MATLAB Introduction Course: Lecture 1

Øivind K. Kjerstad

26. September 2014

Course information

Contact information

Office: C2.093
e-mail: oivind.k.kjerstad@ntnu.no
Help: send e-mail

- Lecture 1: Today
  - Scripts, Variables, Variable manipulation, Simple plotting, Help
- Lecture 2: 3. October
  - Functions, Loops and conditions, Advanced plotting, Vectorization
- Lecture 3: 10. October
  - Linear algebra, Polynomials, Derivation, Integration, Solving dif. eqs.
- Lecture 4: 17. October
  - Probability and statistics, Datastructures, Animations, Debugging
- Lecture 5: TBD
  - Symbolic computation, Simulink, I/O, GUI

TOC

MATLAB basics

- MATLAB can be thought of as a super-powerful graphing calculator
  - Lots and lots of built in functions to help you solve a wide range of engineering problems
  - Made for data processing and visualization
- It is also a programming language
  - MATLAB is an interpreted language
  - Commands executed line by line

You will need to know basic programming!

Once you know how MATLAB works and some basic programming functions that we will cover in these lectures you will be able to do alot

NB! Today, we focus mainly on using MATLAB as a calculator

Why do you need to know MATLAB?

- Thesis and project
- Course projects
- Homework

Most of you will work in similar frameworks later!

Mathematica, GNU Octave, Sage, R, Scilab, Maxima, Python (SciPy, Numpy, python(x,y)), Maple, Microsoft Mathematics, wxMaxima
+ + +

MATLAB is learning by doing!

Download and install the software

- Get it at $\rightarrow$ \\progdist.ntnu.no\progdist

Read more about installation at

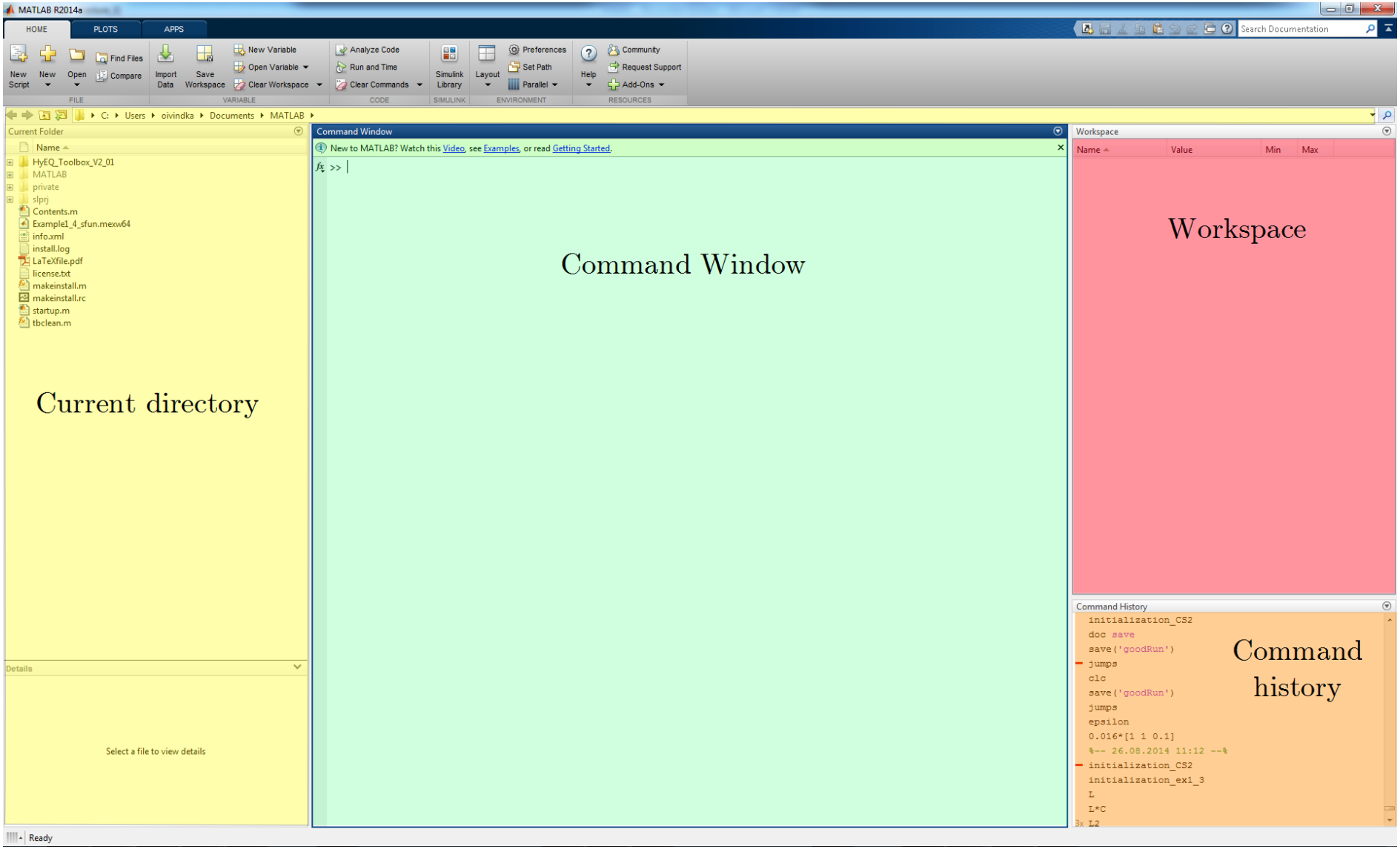
https://innsida.ntnu.no/wiki/-/wiki/English/Matlab+for+students

- Windows, Mac OS X and Unix/Linux clients
  - For most things all OS are good

License

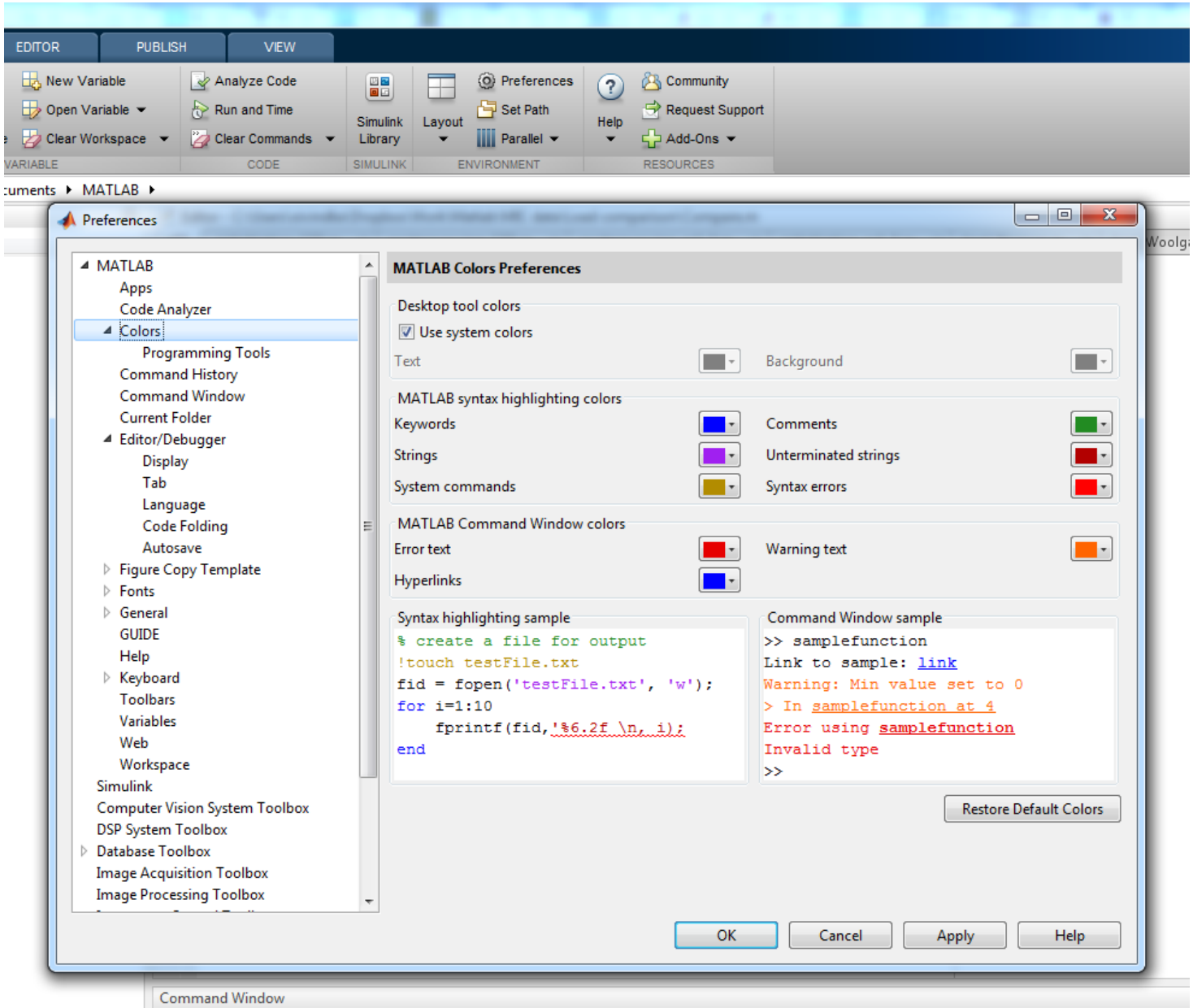
Read the readme file at progdist! There is everything you need to know

- Be patient! It takes a while..



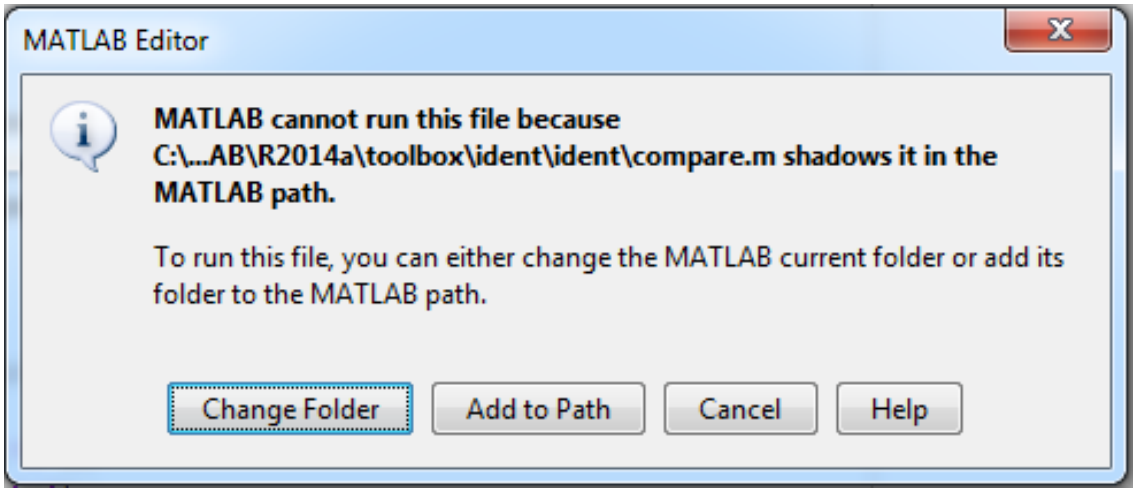
There is also a text/script editor (More on that later)

Customization



Making folders

- Use folders to keep your work organized
- To see scripts and functions outside the current directory, they should be in the Path. Use File $\rightarrow$ Set Path to add folders to the path
- MATLAB will give an error or ask to change folder if it is set incorrectly



help and doc functionality

The MATLAB help function

- The **most** important function for learning MATLAB on your own
- To get info on how to use a function:
  - `>> help ...`

- Notice that `help` lists related functions at the bottom
- To get a nicer version of `help` with examples and easy-to-read descriptions:
  - `>> doc ...`
- To search for a function:
  - `>> help` + tab
  - `>> doc` + tab

Other handy resources

- MathWorks
  - Interactive tutorials, Videos, Documentation, etc.
- MATLAB Central
  - Forum, File exchange, Blogs, etc.
- YouTube
  - Approx 300k results for MATLAB
- Google
  - Lots of other independent sites and forums
- IMT tutorial
  - http://www.ivt.ntnu.no/imt/software/matlab

The community is huge!

TOC

Variables and variable types

In MATLAB a variable is a name given to some data, i.e.

```
>> myVariable = 1
```

- MATLAB is a weakly typed language
  - No need to initialize variables!
- MATLAB supports various types, the most often used are
  - 64-bit double (default)
  - 16-bit char
  - Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc.

Don't worry too much about variable types!

Most variables you'll deal with will be vectors or matrices of doubles or chars. In most cases MATLAB will figure it out for you.

Naming variables

- To create a variable, simply assign a value to a name:
  - `>> myVariable = 9000`
  - `>> myString = 'hello world'`
- Variable names
  - The first character **must** be a letter. After that, any combination of letters, numbers and underscore works
  - Case sensitivity! (`myVar` is different from `myvar`)
- Built-in variables. Don't use these names!
  - `i` and `j` can be used to indicate complex numbers
    - ⋆ Be aware! These are often used as iterators in loops. If so, they will loose the complex number operator
  - `pi` has the value 3.1415926...
  - `ans` stores the last unassigned value
  - `Inf` and `-Inf` are positive and negative infinity
  - `NaN` represents 'Not a Number'

Scalars

- A variable can be given a value explicitly
  - `>> a = 10`
  - It will show up in the workspace
- Or as a function of explicit values and existing variables
  - `>> c = 1.3*45-2*a`
- To suppress output, end the line with a semicolon
  - `>> d = 1.25;`

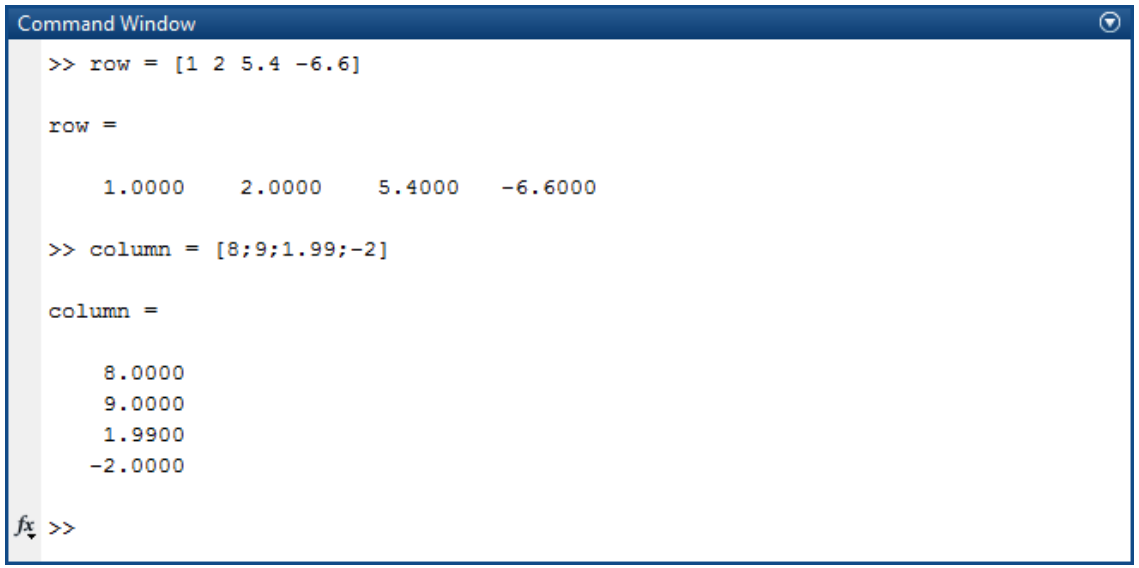
Arrays

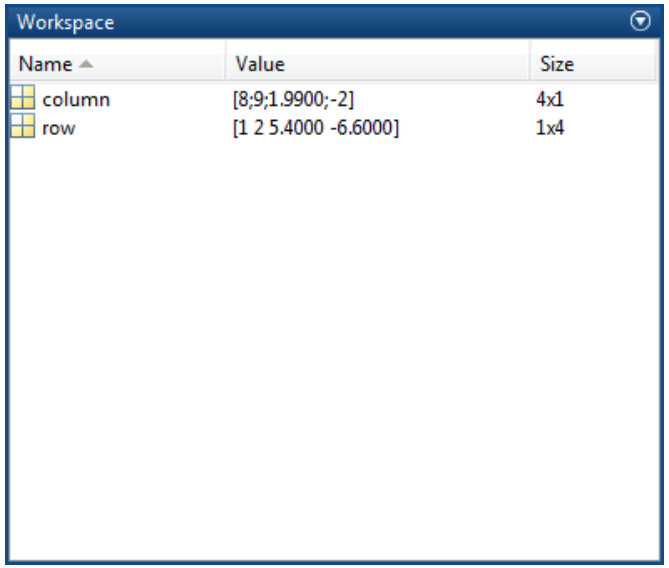
- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays:
  - Matrix of numbers (either double or complex)
  - Cell array of objects (more advanced data structure)
    - ⋆ We will get back to Cell array in a later lecture

MATLAB arrays makes matrices and vectors easy to work with!

Row and column vectors

- Row vector: comma or space separated values between brackets
  - `>> row = [1 2 5.4 -6.6]`
  - `>> column = [8;9;1.99;-2]`
- Command window and workspace:





- Strings are character vectors
  - `>> myStr = ['Hello world' '2014']`

Size and length

- You can tell the difference between a row and a column vector by:
  - Looking in the workspace
  - Displaying the variable in the command window
  - Using the `size` function
- To get a vector's length, use the `length` function

Matrices

- You create matrices just like vectors, element by element

Examples

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

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

```
>> B = [0.1 -54 5;1 99 23; 123 1.1 1.001]
```

$$B = \begin{bmatrix} 0.1 & -54 & 5 \\ 1 & 99 & 23 \\ 123 & 1.1 & 1.001 \end{bmatrix}$$

Matrices

- By concatenating vectors or matrices (dimension matters)

Example

```
>> a = [1 2]      row
>> b = [3 4]      row
>> c = [5;6]      column
```

$$a = \begin{bmatrix} 1 & 2 \end{bmatrix} \quad b = \begin{bmatrix} 3 & 4 \end{bmatrix} \quad c = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

```
>> D = [a;b]
>> E = [D c]
>> F = [[E E];[a b a]]
```

$$D = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad E = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{bmatrix} \quad F = \begin{bmatrix} 1 & 2 & 5 & 1 & 2 & 5 \\ 3 & 4 & 6 & 3 & 4 & 6 \\ 1 & 2 & 3 & 4 & 1 & 2 \end{bmatrix}$$

Save, Load, and Clear

- Use `save` to save variables to a file
  - `>> save myFile a b`
    - ⋆ saves variables `a` and `b` to the file `myFile.mat`
  - `myFile.mat` file is saved in the current directory
  - Default working directory is \MATLAB
- Use `clear` to remove variables from environment
  - `>> clear a b`
    - ⋆ the variables `a` and `b` are removed form the workspace
- Use `load` to load variable bindings into the workspace
  - `>> load myFile`
    - ⋆ the variables `a` and `b` are back
- Can do the same for entire environment
  - `>> clear all`
  - See `help save`
  - See `help load`

TOC

Basic Scalar Operations

- Arithmetic operations (+,-,*,/)
  - `>> 4/20`
  - `>> 2*2`
  - `>> (1+i)*(2-4i)`
- Exponentiation (^)
  - `>> 4^2`
  - `>> (3+4j)^2`
- Complicated expressions, use parentheses
  - `>> ((2+3)*3)^0.1`
- Multiplication is NOT implicit given parentheses
  - `>> 3(1+0.7)` $\rightarrow$ gives an error
- To clear the command window
  - `>> clc`

Built-in Functions

- MATLAB has an **huge** library of built-in functions
- You call them using parentheses (passes the parameters to the function)

Some examples

>> `sqrt()` *Square root*
>> `log()` *Natural logarithm*
>> `log10()` *10 base logarithm*
>> `cos()` *Cosine*
>> `sin()` *Sine*
>> `exp()` *The exponential*
>> `round()` *Rounding the input number to closest integer*
>> `floor()` *Rounding down the input*
>> `ceil()` *Rounding up the input*
>> `angle()` *The phase angle*
>> `abs()` *The absolute value*

Transpose

- The transpose operators turns a column vector into a row vector and vice versa
  - `>> a = [1 2 3 4+i]`
  - `>> transpose(a)`
  - `>> a'`
  - `>> a.'`
- The `'` gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers
- For vectors of real numbers `.'` and `'` give same result

Addition and Subtraction

- Addition and subtraction are element-wise; sizes must match!
  - Unless one is a scalar
- The following would give an error
  - `>> c = row + column`
- Use the transpose to make sizes compatible
  - `>> c = row' + column`
  - `>> c = row + column'`
- Can sum up or multiply elements of vector
  - `>> s = sum(row)`
  - `>> p = prod(row)`

Element-Wise Functions

- All the functions that work on scalars also work on vectors
  - `>> t = [1 2 3]`
  - `>> f = exp(t)`
    - ⋆ is the same as
  - `>> f = [exp(1) exp(2) exp(3)]`
- If in doubt, check a function's help or doc
  - `help ...`
- Operators (`* / ^`) have two modes of operation
  - element-wise
  - standard

Operators: element-wise

- To do element-wise operations, use a dot in front: (`.*` `./` `.^`)
  BOTH dimensions must match (unless one is scalar)

Example

```
>> a = [1 2 3];
>> b = [4;2;1];
```

`>> a.*b, a./b, a.^b` → Error! dimension must agree

`>> a.*b', a./b', a.^(b')` *These are correct!*

Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (`*`) is either a dot-product or an outer-product
  - From linear algebra: inner dimensions must MATCH!
- Standard exponentiation (`^`) can only be done on square matrices or scalars
- Left and right division (`/ \`) is same as multiplying by inverse
  - We will get back to this later in the course

Automatic Initialization

Initialize a vector of `ones`, `zeros`, or `random` numbers:

- `>> o = ones(1,10)`
  - row vector with 10 elements, all 1
- `>> z = zeros(23,1)`
  - column vector with 23 elements, all 0
- `>> r = rand(4,5)`
  - row vector with 45 elements (uniform [0,1])
- `>> n = nan(1,6)`
  - row vector of NaNs
  - Useful for representing uninitialized variables

The general function call for these functions are

`>> myVector = ones(N,M)` *May be any other of the above*

where N is the number of rows, and M is the number of columns

Automatic Initialization

- To initialize a linear vector of values use `linspace`
  - `>> a = linspace(0,10,5)`
    - ★ Starts at 0, ends at 10 (inclusive), 5 values
- Can also use colon operator (`:`)
  - `>> b = 0:2:10`
    - ★ Starts at 0, increments by 2, and ends at or before 10
    - ★ Increment can be decimal or negative
  - `>> c = 1:5`
    - ★ If increment isn't specified, default is 1
- To initialize logarithmically spaced values use `logspace`
  - similar to `linspace`, see `help`

Vector Indexing

- **MATLAB indexing starts with 1, not 0!**
  - Very important! Often confusing since other programming languages starts with 0

Example: $a = \begin{bmatrix} 0.2 & 10 & -3 & 5435 \end{bmatrix}$

```
>> a(1) = 0.2
>> a(2) = 10
>> a(3) = -3
>> a(4) = 5435
```

- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector

You may also

```
>> a(2:3) = [10 -3]
>> a(1:end-1) = [0.2 10 -3]
>> a([1 2 4]) = [0.2 10 5435]
```

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)
  - using linear indices (as if matrix is a vector)

Example: subscripts or linear indices

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
>> A(1,1) = 1 subscripts
>> A(1,2) = 2 subscripts
>> A(3,2) = 8 subscripts
>> A(1) = 1 linear indices
>> A(2) = 4 linear indices
>> A(9) = 9 linear indices
```

Matrix Indexing

- To select rows or columns of a matrix, use `:`

Example: more indexing

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
>> A(:,1:2)
```

$$= \begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$$

```
>> A([1 3 2],[1 3])
```

$$= \begin{bmatrix} 1 & 3 \\ 7 & 9 \\ 4 & 6 \end{bmatrix}$$

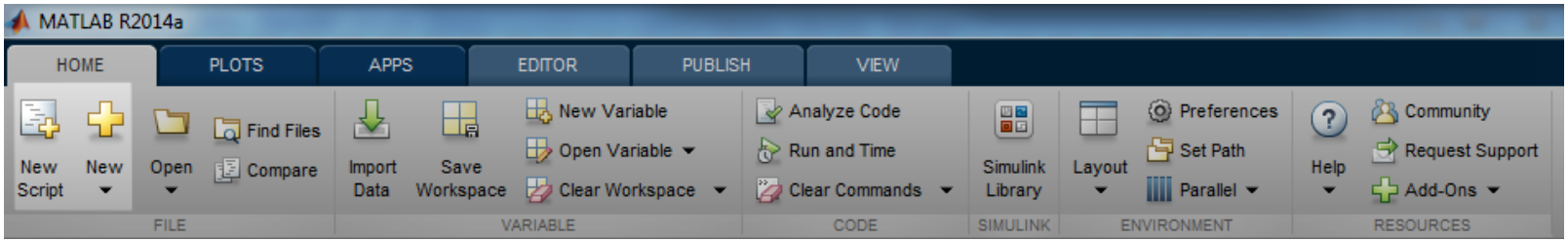
Advanced Indexing

- MATLAB contains functions to help you find desired values within a vector or matrix
  - `>> vec = [5 3 1 9 7]`
- To get the minimum value and its index:
  - `>> [minVal minInd] = min(vec)`
    - ★ `max` works the same way
- To find any the indices of specific values or ranges
  - `ind = find(vec == 9)`
  - `ind = find(vec > 2 & vec < 6)`
    - ★ you may have very complex `find` expressions
- To convert between subscripts and indices, use `ind2sub`, and `sub2ind`. Look up `help`to see how to use them

TOC

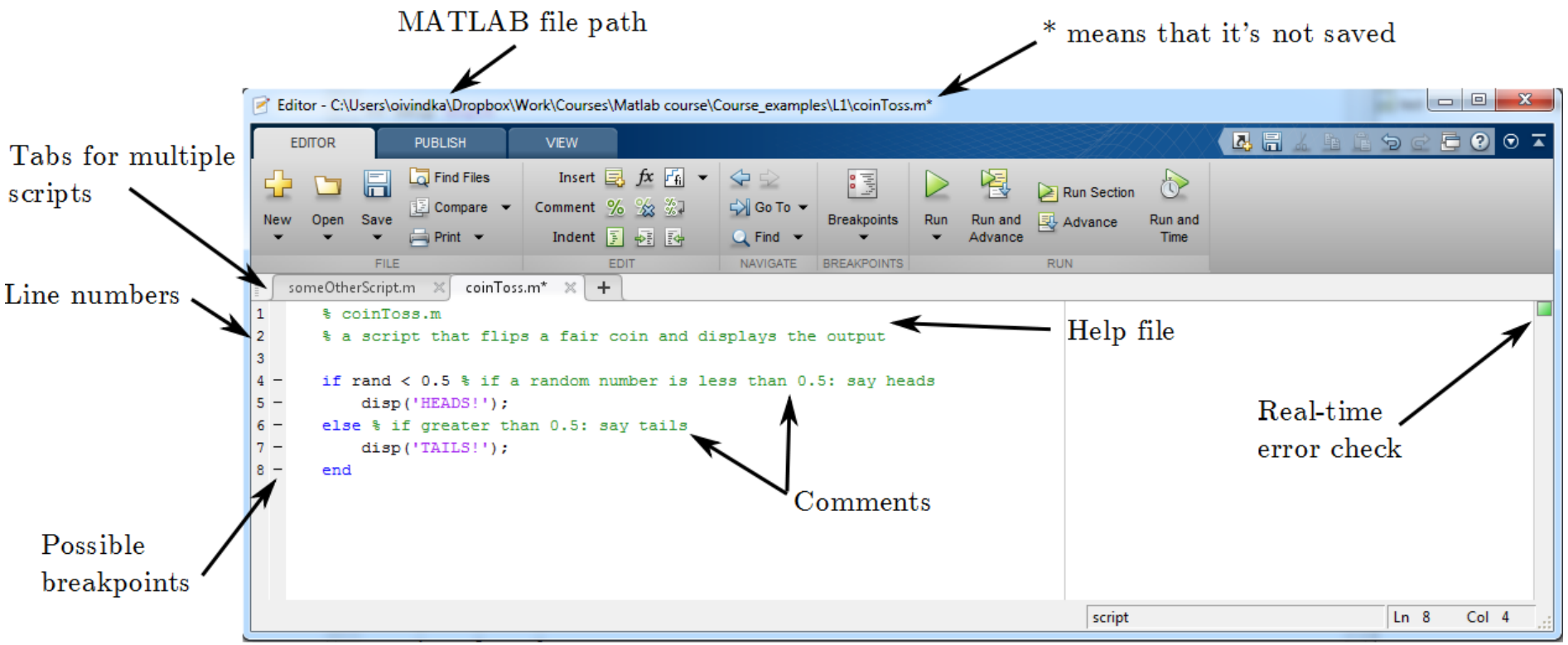
Overview

- Scripts are
  - collection of commands executed in sequence
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)
- To create an MATLAB file from command-line
  - `>> edit helloWorld.m`
- or click either of these



- You may also create new scripts from the current folder view

The Editor




```
Command Window
>> help coinToss
  coinToss.m
  a script that flips a fair coin and displays the output

>> coinToss
TAILS!
>>
```

Some Notes on Scripting

- **Comment your code!**
  - Anything following a % is seen as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later
- Note that scripts are somewhat static, since there is no input and no explicit output
  - This will be handled by functions, which we will get back to next week
- All variables created and modified in a script exist in the workspace even after it has stopped running

TOC

Basic Plotting

- Plotting is visualization of data in a figure
- `plot` generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points
- x and y vectors must be same size or else you'll get an error
  - `>> plot([1 2],[1 2 3])` → *Error!*
  - `plot` can be heavily modified
    - ★ We will get back to that next week

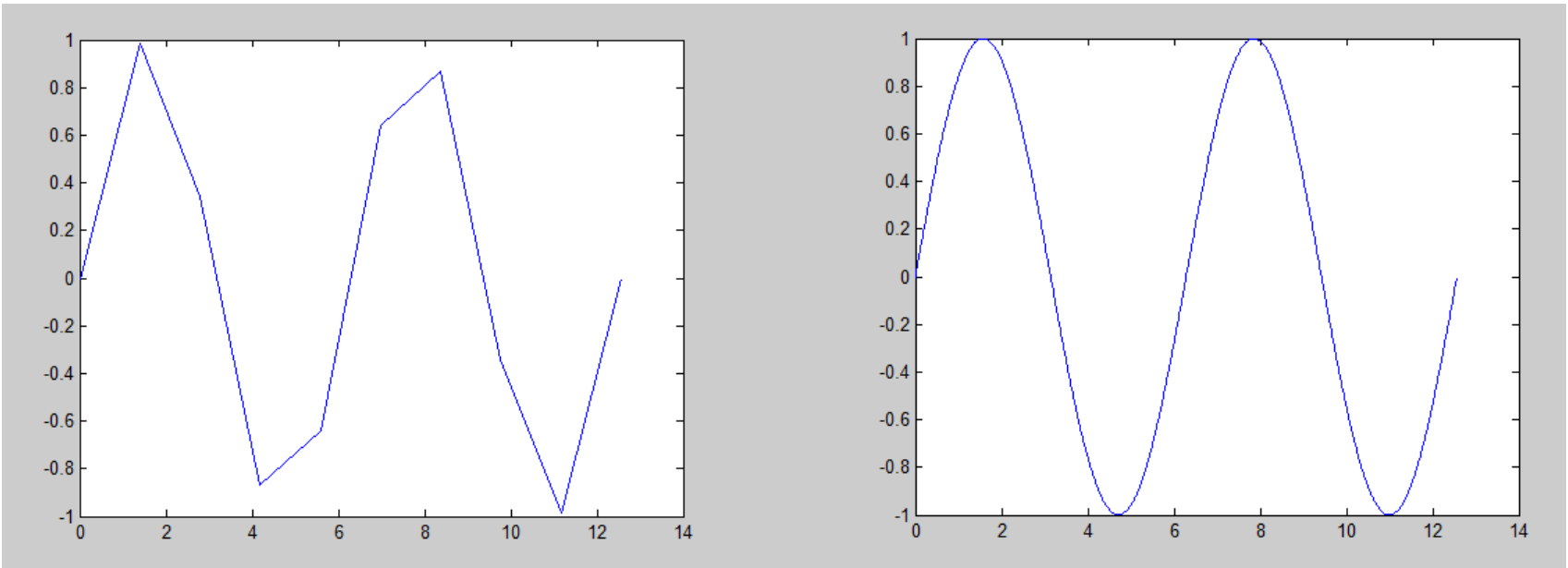
MATLAB makes visualization easy!

Plot example

Example data

```
>> x_low = linspace(0,4*pi,10);
>> y_low = sin(x_low);

>> x_high = linspace(0,4*pi,1000);
>> y_high = sin(x_high);
```



```
>> THE END
```

Some simple exercises to help you learn

Using MATLAB as a calculator

Work in the command window:

1. Create the following variables in the workspace: $a = \begin{bmatrix}1 & 2 & 3\end{bmatrix}$, $b = \begin{bmatrix}9 & 5 & 3\end{bmatrix}^\top$, $c = 7.51$, $D = \begin{bmatrix}8.1 & 6.3 \\ 1 & 100\end{bmatrix}$
2. Compute the expression $\Omega = (a^2 \cdot b)e^{0.1c}D^{-1}$
3. Save all workspace variables to a file named 'Lecture1Problem1.mat'
4. Clear the workspace
5. Load 'Lecture1Problem1.mat'
6. Rotate all values of Ω clockwise using indexes

Scripts and plotting

Create a script called 'GasPriceAnalysis.m' that:

1. Load the demo dataset 'gas.mat'
2. Compute the mean, variance, and standard deviation of each of the price vectors and print these to the command window
3. Plot both price vectors together using `plot` and set different colors for the two curves
4. Set the x-axis text to 'Time [days]' and y-axis text to 'Price [USD]'

→ *Hint: use help/doc to look for functions and how to use them*