 x 3
    MATLAB will answer **test = 22.5** and will store the answer in the variable **test**
- Basic matrix operations
  - Entering matrices
  - Working with matrices

Check **MATLAB documentation** at
http://nl.mathworks.com/help/matlab/


Menus change, depending on the tool you are using.
Enter MATLAB statements at the prompt.
View or change the current directory.
Move or resize the Command Window.
MATLAB 7.5.0 (R2007b)
File Edit Debug Desktop Window Help
I:\my_matlab_files\my_mfiles
Shortcuts How to Add What's New
Current Directory
All Files Type
mydemos File Folder
profile_results File Folder
sea_temp File Folder
temp File Folder
collatz.m M-file
Command History
%-- 7/17/07 2:52 PM --%
clc
a= rand(5)
format long
openvar a
b={50.44, 0.1234567}
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
>>
Start
OVR

Entering Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways:

- Enter an explicit list of elements.
- Load matrices from external data files.
- Generate matrices using built-in functions.
- Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You only have to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered:

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

This matrix matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A. Now that you have A in the workspace, take a look at what makes it so interesting. Why is it magic?

sum, transpose, and diag

You are probably already aware that the special properties of a magic square have to do with the various ways of summing its elements. If you take the sum along any row or column, or along either of the two main diagonals, you will always get the same number. Let us verify that using MATLAB. The first statement to try is

```
sum(A)
```

MATLAB replies with

```
ans =
    34    34    34    34
```

How about the row sums? MATLAB has a preference for working with the columns of a matrix, so one way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result. For an additional way that avoids the double transpose use the dimension argument for the sum function.

So

```
A'
```

produces

```
ans =
    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```

```
sum(A')'
```

produces a column vector containing the row sums

```
ans =
     34
     34
     34
     34
```

The sum of the elements on the main diagonal is obtained with the `sum` and the `diag` functions:

```
diag(A)
```

produces

```
ans =
    16
    10
     7
     1
```

```
sum(diag(A))
```

produces

```
ans =
    34
```

Subscripts

The element in row `i` and column `j` of `A` is denoted by `A(i,j)`. For example, `A(4,2)` is the number in the fourth row and second column. For our magic square, `A(4,2)` is 15. So to compute the sum of the elements in the fourth column of `A`, type

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

This produces

```
ans =
     34
```

but is not the most elegant way of summing a single column.

If you try to use the value of an element outside of the matrix, it is an error:

```
t = A(4,5)
Index exceeds matrix dimensions.
```

On the other hand, if you store a value in an element outside of the matrix, the size increases to accommodate the newcomer:

```
X = A;
X(4,5) = 17

X =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

Subscript expressions involving colons refer to portions of a matrix:

```
A(1:k,j)
```

is the first k elements of the jth column of A. So

```
sum(A(1:4,4))
```

computes the sum of the fourth column. But there is a better way. The colon by itself refers to *all* the elements in a row or column of a matrix and the keyword end refers to the *last* row or column. So

```
sum(A(:,end))
```

computes the sum of the elements in the last column of A:

```
ans =
     34
```

Operators

Expressions use familiar arithmetic operators and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division (described in "Matrices and Linear Algebra" in the MATLAB documentation)
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

The multiplication symbol, *, denotes the *matrix* multiplication involving inner products between rows and columns. Multiplying the transpose of a matrix by the original matrix also produces a symmetric matrix:

```
A'*A

ans =
    378    212    206    360
    212    370    368    206
    206    368    370    212
    360    206    212    378
```

The determinant of this particular matrix happens to be zero, indicating that the matrix is *singular*:

```
d = det(A)

d =

     0
```

Since the matrix is singular, it does not have an inverse. If you try to compute the inverse with

```
X = inv(A)
```

you will get a warning message:

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 9.796086e-018.
```

Arrays

When they are taken away from the world of linear algebra, matrices become two-dimensional numeric arrays. Arithmetic operations on arrays are done element by element. This means that addition and subtraction are the same for arrays and matrices, but that multiplicative operations are different. MATLAB uses a dot, or decimal point, as part of the notation for multiplicative array operations.

The list of operators includes

+	Addition
-	Subtraction
.*	Element-by-element multiplication
./	Element-by-element division
.\	Element-by-element left division
.^	Element-by-element power
.'	Unconjugated array transpose

If the Dürer magic square is multiplied by itself with array multiplication

```
A.*A
```

the result is an array containing the squares of the integers from 1 to 16, in an unusual order:

```
ans =
   256     9     4   169
    25   100   121    64
    81    36    49   144
    16   225   196     1
```

Generating Matrices

MATLAB provides four functions that generate basic matrices.

`zeros`	All zeros
`ones`	All ones
`rand`	Uniformly distributed random elements
`randn`	Normally distributed random elements

Here are some examples:

```
Z = zeros(2,4)
Z =
     0     0     0     0
     0     0     0     0

F = 5*ones(3,3)
F =
     5     5     5
     5     5     5
     5     5     5

N = fix(10*rand(1,10))
N =
     9     2     6     4     8     7     4     0     8     4
```

```
R = randn(4,4)
R =
    0.6353    0.0860   -0.3210   -1.2316
   -0.6014   -2.0046    1.2366    1.0556
    0.5512   -0.4931   -0.6313   -0.1132
   -1.0998    0.4620   -2.3252    0.3792
```

Concatenation

Concatenation is the process of joining small matrices to make bigger ones. In fact, you made your first matrix by concatenating its individual elements. The pair of square brackets, [ ], is the concatenation operator. For an example, start with the 4-by-4 magic square, A, and form

```
B = [A  A+32; A+48  A+16]
```

The result is an 8-by-8 matrix, obtained by joining the four submatrices:

```
B =

    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

Deleting Rows and Columns

You can delete rows and columns from a matrix using just a pair of square brackets. Start with

```
X = A;
```

Then, to delete the second column of X, use

```
X(:,2) = []
```

This changes X to

```
X =
    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

Programming in MATLAB

- You can create your own programs using **M-files**, which are text files containing MATLAB code.
- Use the MATLAB Editor or another text editor to create a file containing the same statements you would type in the MATLAB command window.
- Save the file under a name that ends in '.m'.
- Run the program by typing the name of the M-file in the MATLAB command window, providing input and output variables.

Types of M-Files

M-files can be *scripts* that simply execute a series of MATLAB statements, or they can be *functions* that also accept input arguments and produce output.

MATLAB scripts:

- Are useful for automating a series of steps you need to perform many times.
- Do not accept input arguments or return output arguments.
- Store variables in a workspace that is shared with other scripts and with the MATLAB command line interface.

MATLAB functions:

- Are useful for extending the MATLAB language for your application.
- Can accept input arguments and return output arguments.

Scripts

When you invoke a *script*, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like `plot`.

Functions

Functions are M-files that can accept input arguments and return output arguments. The names of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

Basic Parts of an M-File

This simple function shows the basic parts of an M-file. Note that any line that begins with % is not executable:

```
function f = fact(n)                          Function
definition line
% Compute a factorial value.                 H1 line
% FACT(N) returns the factorial of N,        Help text
% usually denoted by N!

% Put simply, FACT(N) is PROD(1:N).          Comment
f = prod(1:n);                                Function body
```

Function Arguments. If the function has multiple output values, enclose the output argument list in square brackets. Input arguments, if present, are enclosed in parentheses following the function name. Use commas to separate multiple input or output arguments. Here is the declaration for a function named `sphere` that has three inputs and three outputs:

```
function [x, y, z] = sphere(theta, phi, rho)
```

If there is no output, leave the output blank

```
function printresults(x)
```

or use empty square brackets:

```
function [] = printresults(x)
```

The variables that you pass to the function do not need to have the same name as those in the function definition line.

The Function or Script Body

The function body contains all the MATLAB code that performs computations and assigns values to output arguments. The statements in the function body can consist of function calls, programming constructs like flow control and interactive input/output, calculations, assignments, comments, and blank lines.

Flow Control

MATLAB functions that provide program control

- conditional control: if, else and elseif
- loop control: for, while, continue and break

if, else, and elseif

The `if` statement evaluates a logical expression and executes a group of statements when the expression is *true*. The optional `elseif` and `else` keywords provide for the execution of alternate groups of statements. An `end` keyword, which matches the `if`, terminates the last group of statements. The groups of statements are delineated by the four keywords—no braces or brackets are involved.

for

The `for` loop repeats a group of statements a fixed, predetermined number of times. A matching `end` delineates the statements:

```
for n = 3:32
   r(n) = rank(magic(n));
end
r
```

The semicolon terminating the inner statement suppresses repeated printing, and the `r` after the loop displays the final result.

It is a good idea to indent the loops for readability, especially when they are nested:

```
for i = 1:m
   for j = 1:n
      H(i,j) = 1/(i+j);
   end
end
```

while

The `while` loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching `end` delineates the statements.

continue

The `continue` statement passes control to the next iteration of the `for` loop or `while` loop in which it appears, skipping any remaining statements in the body of the loop. The same holds true for `continue` statements in nested loops. That is, execution continues at the beginning of the loop in which the `continue` statement was encountered.

break

The `break` statement lets you exit early from a `for` loop or `while` loop. In nested loops, `break` exits from the innermost loop only.

Tips and Tricks

- TRY THINGS, this is how you learn!
- Use the help files or just google
- Write and check program line by line
- Make your programs as self-reliant as possible
- Use 'structure arrays' to structure data and results
  - results.OLS.coefs
  - results.OLS.R2
  - results.OLS.residuals
  - results.GLS.coefs
  - ...

- CTRL–C kills the current command if you are in an infinite loop :-)