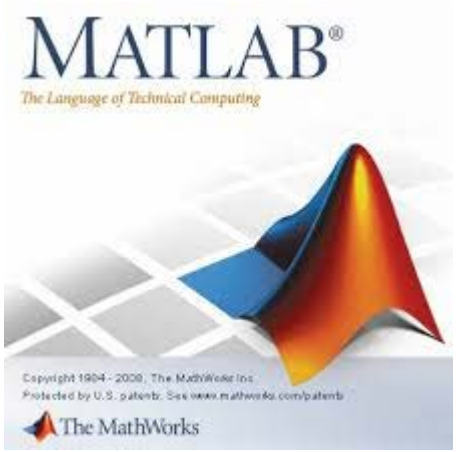
# CS 383 – Machine Learning

Matlab Intro

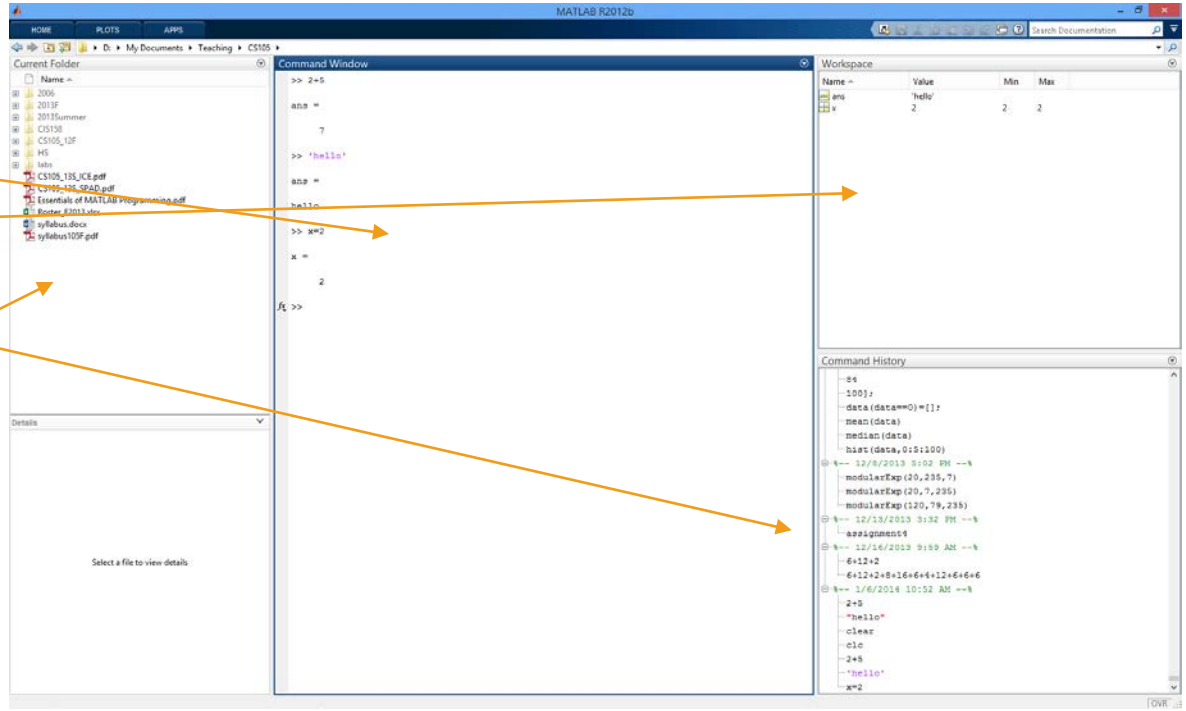


# What is MATLAB

- MATLAB (**Mat**rix **Lab**oratory) is a popular programming language for fields like
  - Research (general)
  - Engineering
- Built on C with a Java interface
  - But interpreted so slow ☹
- Why MATLAB?
  - Lots of built in capabilities (the basic installation is ~4 gigs)
  - Tons of built-in functions and optional add-on packages
  - Cross-platform (different installer, but code pretty much works in each)
  - Quick and easy to use/learn
  - Built-in stuff for matrix operations/manipulation

# MATLAB Interface

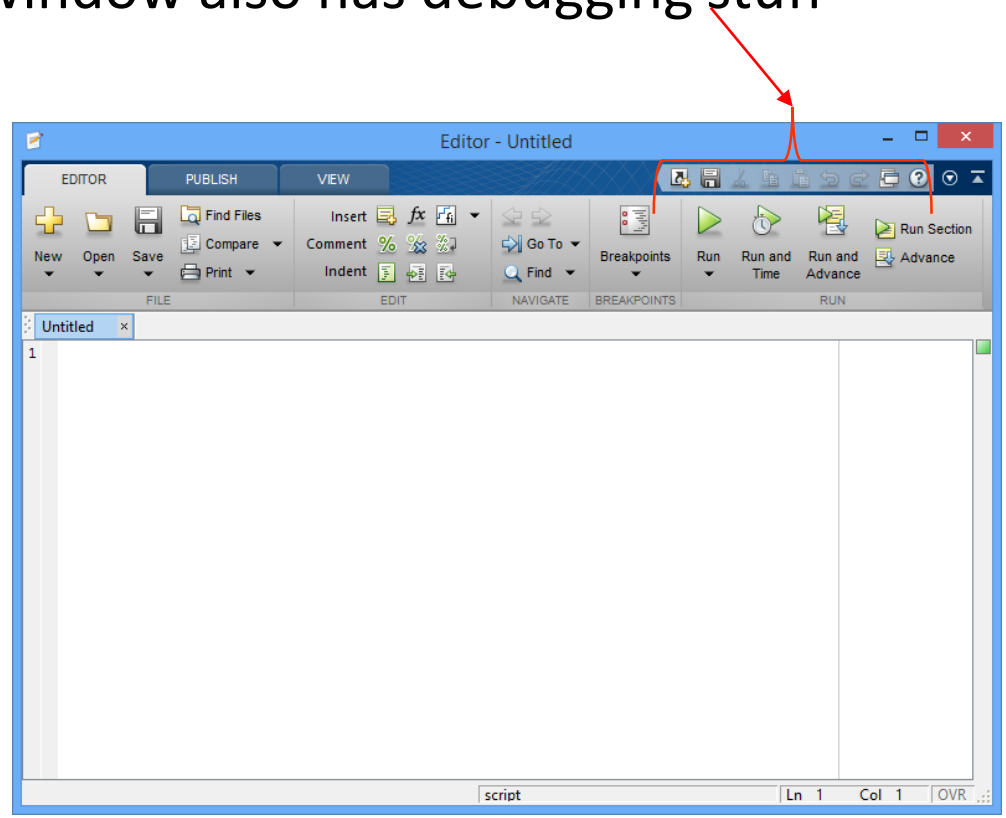
- Command Line
- Workspace (variable list)
- Command History
- Directory Browser




- Can type *commands* in the command line
  - *Scripts* to run
  - Basic math operations
  - Run functions on existing variables (or make new ones)

# MATLAB Scripts

- Under Home→New/Open you can create/open scripts (small programs) and/or functions
- Editor window also has debugging stuff



# Variables

- Variables in MATLAB do not need to be declared as a type
  - It is inferred, and can change based on assignment
  - HOWEVER, often you can only do operations with certain data types, so you may need to do
    - *casting*
      - `double(X)`
      - `uint8(X)`
- Variable names have the following rules:
  - Must start with a letter
  - All other characters must be letter, number, or the underscore _ character
  - Variables **are** case-sensitive

# Variables

- HINT:  At the beginning of a script you may want to clear out old variables (this doesn't happen automatically in MATLAB)
  - To do this type `clear all` (maybe have this be the first line of your script?)

# Operators

- MATLAB uses the standard arithmetic operators: + - * /
  - **There is no increment/decrement operator ☹**
- It also uses ^ for *power*
- MATLAB uses % for comment lines, so to do modulus, we use a modulus *function*

```
x = mod(5,3);
```

- Since it specializes in matrix operations, you can do standard matrix math (both matrix/vector and scalar)
- But can also do *element-wise* operations
  - Apply operator to each element of the two matrices/vectors
  - These start with a "."

```
.*   ./  .^
```

# Arrays

- In MATLAB we refer to these as **vectors** (1D) and **matrices** (2D)
  - Can also be higher dimensional (we'll see a lot of 3D matrices)
- Create a matrix/vector by putting elements in square brackets separated by commas or spaces

```
myVector = [4, 3, 8, 0];
```

- To go to a new row, we use a semicolon

```
myMatrix = [4, 3, 5; 0, 2, 3];  % a 2x3 matrix
```

# Arrays

- Can use built-in MATLAB functions to make vectors/matrices

```
myVector = zeros(1,5);  % 1x5 matrix of zeros

myMatrix = rand(2,4);
% a 2x3 matrix of random numbers, each in the range of [0,1)
```

- We can create a vector using the colon : operator

```
x = 1:1:10    %from 1 to 10 in increments of 1
x = 1:10      %shortcut
x = 1:3:50;   % from 1 to 50 in increments of 3
```

# Arrays

```
myMatrix = [4 3 5; 0 2 3];
```

```
myVector = [4 3 8 0];
```

- We can access locations in vectors/matrices by putting the location (index) in **parenthesis** after the variable name
  - **NOTE: In MATLAB locations start at 1 ☹**

```
disp(num2str(myVector(3)));
disp(num2str(myVector(0)));                %error!
disp(num2str(myMatrix(2,1)));
```

- We can use the function *size* to get the number of elements for a given dimension

```
size(myMatrix,1); %number of rows
size(myMatrix,2); %number of columns
```

# More Matrix Indexing

```
myVector = [4 3 8 0];
```

```
myMatrix = [4 3 5; 0 2 3];
```

- We can specify several locations at once by specifying a vector for the index

```
myMatrix([1 2], [1 3]);
myVector(1:2);
```

- We can also use a special *end* keyword to say “the last”

```
myMatrix(1, 2:end)
```

- Or use the colon operator by itself to mean “all”

```
myMatrix(2,:);
```

# Control Statements

- MATLAB has the same standard control statements as most modern programming language
  - if … else
  - switch
  - while
  - for

- NOTE:  Control statements end with the **end** keyword
  - Doesn't use brackets or indentation like other languages

# If Example

```
num = -3;

if ( num < 0 )
    disp([‘The number ‘ num2str(num) ‘ is negative’]);
elseif (num==0)
    disp([‘The number ‘ num2str(num) ‘is zero’]);
else
    disp([‘The number ‘ num2str(num) ‘ is positive or zero’]);
end
```

# For Loops

- These are actually more like *for each* loops in other languages
  - Does the body of the loop *for each* value in a vector

```
for i=1:10
    disp([‘Current number: ‘ num2str(i)]);
end
```

# Available Functions

- There's tons of built-in functions for us to use.
- The built-in documentation should be your best friend!
- As we go through the course I'll show code illustrating "new" relevant built-in functions

# Functions

- We can make our own functions of course
- Typically each function should be in its own file with the same file name as the function name
  - Although you can have multiple functions in a file
- The format of a function is

```
function [return_params] = name(params)
        %comments
end
```

# Parsing Text Files

- The first thing we'll need to do in this class is get data!
- We'll get data from text files
- But unfortunately they may be in all different formats ☹
  - If we're lucky it may be in .csv format and we can use the `csvread` function
  - But even then there may be mixed data types making it impossible.
- So let's look at how to parse a general text file
  - Plus this will allow us to look at some Matlab!

# Parsing Text Files

- Parsing files takes a while
  - And we might need to do this over and over again while we test/build our system.
- So often we want to just parse it once, save its data in an easy to use format, then next time load it.
- Let's get started

```
clear all; %remove all the old variables in the workspace
close all; %closes all open figures

filename = 'toy.data';  %the file to read
datafile = 'toy.mat';
if(exist(datafile,'file'))
    load(datafile);
else
```

# Parsing Text File

- Else we need to parse the file.
- First we need to try to open it.

```
fid = fopen(filename);
if(fid<0)
    display('File not found');
    return;
end
```

# Parsing Text File

- We opened the file and saw
  - The first line is just header info, so we want to skip it
  - Then the data is in the format
    - “[integer]” [double] [double] “[integer]”
- So we can use regular expressions to help us get this data!

```
fgetl(fid);%remove the first line
x=[];
y=[];
while(~feof(fid))
    line = fgetl(fid);
    C = textscan(line,'"%d" %f %f "%d"');
    x(end+1, :) = [C{2}, C{3}];
    y(end+1,1) = C{4};
end
```

# Parsing Text File

- Finally let's close the file and save the data so next time we don't need to re-parse

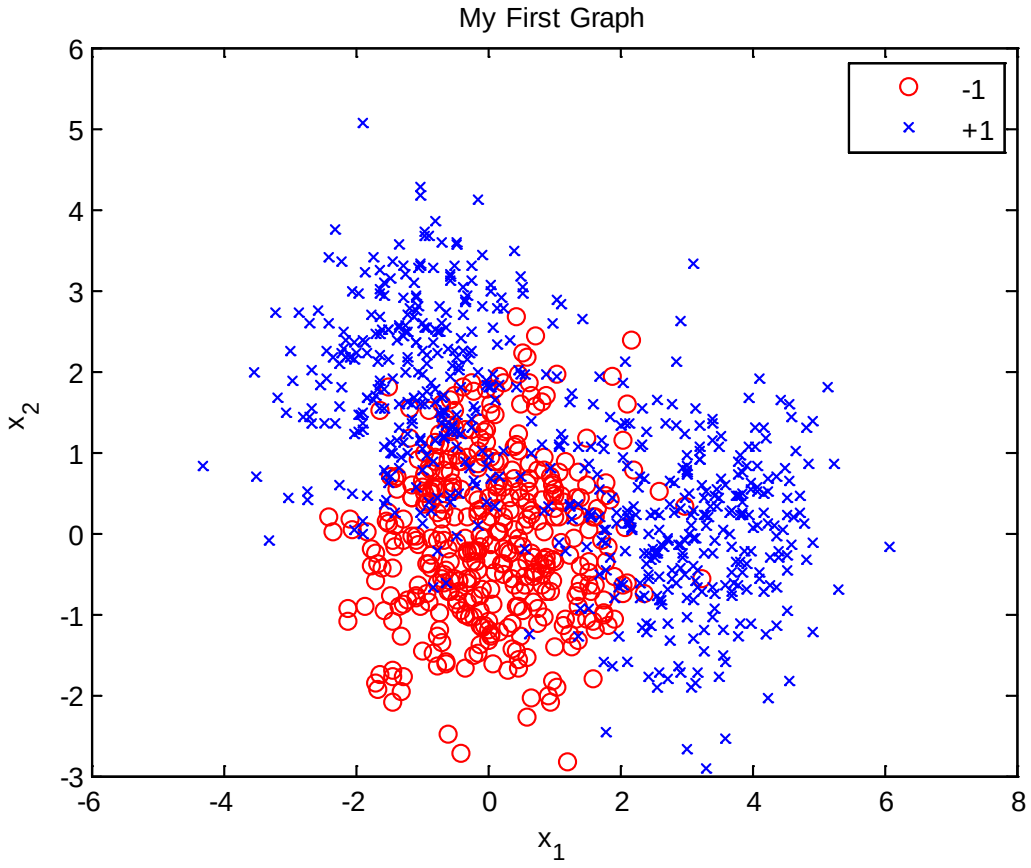
```
fclose(fid);  %don't forget to close your file!
    save(datafile,'x','y');
end %end of if(exist.... else...
```

# Parsing Text File

- Finally it would cool to see the plot of this data

```
figure(1);
plot(x(y==-1,1), x(y==-1,2),'or');
hold on;
plot(x(y==1,1), x(y==1,2),'xb');
hold off;
title('My First Graph');
xlabel('x_1');
ylabel('x_2');
legend('-1', '+1');
```

# Parsing Text File



# Available Matlab Functions

- I've created an area on Blackboard called *Matlab Functions* where I'm keeping a running list of built-in Matlab functions that I think are important for the course.
- Below is also a link to a Matlab tutorial:

  http://www.tutorialspoint.com/matlab/index.htm