

MATLAB: Introduction

Part 1

Bruno Abreu Calfa

Last Update: January 29, 2014

Table of Contents

Outline

Contents

1	What is MATLAB?	1
2	MATLAB Windows	2
3	MATLAB as a Calculator	3
4	MATLAB Classes	3
5	Scripts and Functions	6
5.1	Writing MATLAB Programs	6
5.2	Code Cells and Publishing	6

1 What is MATLAB?

A powerful tool!

- **MATLAB** stands for *Matrix Laboratory*
- Enhanced by *toolboxes* (specific routines for an area of application)
 - Optimization
 - Statistics
 - Control System
 - Bioinformatics
 - ...
- Excellent for numerical computations
- Commonly regarded as a ‘Rapid Prototyping Tool’
- Used in industry and academia

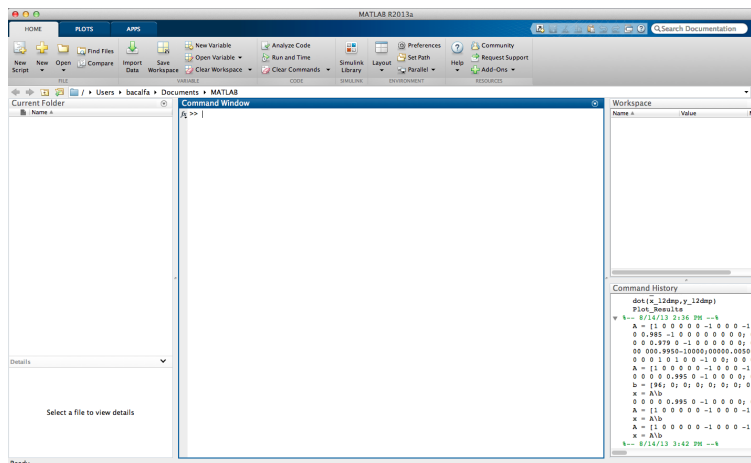
Help with MATLAB?

- MATLAB's Help
- Google
- A book about MATLAB

2 MATLAB Windows

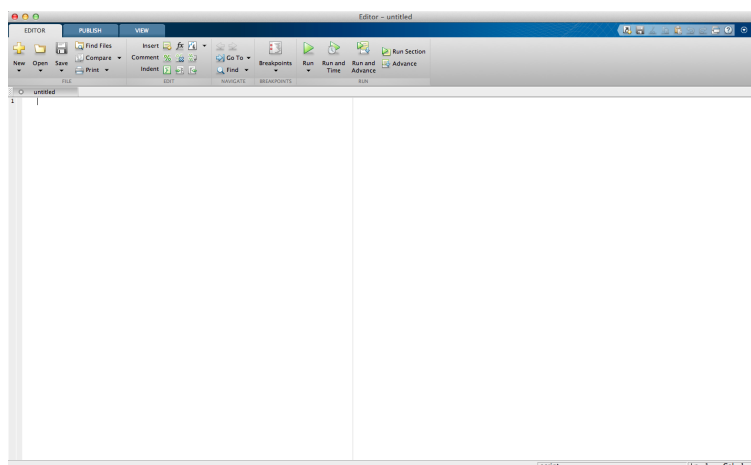
Main Window

- Command Window (prompt >>)
- Current Directory
- Workspace (contains variables stored in memory)
- Help Menu



Editor Window

- Click on “New Script” in the ribbon to open the Editor Window
 - Window Menu (Tile)
 - Debug Menu (Run, Step In, Step Out...)
 - Cell Menu (Cell Mode)



3 MATLAB as a Calculator

Basic Operators

- MATLAB supports the following mathematical operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation

- Some examples:

```
- >> 1 + 2
- >> 2*3 + 4
- >> 4/3 - 3/4 + 2^3
```

Basic Operators

- Beware of operator *precedence* rules!

```
- >> 2*3 + 4
- >> 2*(3 + 4)
- >> 4.2/3 + 1.2
- >> 4.2/(3 + 1.2)
- >> 15/(2 + 3)*(4 - 1)
- >> 15/((2 + 3)*(4 - 1))
- >> 2^3/2
- >> 2^(3/2)
```

- Use parentheses to enforce the desired order

4 MATLAB Classes

All Matrices!

- “Everything” in MATLAB is a matrix
 - A *scalar* is a 1-by-1 matrix
 - A 1D array of n elements can be a n -by-1 (column vector) or a 1-by- n (row vector) matrix
 - A *string* of n characters is a 1-by- n matrix
 - ...
- Some MATLAB *classes*:
 - `double` (Double-precision floating-point number array) (default)
 - `single` (Single-precision floating-point number array)
 - `char` (Character array)
 - `cell` (Cell array)
 - `struct` (Structure array)
 - `function_handle` (Array of values for calling functions indirectly)

Scalar Variables: 1-by-1 Matrices!

- Use the '=' sign for *assignment*
 - >> a = 1 % The scalar variable 'a' stores the value 1
 - >> % This is a *comment* and is ignored by the interpreter
 - >> sin(a) % Sine of 'a' = 0.8415
 - >> sin(a); % ';' avoids displaying the result of the command
 - >> size(a) % = [1,1], i.e. 1-by-1 matrix
 - >> b = a + 2 % b = 3
 - >> c = cos(b*pi/.2) % 'pi' is the builtin constant π
 - >> d = rand % A random scalar
- Use the commands `who` or `whos` to list the variables defined in the Workspace
- Other common functions are available: `exp`, `tan`, `sinh`, `acos`, ...

1D Arrays: Real Vectors (or Matrices!)

- Use `[... , ...]` or `[... ...]` for *horizontal stacking* and `[... ; ...]` for *vertical stacking*
 - >> v1 = [1 2 3] % Row vector, same as v1 = [1, 2, 3]
 - >> v2 = [4;5;6] % Column vector
 - >> v3 = v2 - v1 % Error! Incompatible *matrix* dimensions
 - >> v3 = v2 - v1.' % Transpose a real *matrix* with `.'`
 - >> v4 = v1*v2 % Dot product, also `dot(v1, v2)`
 - >> v7 = .1*v1 % Scalar-vector multiplication
 - >> v7(1) % First element of array 'v7'
 - >> v8 = exp(v7) % Element-wise operation
 - >> sz8 = size(v8) % = [1 3]
 - >> v9 = rand(1,5) % Random 1-by-5 array
 - >> p = prod(v1) % Product of elements = 6

2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise
 - >> m1 = [1 2 3; 4 5 6] % 2-by-3
 - >> m1p = [1,2,3; 4,5,6] % 2-by-3, same as m1
 - >> m2 = rand(2,3) % Random 2-by-3 matrix
 - >> m3 = m1 + m2 % Matrix addition
 - >> m4 = m1*m2 % Error! Dimensions don't agree
 - >> m4 = m1*m2.' % OK! Transpose a real matrix with `.'`
 - >> m4(1,2) % Element in row 1 and column 2 of 'm4'
 - >> len4 = length(m4) % Size of longest dimension
 - >> m5 = m3/2 % Element-wise division
 - >> m6 = tan(m5) % Element-wise operation

Element-wise Operations

- The following are element-wise mathematical operators

Operator	Operation
<code>.*</code>	Element-wise Multiplication
<code>./</code>	Element-wise Division
<code>.^</code>	Element-wise Exponentiation

- More examples:

```
- >> v1 = [1 2 3] % 1-by-3
- >> v2 = [2 4 6] % 1-by-3
- >> v3 = v1.*v2 % = [2 8 18]
- >> v4 = v2./v1 % = [2 2 2]
- >> v5 = v1.^v4 % = [1 4 9]
- >> m1 = [0 1; 1 0] % 2-by-2
- >> m2 = [3 5; 7 2] % 2-by-2
- >> m3 = m1.*m2 % = [0 5; 7 0]
```

The Colon (:) Operator

- Use it extensively!

```
- >> v1 = 1:10 % Same as v1 = [1,2,3,...,10]
- >> v2 = 0:.1:1 % Same as v2 = [0,.1,.2,...,1]
- >> m1 = rand(5) % Random 5-by-5 matrix
- >> v3 = v1(5:end) % v3 = [5,6,7,8,9,10]
- >> v4 = m1(:,3) % 'v4' has the elements in column 3 of 'm1'
- >> v5 = m1(1,:) % 'v5' has the elements in row 1 of 'm1'
```

- Do not forget `linspace` to generate linearly spaced vectors!

```
- >> v6 = linspace(0,1,10) % = [0,0.1111,0.2222,...,1]
- >> v7 = linspace(0,10,5) % = [0,2.5,5,7.5,10]
- >> v8 = linspace(0,1,100) % = [0,0.0101,0.0202,...,1]
```

Strings: char Arrays

- Remember that strings are also *matrices* in MATLAB!

```
- >> str1 = 'Hello, world!' % A simple string
- >> sz1 = size(str1) % = 1-by-13
- >> a = rand; str2 = ['a = ' num2str(a)] % Horizontal stacking concatenates strings
- >> b = str2num('500')*rand % MATLAB has many handy *2* functions!
```

- Format your strings with `sprintf`

```
- >> sprintf('Volume of reactor = %.2f', 10.23451) % Floating-point format with two
decimal digits
- >> str3 = sprintf('A large number = %e', rand*10^5) % Exponential notation format
- >> sprintf('Another large number = %g', rand*10^5) % More compact format between %e and %f
```

function_handle (@) Class

- Used in calling functions indirectly
 - `>> Sin = @sin; % The variable 'Sin' points to the function 'sin'`
 - `>> Sin(pi) % Evaluates the sine of π`
- Can be used to create 'anonymous functions'
 - `>> myfun = @(x) 1./(x.^3 + 3*x - 5) % Anonymous function`
 - `>> quad(myfun,0,1) % Adaptive Simpson quadrature to integrate 'myfun'`

5 Scripts and Functions

5.1 Writing MATLAB Programs

M-Files

- The file with source code is called M-File (*.m)
- **Scripts:** No input and no output arguments. Contain a series of commands that may call other scripts and functions.
- **Functions:** Accept input and output arguments. Usually called program *routines* and have a special definition syntax.
- Inside scripts and functions you may use programming statements, such as [flow](#), [loop](#), and [error control](#)
- Open the Editor Window and start coding!

Function M-Files

- General form:

```
function [out1, out2, ...] = funname(in1, in2, ...)
    statement
    ...
end % Optional
```

- Example:

```
function Z = virialgen(P,Pc,T,Tc,omega)
Pr = P/Pc;
Tr = T/Tc;
[B0,B1] = virialB(Tr);
Z = 1 + Pr/Tr*(B0 + omega*B1);

function [B0,B1] = virialB(Tr)
B0 = 0.083 - 0.422/Tr^1.6;
B1 = 0.139 - 0.172/Tr^4.2;
```

5.2 Code Cells and Publishing

Code Cells

- Allow you to divide your M-files into sections (cells)
- Enable you to execute cell by cell
- Foundations for *publishing* your M-file to HTML, PDF, and other formats
- To begin a code cell, type `%%` at the beginning of a line
- The first line after the `%%` is the **title** of the code cell
- The next lines starting with `%` are a description of the code cell
- Place your code in the next lines
- A new code cell starts at the next `%%` at the beginning of a line

Code Cells: Example

- Simple example:

```
%% 99-999: Homework 1
% Bruno Abreu Calfa

%% Problem 1
x = linspace(0,1);
y = sin(x.^2).*exp(-x.*tan(x));
plot(x,y);

%% Problem 2
a = 0;
b = 1;
f = @(t) exp(-t.^2);
intf = quad(f,a,b);
sprintf('Integral of f from %g to %g = %g',a,b,intf)
```

Publishing your Code

- Saves output of your code to a specific file type
- Formats available:

File Format	Description
doc	Microsoft Word ¹
latex	L ^A T _E X ¹
ppt	Microsoft Powerpoint ¹
xml	Extensible Markup Language
pdf	Portable Document Format
html	Hypertext Markup Language

- MATLAB evaluates your M-file and generates the output
- To publish your M-file, go to “Publish” tab