M599 Lab First Week(s) - Further reading and advanced topics.

José L. Pabón

September 7, 2024

1 Math 599.

1.1 Introduction

Welcome to Math 599, Lab sections for further reading and advanced topics.

1.2 Objectives

The educational goals of this laboratory will be to acquire a mathematics graduate student understanding of the following topics.

- Basics of operating systems Windows and MacOs Sonoma.
- Basics of Github.
- Basics of scripting and coding.

This list is subject to change.

2 Instructions for our activity.

3 Advanced topics.

3.1 Install

This section will review some basics of scripting and coding. We recommend you use MATLAB, but any programming language you want to use you can. Lab participants who need specific instructions to complete these tasks should probably just use MATLAB. We recommend you download MATLAB on your own machine. Create an account on Mathworks using your NJIT email account. Your account should have an active MATLAB license. Download the latest version of MATLAB directly from their website, not the NJIT website. All you need to do is activate the software using the account you just made.

3.2 Interface

We write code in MATLAB but it's much easier than writing code in a compiled language: you don't have to load any packages, you don't have to worry about pointers or data types, syntax is intuitive, and debugging is a straightforward. The MATLAB interface consists 3 windows: a Command Window, a file browser, and a Workspace; and it has a toolbar at the top. The Command Window works similar to a terminal in Linux that you used last week. In MATLAB, you just use equal sign. Once you create a variable, the variable name and its value will be listed in Workspace. The path of your working directory is displayed right below the toolstrip. The contents of your working directory is shown in the Current Directory panel.

3.3 The Terminal

First, you should learn the Command Window basics. Try out these commands: `cd, pwd, mkdir, rmdir, delete, ls, dir, edit, type, clc, clear, help, lookfor`. Many of these should look familiar to you. The MATLAB command `help` is the equivalent of `man` (manual) command in Bash. Try typing in these commands:

```
pwd
name = 'MATLAB_folder'
mkdir(name)
ls
cd(name)
pwd
edit 'readme.txt'
```

The last command creates a textfile for you and opens up a editor window. You can take some notes if you wish. It's good practice to document every code you write.

3.4 Tutorial

Just follow the tutorial here.

3.5 Functions

The files with .m extension can either be scripts or functions. You can make your own function and then call it just like a built-in MATLAB command. The first line of every function has to follow this basic format:

```
function [ output_args ] = function_name( input_args )
```

Type in the command `edit hello.m`. This will create a file called 'hello.m' and open up a text editor for it. Type in these lines of code

```
function [name_length] = hello(name)
% This is a function that says hello to you and returns the number of
% characters in your name.
message = ['Hello, ' name];
name_length = length(name);
```

```
disp(message)
end
```

Try out the command using your own name. Call the function from the terminal. Also try typing `help hello`. One thing to note about functions is that they have their own private workspace. Type in `name = 'abc'` in the terminal to create a new variable with same exact variable name as the one used in the function. When you type in the command `hello('def')`, what will the output be? When you run a command, you never have to worry about clearing your workspace because functions only used local variables. For now, you can only call the function from the directory where the function is saved. If you want to be able to call the function from directory, then you need to change the settings. Go to the Home tab and click on Set Path. Add the folder containing your function to MATLAB's search path and then you will be able to use the function `hello` from any directory.

You can also make anonymous functions for when you need simple arithmetic functions. For example, suppose you wanted a to code the real-valued function $f(x) = \frac{1}{1+x^2}$. Then you can type `f=@(x) 1./(1+x.^2)` to create the function handle `f`. Copy and paste these lines of code into a script and run it:

```
x = linspace(-1,1);
f=@(x,p) 1./(1+x.^2).^p;
hold on
for i = 1:3
    plot(x,f(x,i))
end
legend('1','2','3')
```

MATLAB can also do symbolic computing. Symbolic functions are useful for when you need exact representations, derivatives, integrals, or solve simple ODEs. See the link for the syntax and basic usage.

3.6 Assignment

Instructions: Place all required responses (codes and figures) into a single pdf document(look for matlab publish feature). Submit this document electronically.
Problem: Consider the iteration of the map $f(x) = c \sin x$ where c is a constant. In particular we seek the largest interval $I = [-\pi, \pi]$ containing the origin for which $c \in I$ implies that the sequence

$$x_{n+1} = f(x_n) \tag{1}$$

converges for a reasonably wide range of starting values. In other words, this requires us to check all the possible values of c. Of course, they cannot be checked one by one but must be grouped into sets with properties that allow us to check all the values in the set together.
The analysis begins with a bifurcation diagram:

- Write a Matlab program *csin.m* to produce figure 1, which plots the points $\{(c, x_n) : n = 151, ..., 160\}$(this means you are going to skip the first 150 iterates) for a range of $c \in I$ with an initial point of $x_0 = 2$. This plot suggests that the iterates converge for $c \in [-1, c_{max}]$ where c_{max} is slightly larger than 2. Just outside this interval the iteration seems to settle

into a solution with period two, i.e. it oscillates between two values. Can you say something about the convergence? (hint: consider $c \in [-1, 1]$ and $c \in (1, c_{max})$ seperately)

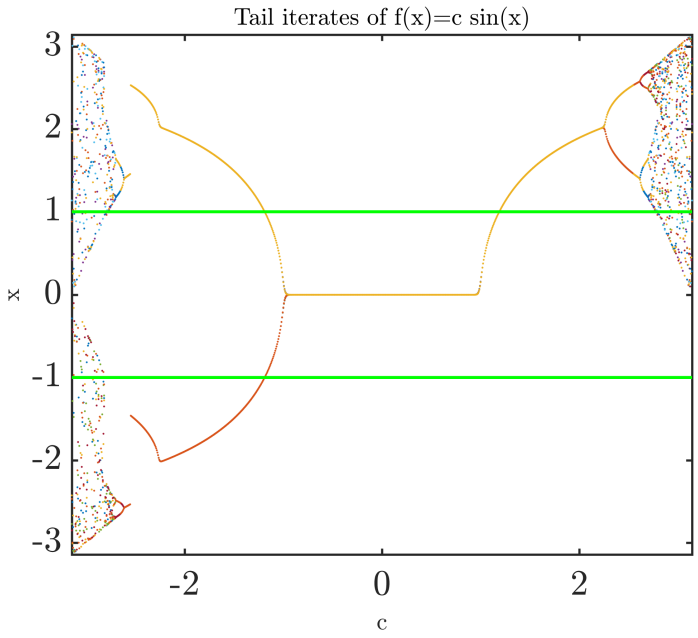
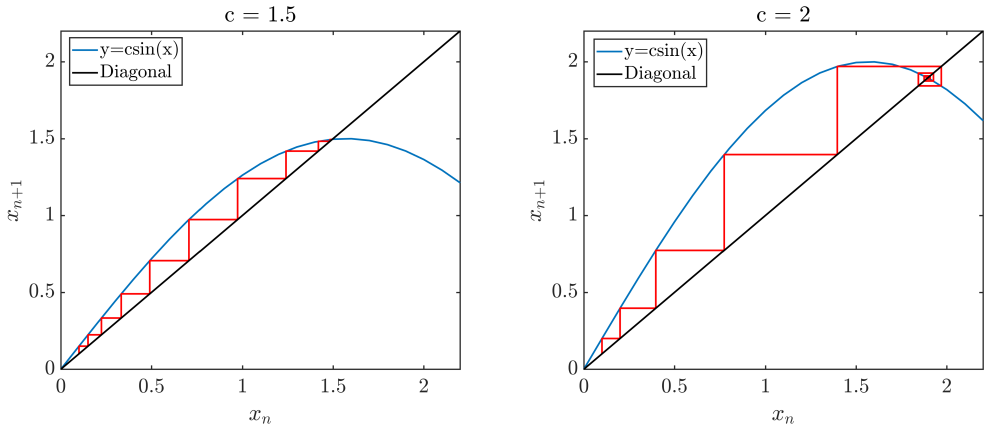


Figure 1: Bifurcation diagram

- To get some idea of what is going on for $c \in (1, c_{max})$, consider cobwebbing for $c = 1.5, 2$. Write another Matlab code to produce figure 2 and explain why the iteration of $c = 1.5$ is always forward, but the iteration of $c = 2$ is oscillating near the fixed point.



(a) c less than $\frac{\pi}{2}$ (b) c greater than $\frac{\pi}{2}$

Figure 2: Cob web diagram

- Find c_{max} and the associated fixed point to at least six decimal digits of accuracy. **Hint:** At the particular value c_{max}, $f'(x_s) = -1$. This is the point at which the fixed point is about to loose its stability. This value is characterized by the equations:

$$\begin{aligned} c_{max} sin(x_s) &= x_s \\ c_{max} cos(x_s) &= -1 \end{aligned} \tag{2}$$

It following that the associated fixed point x_s satisfies

$$\frac{\sin x_s}{\cos x_s} = -x_s \tag{3}$$

so that it can be found as a root of

$$g(x) = sin(x) + xcos(x) \tag{4}$$

Then you can use Newton's method to find the approximate value of x_s and the corresponding value of c_{max}.

4 Advanced topics, further.

We will be doing more MATLAB today. Everybody should have a go-to scripting language to do quick-and-dirty computations. It can be Python, R, Julia, etc. I prefer MATLAB because it has many practical, well-documented functions and its interface allows for easy debugging. If you have not done the last assignment, continue working on that, and if you have done so, you can check out the following MATLAB feature by yourself:

- Basic code structuring (comments, variables, blocks, functions, loops)
- Cells, Tables and Structures (which are objects in MATLAB)
- The Publish feature
- Live Script
- Debugging
- Improve efficiency (parfor and vectorize)

The best way to learn is to try to solve a problem yourself.

4.1 Problem Solving Exercise

Sudoku is a puzzle game where you fill in the remaining spots of a partially filled 9×9 grid according to a specific rule. Here is a description:

> The classic Sudoku game involves a grid of 81 squares. The grid is divided into nine blocks, each containing nine squares.
>
> The rules of the game are simple: each of the nine blocks has to contain all the numbers 1-9 within its squares. Each number can only appear once in a row, column or box.
>
> The difficulty lies in that each vertical nine-square column, or horizontal nine-square line across, within the larger square, must also contain the numbers 1-9, without repetition or omission.
>
> Every puzzle has just one correct solution.

Your task is to write a code that verifies the solution of a solved board. Go here and download a textfile of a solved board. Write a MATLAB script that imports that file and checks the solution of that particular board.
Due: Next week, in class.
Hint: First, you are going to find the most proper way to import the data, I used *readtable* to import the *.txt* file as a MATLAB table (slightly modify the *.txt* file before using *readtable*). Then convert the table into a matrix. Second, to check if the solution is correct, you need to consider the following constraints:

- check the sum on each row is 45.
- check the sum on each column is 45.
- check for sum on each box is 45.

- check for duplicate numbers on each row.
- check for duplicate numbers on each column.
- check for duplicate numbers on each box.

4.2 A couple of good LaTeX references.

One good, open source, i.e., free reference about LaTeX is this book [1]. LaTeXis an extremely mature language; there's no shortage of reference material out there for it including this univeral resource link (URL).

References

[1] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. The not so short introduction to latex2ε. 1995.