EE133A Mathematics of Design—Week 1

Cameron Allan Gunn
Electrical Engineering Department
UCLA

1 Introduction

This week we will work through the foundations of using the programming language MATLAB. If you have used programming languages in the past, some of this material will be familiar to you.

MATLAB is an important language for research and development. Traditional languages such as C, Python, and Java are used to create products for consumers to use. However, they are difficult to use when working with complex mathematics, particularly in Science and Engineering. MATLAB instead focuses on making math easy to compute, and is used extensively in research.

Write out each task in MATLAB as you read through it, and make sure you understand what the computer is doing with each line of code. As your work through these tasks, create your own *MATLAB Quick Reference Guide* in the back pages of your notebook. List useful commands and notation as you learn them, and write down quick examples so that you can look back for inspiration when you need to.

Note: MATLAB will be necessary for components of this course. Because you are enrolled, you should have access to UCLA's licences via SEAS computers or VPN. MATLAB also has student licences available. Additionally, **Octave** is a free alternative to MATLAB, that is mostly identical for the purposes of this course.

2 Getting started

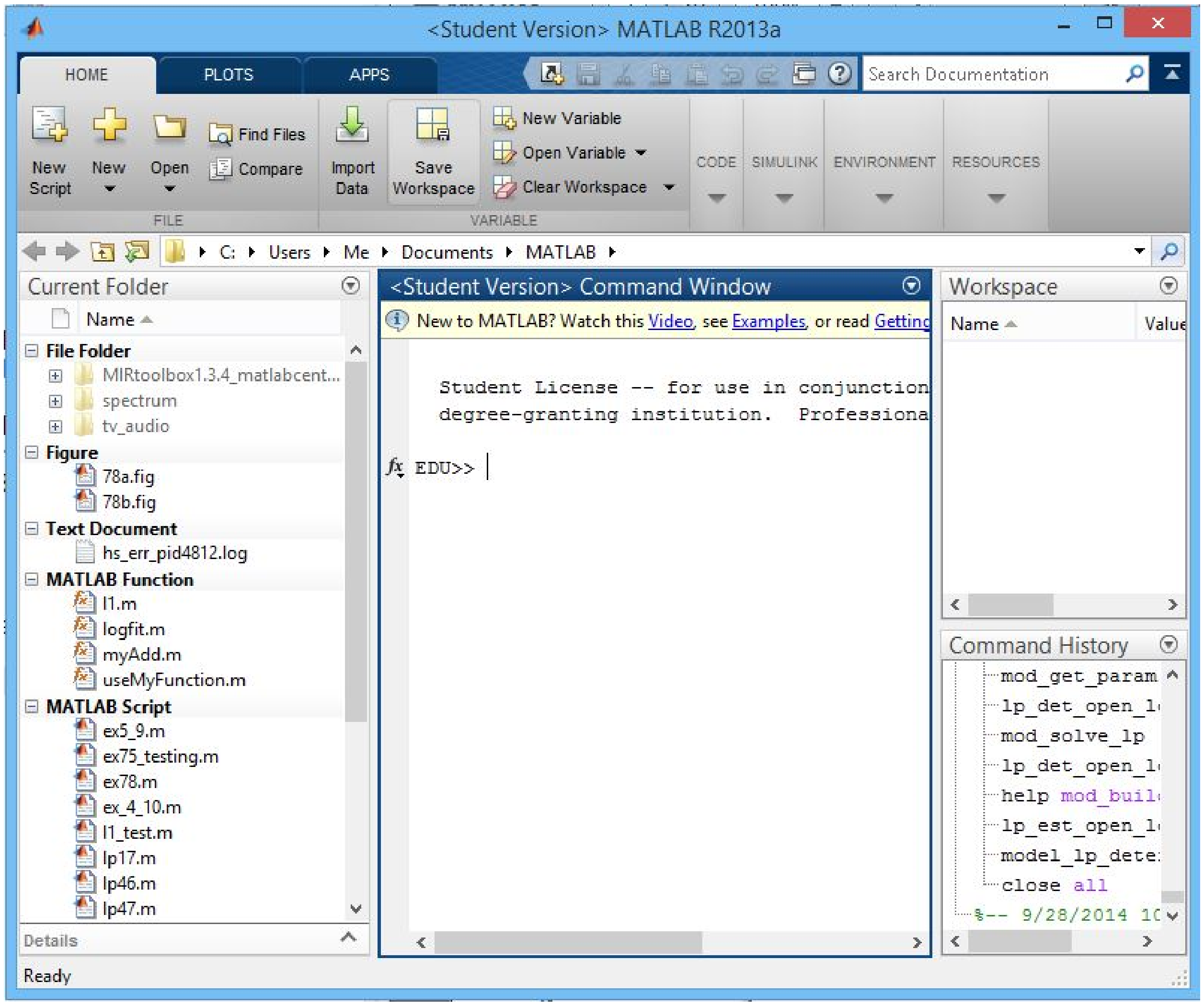
Open up MATLAB on your machine. You should see a window like that shown in Fig. 1. The command window is the main window we will focus on today.

3 MATLAB is a calculator!

MATLAB can be used as a calculator by typing equations into the command window. Some operators include:

- `+`, `-`, `*`, `/`, \ for simple arithmetic
- Brackets, ( ), help order operations
- `i` or `j` are the imaginary unit $\sqrt{-1}$
- `^` raises number to an exponent
- `sqrt( )` takes the square root of the number in the brackets
- `sin( )`, `tan( )`, `log10( )`, ...

Try these functions out. Think about:



File Folder
- MIRtoolbox1.3.4_matlabcent...
- spectrum
- tv_audio

Figure
- 78a.fig
- 78b.fig

Text Document
- hs_err_pid4812.log

MATLAB Function
- l1.m
- logfit.m
- myAdd.m
- useMyFunction.m

MATLAB Script
- ex5_9.m
- ex75_testing.m
- ex78.m
- ex_4_10.m
- l1_test.m
- lp17.m
- lp46.m
- lp47.m

New to MATLAB? Watch this Video, see Examples, or read Getting

```
  Student License -- for use in conjunction
  degree-granting institution.  Professiona

fx EDU>> |
```

```
mod_get_param
lp_det_open_l
mod_solve_lp
lp_det_open_l
help mod_buil
lp_est_open_l
model_lp_dete
close all
%-- 9/28/2014 10
```

Figure 1: MATLAB command window.

- How would you take a cube root of a number?
- How does MATLAB order operations?
- Does MATLAB use degrees or radians? How would you switch between the two?

4 Assigning variables

MATLAB lets us store numbers in boxes called **variables**. The `=` operator stores information into a variable. Try this:

```
a = 5
b = 10
a + b
c = b - a
```

MATLAB takes the expression on the *right hand side* first, evaluates it, and then stores the result in the box labelled on the *left hand side*.

- The `=` should not be read as "equals" in MATLAB, but as **"becomes equals to"**. The command `a = 5` is not an equality, but a command to store `5` in the box labelled `a`.
- Variable names are case sensitive, so `A` and `a` are different variables.
- Variable names cannot have spaces, any special characters, or start with a number.

Take note of the **workspace** panel in MATLAB. Here you can see what your values are stored in your variables.

Try this variation of our code:

```
a = 5;
b = 10;
c = b - a
```

- What does the `;` operator do? What is the advantage of using (and not using) it?

All of our variables can be deleted by typing `clear` in the command window. Similarly, we can refresh the screen by typing `clc`. It's a good habit to clear variables before you start working, so that those from previous calculations aren't lingering around.

5 Making scripts

We want to be able to write code in a document and call on it when necessary. For this, we use scripts. In MATLAB, scripts have the suffix `*.m` and are sometimes called "m-files".

Click on "New Script" in the MATLAB window to open the editor to a new script.

- **Pro-tip:** Snap the MATLAB window to the left of your screen and the editor to the right, so that you can use them interchangeably.

Now we'll write some code:

```
%% My first script
% Author:  Cameron Gunn
% Date:  28 Aug 2014
clear;
clc;

a = 5;
b = 10;
c = b - a % My result
```

Save your script with a sensible filename. Now, run it by pressing the *play* button in the editor, or pressing **F5**.

Notice that anything written on the line after the `%` is ignored by MATLAB. These are called **comments**, which are important for keeping our code tidy. Commented code is significantly easier to come back to after a week away from MATLAB! (Tidy code also makes for happier graders...)

Pro-tip: All good script files start with the `clear;` command, so that we know that variables from previous projects have been removed!

6 Loops

Computers' key advantage in math is being able to repeat instructions very quickly. Let's see how to make our scripts perform *loops*:

```
%% My first loop
% Author:  Cameron Gunn
% Date:  28 Aug 2014
clear;
clc;

for ii = 1:10
    ii
end
```

Here, MATLAB creates a variable that we have called `ii`. At first `ii` is set to 1, and everything inside the loop is run. Then `ii = 2`, and run again. This continues until the last execution, when `ii = 10`.

We call `ii` the *loop counter.* Note:

- In some languages, `i` and `j` are commonly used as loop counter variables. But because these are set aside as the number $i = j = \sqrt{-1}$ in MATLAB, **never** use these as variable names—use `ii`, `jj`, `kk` instead.

We can also put loops inside of loops:

```
for ii = 1:10
    for jj = 1:5
        ii
        jj
    end
end
```

7 Matrices and vectors

One of MATLAB's strengths is its ability to seamlessly work with matrices and vectors. Try some of these commands in the command window to see how they work:

```
a = [1, 2]
b = [4; 5]
Q = [3, 6; 9, 13]
G = [1, 2, 3; 4, 5, 6]

2 * G
G'
eye(2)
ones(3, 2)
zeros(4, 5)
Q * Q
inv(Q)
det(Q)
```

By default, MATLAB will use **matrix arithmetic**. For example, the dot product $b^T c = \sum_{i=1}^{n}(b_i c_i)$ could be found by typing:

```
b = [4, 5]
c = [4; 1]

b' * c
```

We can multiply, add etc. each corresponding element in a vector or matrix by using **element-wise** operators:

```
b = [4, 5]
c = [4; 1]

b .* c
```

This is sometimes called "dot-multiplication".

We can find out information about our vectors and matrices, as well as access individual rows, columns or elements. First we'll define some vectors and matrices using some different methods:

```
A = [1, 2; 3, 4]
b = [4, 5, 6, 7]
c = [4; 1]
G = eye(7)
F = ones(4)
q = 1:10
p = 3:0.5:7
```

We can find out information about the matrices overall:

```
length(b)
size(A)
```

We can access individual elements by asking for a scalar entry:

```
b(1)
c(end)
A(1,2)
```

We can also access multiple entries by providing a vector containing a list of entries desired:

```
b(2:3)
b([2,4])
A(2,:)
G(:,6)
G(3:end,2:3)
A(1,1) = 8
```

Notice that the first entry in vector `a` is `a(1)`, not `a(0)` as it may be in other programming languages.

Think about:

- What is the order of rows and columns when accessing an element in a matrix?
- What are the two functions of the `:` operator?

8 Plotting activity

Let's say that we want to plot out the function $f(x) = x^2 - 2$ over the range $(0, 5)$. We can't plot a function directly, but we *can* evaluate the function at a series of points that are close together to approximate it.

First, we want to generate our series of points along the x-axis.

```
lowerBound = 0;
upperBound = 5;
N = 50;
x = linspace(lowerBound, upperBound, N);
```

The `linspace` command generates N evenly-spaced points between a lower and upper bound. Write `x` to see the contents of our vector.

For each of our 50 points, we want to evaluate our function $f(x) = x^2 - 2$. We know we want $N = 50$ points, so we can prepare a corresponding $f(x)$ vector:

```
fx = zeros(N,1);
```

Pre-defining a large vector lets MATLAB know in advance how large it needs to be. Thus, MATLAB can store it in an optimal location, speeding up our program.

To calculate each value of $f(x)$ at its corresponding x:

```
fx(1) = x(1)^2 - 2;
fx(2) = x(2)^2 - 2;
⋮
fx(50) = x(50)^2 - 2;
```

We can put this behaviour into a loop!

```
for ii = 1:N
    fx(ii) = x(ii)^2 - 2;
end
```

Now we can plot our results:

```
plot(x, fx);
title('This is my function');
xlabel('x');
ylabel('f(x)')
```

Try and write this task in a single script file. Remember to start your script with `clear;` and `clc;`!

What happens if we set `N = 3`? What about `N = 2358923497`?

Type `help plot` to see some additional options related to the `plot` function. The `help` keyword will provide information for other functions too!

9 Functions

We often work with mathematical functions of the form $f(x)$, which contains a command translating the *argument* x into the *function result* $f(x)$. We can replicate this in MATLAB.

```
f = @(x) x^2 - 2;
```

If we now type `f` into the command window, we don't get a number, but a description of the function. But by typing `f(2)`, the function is evaluated at $x = 2$. **Re-write your plotting task using a function**.

We can have as many arguments as we want:

```
f = @(x,y,z) x^y - z;
```

The drawback of this notation is that we can only write a simple, one-line function. However, MATLAB lets us create arbitrarily complicated functions, so long as they are in a file of their own.

We're going to make a function that finds the weighted sum of a vector. Open a new script:

```
function [y] = myFn(x)
%% Sum function
% Author:  Cameron Gunn
% Date:  28 Aug 2014

end
```

Note that:

- Functions, unlike scripts, begin with the keyword `function`.
- The notation mimics $y = f(x)$ from mathematics: here `myFn` is the function name, `x` is the list of arguments, and `[y]` is the list of outputs.
- Function files must have the **same name** as the name of the function, or MATLAB won't be able to find it.
- Typing `help myFn` in the command window will display the first block of comments—so we should put useful information here for future users.

When we enter a function, we enter another a new workspace. This means that

- We can only access the variables we gave the function in the arguments.
- We don't want to write `clear`, because MATLAB has already cleared the variables that we don't want.

Our weighted summer would look something like this:

```
function [sum] = weightedSummer(x,w)
%% Sum function
% Author:  Cameron Gunn
% Date:  28 Aug 2014

end
```

Write a script that uses a *loop* to calculate the equation

$$\text{sum} = \sum_{i=1}^{N} (w_i x_i)$$

Remember that you have a command to find the length of the vector `x`, to calculate how many times the loop must be run. Save your function as `weightedSummer.m`.

Now in the command window, type

```
x = [1;2;3;4];
weights = [.2; .2; .4; .2];
myWeightedSum = weightedSummer(x, weights)
```

10 Conditional branching

We can use MATLAB to evaluate mathematical statements to true (1) or false (0).

- `>, <` for strict inequalities
- `>=, <=` for nonstrict inequalities
- `==` for equality
- `~=` for "not equal to"

Choose an `x` and try these equalities, such as

```
x == 5
x^2 >= 2*x
```

We can string these statements together using *Boolean operators*:

- `&&` for AND
- `||` for OR
- `~` for NOT

So "x is greater than 2, and y is not equal to 3, or instead of all of that, z is equal to 10" can be written

```
(x > 2 && y  = 3) || (z == 10);
```

We can have MATLAB execute tasks based on the outcomes of these comparisons. For example:

```
if height <= 60
    disp('You are too short to ride the roller coaster.');
elseif height >= 80
    disp('You are too tall to ride the roller coaster.');
else
    disp('You may ride the roller coaster!  Whoop whoooooop.');
end
```

We can also put braches within branches.

```
if x < 50
    if x > 25
        disp('Your number is between 25 and 50');
    else
        disp('Your number is less than 25.');
    end
else
    disp('Your number is greater than 50.');
end
```

Can you write this task using Boolean algebra instead of nested `if` statements?

11 Extra task

Write a script file that takes three scalars a, b, and c, and calculates the roots of the function

$$ax^2 + bx + c = 0$$

Do not use MATLAB's support for complex numbers. Look up the quadratic formula online if you need to. Use `if` statements to separate out the special cases.

Some cases to think about:

- Check $b^2 - 4ac$. Is it equal to, less than, or greater than zero?
- Calculate the imaginary parts separately if necessary.
- Think about special cases, e.g. $a = 0$, $b = 0\ldots$

Alter your script so that it is a function of the form

```
function [x] = solveQuadratic(p)
```

where `p` is a vector $p = [a \quad b \quad c]^T$.

12 References

[1] Alex James (2014). *EMTH171 Mathematical Modelling and Computation 1—Lab Manual*, University of Canterbury.