MATH2070: LAB 1: Python Setup

1 Introduction

You will find instructions for each lab, including this one, on canvas. This lab will occupy today's lab session only. This session will introduce the mechanics of using Python. The second lab, two sessions long, will present exercises in Python use.
The discussion that follows assumes that you are basically familiar with browsing the Web. The next few sections will give a brief introduction to Python and explain how to use it and those aspects of the environment that will be important to doing the labs.

2 Grading

The labs roughly follow the material presented in lecture, but are independent of the homework and other exercises presented in lecture. Lab grades count as 30% of your course grade.
You are encouraged to work together with other students, but you must provide your own summary files (explained further below).
Each lab will be given a grade of A+, A, B, C, D or 0. These grades correspond with percentage grades of 100, 95, 85, 75, 65 and 0. At the end of the semester, your grades will be averaged and then integrated with your grade in lecture. The grading criteria are:

Grade	Value	description
A+	100	All exercises were completed correctly.
A	95	All exercises were attempted and are substantially correct.
B	85	All exercises were attempted but there are some serious errors.
C	75	Substantial portions of some exercises were omitted.
D	65	Little or nothing correct in the submission.
Zero	0	Lab was not submitted.

Each lab is due before 11:59 PM the day the subsequent lab begins. **Labs which are late will not be accepted, however your lowest lab grade will be dropped.**

3 Lab summaries

You should complete a report of the results you obtained for each completed lab. This report need not be elaborate. The summary file can be easily created as you do the lab by keeping a text file up in the editor and copying parts of the web page, your commands and output to the file as you work.
This summary file is very important. It is what I will read first and, if it is well-written and the work is done correctly, I will not need to read anything else. *Never* put incorrect Python statements into your summary because it will take me a lot of time to discover you really didn't mean them. I will regard everything in the summary as information you believe is correct and will grade it accordingly. I expect to see

- Brief descriptions of the work you did for each exercise.
- Copies of the main Python commands you used for the exercises, along with the numerical results.
- Your description *must* include more detail than merely "yes, I did it and it worked."
- Names of plot files (`.jpg` or `.png`) corresponding with the different exercises.
- Explanations of anything unusual or interesting, or points of confusion that you were unable to resolve outside lab.

- If you believe I have an error in a lab, please inform me of it. Explain why you think it is an error and, if you like, suggest a correction.

Summarizing your work is important not only for my convenience in grading, but also to help fix in your mind the focus of each exercise.
Equally important, the summary file helps get you into the habit of keeping track of your numerical experiments in some formal manner. When you are doing research, you may be doing hundreds of numerical experiments, and you *must* get into the habit of documenting your work or you will not remember from month to month what each one did. The idea of the summary is that you can easily refresh your memory on exactly what you did to accomplish some particular task.
Here is what I want to see in the summary file:

1. Those parts of the answers to each exercise that I ask for.
2. Explanations of what you did, in full sentences. There should be enough information here to repeat the experiment. Python files will help in this documentation.
3. I would like to see a few lines expressing your opinion of the point of each exercise.
4. Easily identified answers to exercises, including numerical values. I do not want to look in your diary file to try to figure out what you did. *Do not* write, "see summary file for details."
5. What was the result of the experiment? Not just the numbers, but what the experiment told you.

Here is what I do NOT want to see in the summary file:

1. Python error messages (unless I ask for them).
2. False starts, mistyped commands, etc.
3. Incorrect results that are corrected later.
4. Duplications of anything, unless I explicitly ask for them.
5. Large numbers of printed values, for example, the contents of a vector of length 100. I will not read all these numbers and they end up being like spam.

If you want to know how much detail to include, think of the following scenario. You have completed this course and, a year from now, a friend who is taking the course is having trouble. Your friend comes to you and asks how you did a particular exercise. You have saved your work, so you go look at it. The first place you will look is in your summary to see what you did. If the summary file contains only "Exercise 1.a: complete," you will then have to go re-read the original lab and look for your script files, *etc.* Instead, the summary should describe what you did so you can explain it in general to your friend without referring to other materials. If your friend needs more detail, you can look at the other files you wrote for the lab.

4 Unix/command prompt basics

For this course you will need a basic understanding of the directory structure your computer uses and some basic commands for navigating it from the terminal (if you are on a mac/linux) or command prompt (if you are on windows).
The commands you will need are as follows:

- cd /path/to/directory- This command changes the directory you are currently in to the diretory inputted.
- ls - Lists the files in your current directory

- pwd - Prints the path of your current directory (unix only)
- cd (with no arguments) - Prints the path of your current directory (windows only)
- mkdir - used to make a new directory
- rm - used to delete files (but not directories)
- rmdir - used to delete directories
- ./ - used to run a compiled program in your current directory

The following exercises are meant to give you some familiarity with these commands.

Exercise 1:

(a) Open up command prompt or terminal on your computer. This can be found in the start menu if you are on Windows or using search and typing terminal if you are on a Mac.

(b) In the terminal type of the command `pwd`. Save the output to a summary file. Note that this output is telling you what your systems home directory is.

(c) Type the command `ls` into your terminal. Note the output to yourself (no need to put this in your summary file). This is listing all the files and directories located inside your current directory.

(d) Make a directory called test using the command `mkdir test`. Afterwards type the command `ls`, has the output changed at all from the previous step. Note the change in output in your summary file.

(e) Move to your newly created directory using the command `cd test`. Next, type the command `ls`, note what the output is (it should be nothing, since there is nothing in this directory yet).

(f) Download the executable file hello_world.sh from canvas and save it to the directory test.

(g) Now delete the hello world file with the command `rm hello_world.sh`. Type the command `ls` to verify that is has been deleted.

(h) Now move up one level in your directory back to the home directory using the command `cd ..`. Here the `..` is saying to move up one level.

(i) Delete the test directory using the command `rmdir test`. Use the command `ls`. Do you still the directory test in the output? If not then it has been successfully deleted.

5 Python Introduction

Python is an open-source object oriented programming language. It is one of the most popular programming languages in the world and has become an industry standard in the fields of scientific computing, data science and machine learning. The purpose of this course is not to teach you the Python programming language, but to use it as a tool to explore the numerical methods you will be learning about during the lecture portion of this class. However, we will be introducing some of the modern day tooling that is used for professional Python development. You will be required to use all of the tools discussed in the following sections except for the Visual Studio Code (VScode) editor. You are welcome to use any text editor of your choosing.
Some links and references will be included at the end of the lab if you are interested in learning more. If you have never programmed before I highly suggest you go through one of the free basic tutorials which is available online.

6 Visual Studio Code

VScode is a lightweight open-source text editor made by Microsoft for Windows, Linux, and MacOS. While there are many text editors available VScode is currently the most popular developer environment tool in the world. It offers support for many programming languages beyond Python. I personally use it for all of my work and recommend you do too for this class.

Exercise 2:

(a) Download and install VScode for your operating system (https://code.visualstudio.com/)

7 Installing Python via Anaconda

There are a number of different ways to install Python on your computer. Depending on your operating system (non Windows) Python may already be installed on your computer. However, a default installation will come with just the Python standard library. The Python standard library is an extensive set of Python modules that handles things like string handling, text processing, and basic math functionality. However, the real power of Python comes the vast number of available 3rd party open-source packages.
The default installer for these packages is the Python package installer pip. A difficulty with using pip however is that frequently different packages will have conflicts with one another. Resolving these conflicts manually represents a significant amount of work, especially in fields such as data science. To tackle this problem a number of tools have been developed. One of these tools and the one we will be using for this class is the conda package manager built into Anaconda. We will see that it easily allows us to install different packages while it automatically deals with any dependency conflicts.

Exercise 3:

(a) Download and install Anaconda for your operating system (https://www.anaconda.com/products/individual). If you are on Windows you should see an Anaconda powershell prompt appear in your start menu. Click on that to open it, we will be making use of it throughout the course.

8 Code Beautifier and Linter

Code quality is an important aspect of any program we write. Of course, we want the programs we write to do what we think they should do and be easy to understand, as well as modify. Python is a somewhat unique language in that it by default enforces some level of code quality due to fact that white space matters in the code. This is unlike other programming languages such as C++ and Java, which interpret white space as more or less not existing. Python is also somewhat unique in that it has an accepted style guide which defines a consistent way to write your code (the PEP8 https://www.python.org/dev/peps/pep-0008/ and PEP257 https://www.python.org/dev/peps/pep-0257/ style guides).
Even though we have these style guides, the question is how do we enforce them? This is what code linters and beautifiers are used for. Code linters automatically check your code for code and style errors. It's possible in most text editors, including VScode, for the linter to automatically run in the background as you type. This can greatly increase your efficiency as it will spot errors for you as you go. Code beautifiers on the other hand automatically format your code so that they conform to certain standards. Code beautifiers work very well in conjunction with code linters.
There are various Python code linters and beautifiers available, each with their own benefits and drawbacks. For the purpose of this course we will use the linter flake8 and the code beautifier Python black. We will see in the next section that these can easily be installed into our Python environment using anaconda.

9 Putting it all together

Now that you have installed all the tools needed we will run through an example of what your workflow for these labs will look like. You will need to download some files from canvas.

Exercise 4:

(a) In the lab01 directory download the install_conda.sh and lab01.yaml file in a folder on your computer named lab01_YOURNAME.

(b) Open a terminal on your computer. If you are on windows this should be done using the Anaconda power shell. On Linux or a Mac this can be done in VScode by navigating to the menu bar the top where it says Terminal and selecting new terminal or by simply opening a seperate terminal on your system by doing a search for terminal. In the terminal it should say (base) followed by your username on the computer, this indicates that Conda is working properly. Using the `cd` command in the terminal navigate to the folder you created at the previous step.

(c) Using the `cd` command in the terminal navigate to the folder you created at step (a). If you're unsure of how to do this either ask me or quickly read through the tutorial here https://www.geeksforgeeks.org/cd-command-in-linux-with-examples/

(d) Create your conda environment by running the comand:
`conda env create --force -f ./lab01.yaml -p ./condaenv`
This command is building the conda file based off of the lab01.yaml file. If you look in that file you will see this installs the scientific packages numpy and scipy, plotting package matplotlib, and the previously mentioned code linter flake8 and code beautifier black.

(e) Activate your conda environment using the command `conda activate ./condaenv`

(f) Create a new folder in your directory named code. Download the file lab01.py in the code directory on Canvas.

(g) Looking at the file lab01.py describe in your own words what the file does.

(h) Create a copy of the lab01.py file named lab01_old.py and save it to somewhere that is not in the same directory as lab01.py. Run Python Black with the command black ./code in the directory that contains the .yaml file. Copy paste the output and describe how the file lab01.py was changed.

(i) Run flake8 with the command `flake8 ./code` and describe the output.

(j) Make the changes requested by flake8, and run flake8 again to verify that it does not give any errors.

(k) Lastly, in the terminal move to the code directory and run lab01.py with the command `python lab1.py` record the output in your summary file.

(l) In your `lab01` directory there should be another directory called `condaenv`. This was generated when you ran the .yaml file and contains the Python environment you used for this lab. This directories tend to be large so first delete the `condaenv` directory and then zip your files and submit them via Canvas.

10 References

Below some links and book titles are given if you would like to learn more. Nothing is required, however if you have no programming experience I recommend trying to work through a bit of the beginners tutorial.

- Python beginners tutorial - https://python.land/python-tutorial
- In-depth discussion of Python linters and code quality - https://realpython.com/python-code-quality/
- Introductory Python book - Dive Into Python 3 https://diveintopython3.net/
- Intermediate Python book - Effective Python: 90 Specific Ways to Write Better Python