PP275 Spatial Data and Analysis, Fall 2019

Lab 0: Matrices, Coordinate Systems and Introduction to Python

Solomon Hsiang

University of California, Berkeley

About this lab

Due Date: Sep 4, 2019, 11:59AM

- When referring to computer commands and files, `I will use this font`.
- You may talk with other students about the lab, but each student is responsible for doing all exercises in the lab themselves and turning in their own write up. We will be checking code for evidence of copying and pasting.
- Please submit this lab to bCourses as `YOURCAL1ID.pdf`. This will facilitate anonymous grading. PDFs can be generated by scans or pictures taken of handwritten pages. Many iOS and Android apps can generate PDFs from pictures taken by the phone. One option is Genius Scan (for iOS or Android).
- The class just recently transitioned from Matlab to Python. This is a newly developed assignment, so if you think there is an error or something is unclear, let us know right away. That will be extremely helpful to us and your fellow students.
- These labs have been developed over the years by Solomon Hsiang, and past/current GSIs including Ian Bolliger, Tamma Carleton, Shubham Goel, Felipe González, Luna Yue Huang, Jonathan Kadish, and Jonathan Proctor (in alphabetical order).

1 Review of vectors and matrices

Following our notation in class, let

$$a = 1$$

$$b = 2$$

$$c = 50$$

$$d = 63$$

1. Let vector $\overrightarrow{r} = [a, b]$. Write out $\overrightarrow{r}$ in numbers.
2. Write out $\overrightarrow{r}'$.
3. Write out $\overrightarrow{r} + \overrightarrow{r}$.
4. Let vector $\overrightarrow{v} = [c, d]$. Write out $\overrightarrow{v}$.
5. Write out $\overrightarrow{v} + \overrightarrow{r}$.
6. Write out $\overrightarrow{v} + \overrightarrow{r}'$.
7. $\mathbf{A} = [\overrightarrow{r}', \overrightarrow{v}']$ is a 2 x 2 matrix. Write it out.
8. Write out $\mathbf{A}'$.
9. If we write $2 \times \overrightarrow{v}$, that means you should multiply each element in $\overrightarrow{v}$ by 2. Write out $2 \times \overrightarrow{v}$.
10. Write out $2 \times \mathbf{A}$.
11. Write out the set $\tilde{S} = \{\mathbf{A}, [\overrightarrow{r}, \overrightarrow{v}]\}$.
12. Write out the set $\tilde{S} = \{\mathbf{A}, \mathbf{A}'\}$.
13. Write the matrix $\mathbf{B} = \begin{bmatrix} 2 & 4 \\ 150 & 189 \end{bmatrix}$ in terms of $\overrightarrow{v}$ and $\overrightarrow{r}$.
14. If we write $\mathbf{A}(i, j)$, it means we to are referring to a specific element in $\mathbf{A}$. The first argument (i) refers to the row in $\mathbf{A}$ that we want to look at, and the second argument (j) refers to the column we want to look at. So if we write $\mathbf{A}(0, 1)$, it means we want to look at the number in the first row and in the second column. We always start counting rows and columns from the upper left corner of the matrix. What is $\mathbf{A}(0, 1)$?
15. What is $\mathbf{A}(1, 0)$?
16. If we replaced $\mathbf{A}(0, 1)$ with a "5", what would $\mathbf{A}$ look like?

2 Coordinate systems

In class we saw that if we took a fixed set of vectors and plotted them in different coordinate systems, the resulting object that we plotted changed dramatically. Now we're going to take a fixed object and try to describe it in different coordinate systems. Changing coordinate systems changes the description, but not the structure of the object (so long as we do it right). You'll probably need to print out the next several pages to complete this section.

On the next page (in Figure 1) you'll see that an object that occupies a portion of the sheet of paper. Because this object is defined by a set of six corners, we'll call it the set $\tilde{G}$ (perhaps its a bad map of the Goldman School). Your job is to characterize $\tilde{G}$ using seven different coordinate systems, shown on the following pages. Each coordinate systems describes specific locations on the page. Probably the easiest way to make sure that you properly map $\tilde{G}$ onto each coordinate system is to overlay the two sheets of paper (making sure the corners line up!) and trace $\tilde{G}$.

Each coordinate system has two dimensions, A and B. For simplicity, let's always list the location in the A dimension first and the location in the B dimension second (it could be the other way around and it would be okay, so long as we are clear about which convention we use). So we'll define the object $\tilde{G}$ by listing the set of corners $\{[A_1, B_1], [A_2, B_2], ...[A_6, B_6]\}$.

1. Create a table like the following one to list the corners of $\tilde{G}$ using the various coordinate systems depicted on the following pages. Make sure that both the location and shape of the object are preserved when you list the corners in each coordinate system. Use 1 decimal point of accuracy (we will allow for ± 0.5 in measurement error when grading).

Coord. System	1	2	3	4	5	6
"Cartesian XY"						
"Cartesian IJ"						
"Skew"						
"Polar rectangular"						
"Seuss"						

2. Do you prefer a specific system? Why?

3. Sometimes we document the location of objects using one coordinate system, but then try to reconstruct them using a different coordinate system.

   (a) Taking your description of $\tilde{G}$ using the "Cartesian IJ" system (the set of vectors you wrote down under that column in the table above), try to reconstruct the object by plotting these points out in the "Cartesian XY" system (if you printed this lab, then you have printed 4 copies of this figure). Does $\tilde{G}$ look right? In which ways does it look wrong?

   (b) Taking your description of $\tilde{G}$ using the "Skew" system, try to reconstruct the object by plotting these points out in the "Cartesian XY" system. Does $\tilde{G}$ look right? In which ways does it look wrong?

   (c) Taking your description of $\tilde{G}$ using the "Seuss" system, try to reconstruct the object by plotting these points out in the "Cartesian XY" system. Does $\tilde{G}$ look right? In which ways does it look wrong?

Collaborators

Please list everyone you worked on this assignment with outside of public Piazza discussions.

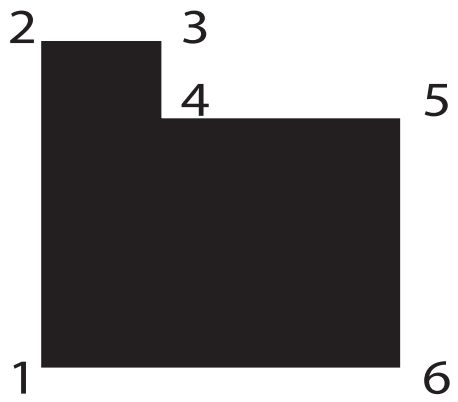

Figure 1: Object $\tilde{G}$ defined by a set of 6 locations on the sheet of paper (one at each corner).

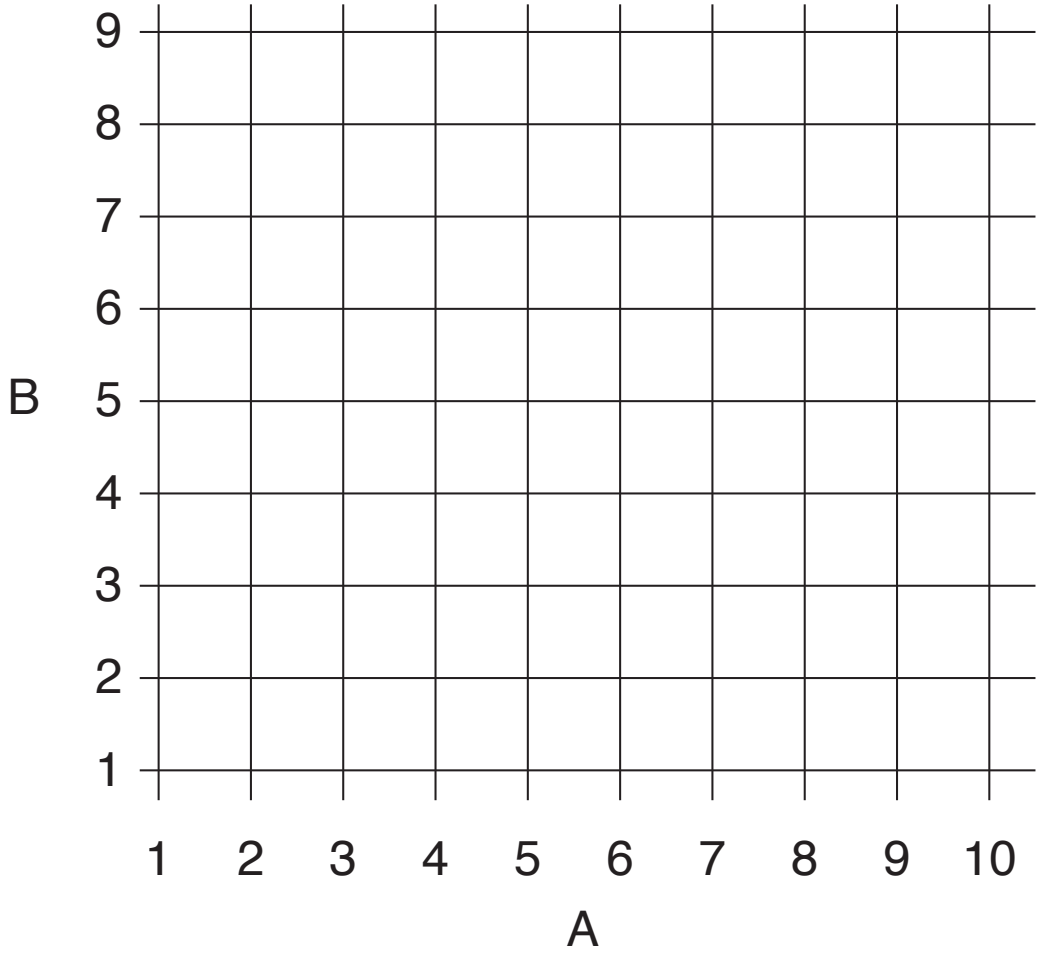

Figure 2: Coordinate system “Cartesian XY”

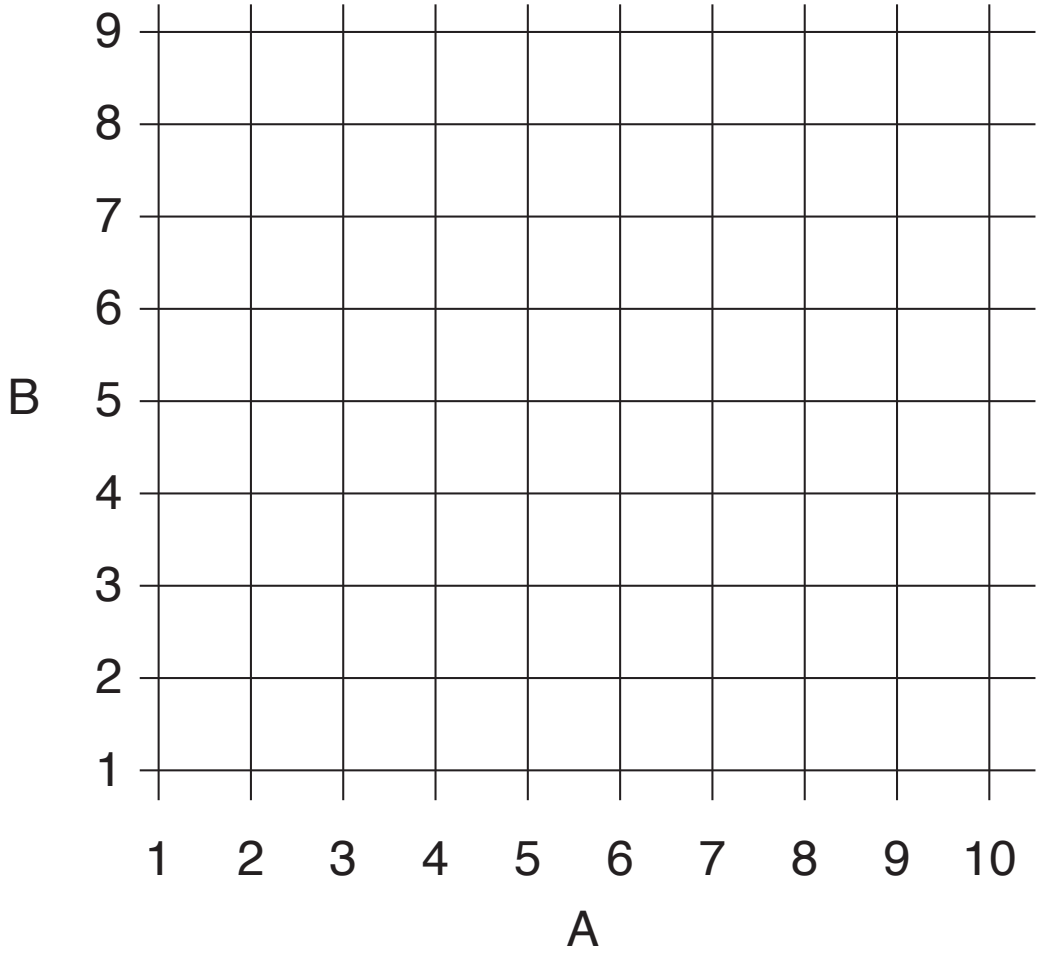

Figure 3: Coordinate system “Cartesian XY”

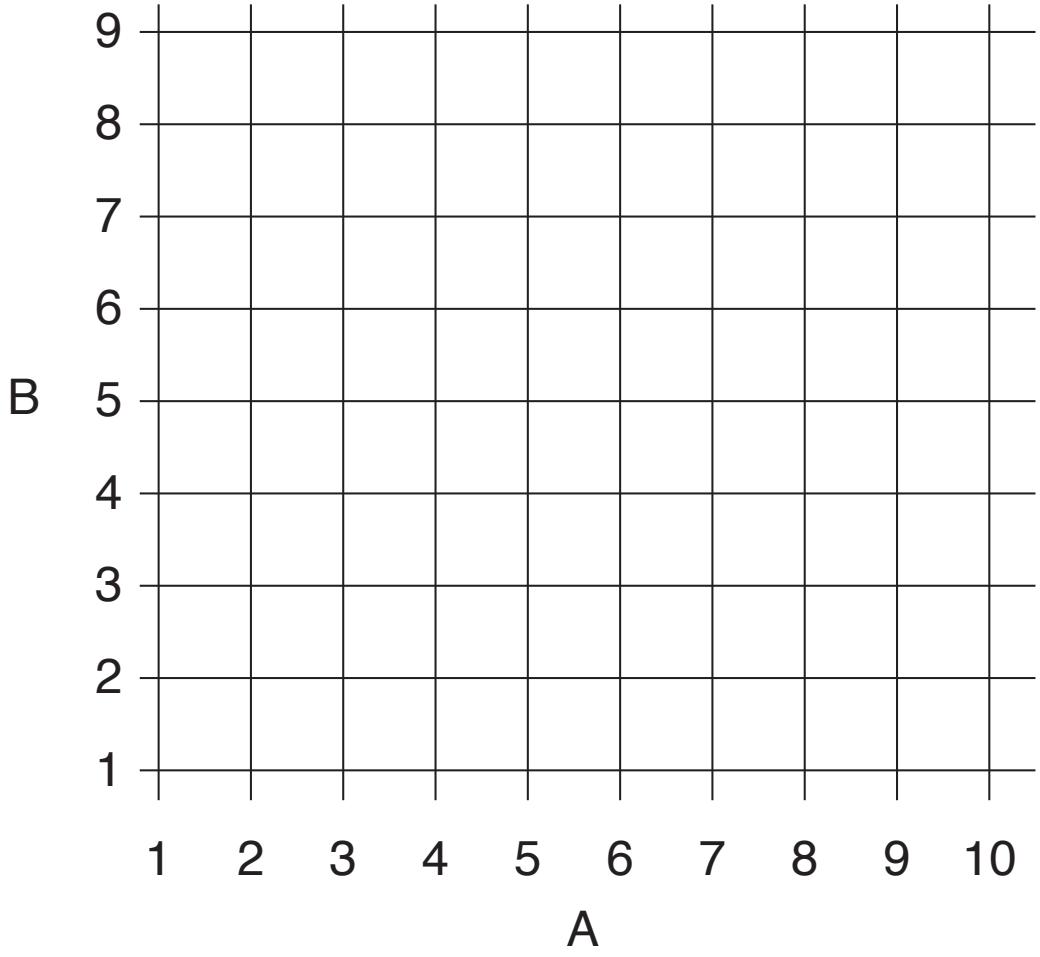

Figure 4: Coordinate system “Cartesian XY”

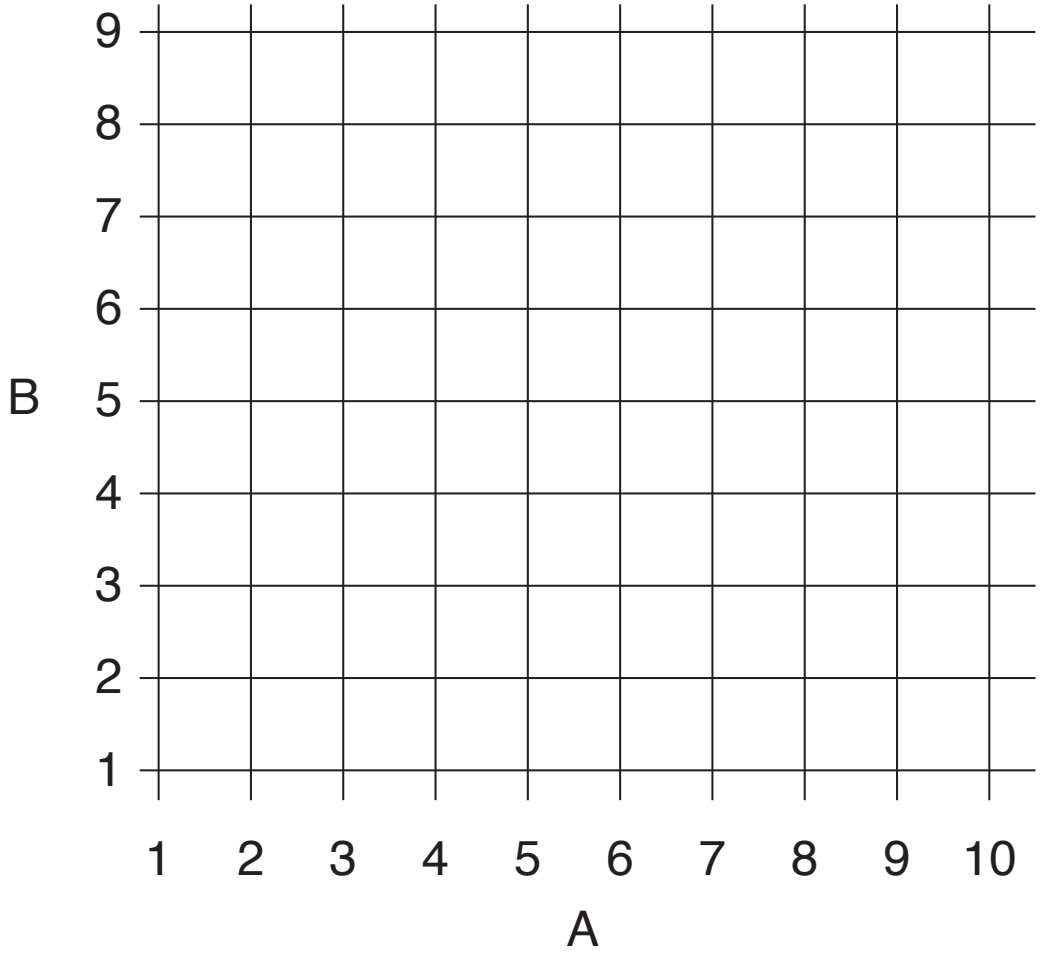

Figure 5: Coordinate system “Cartesian XY”

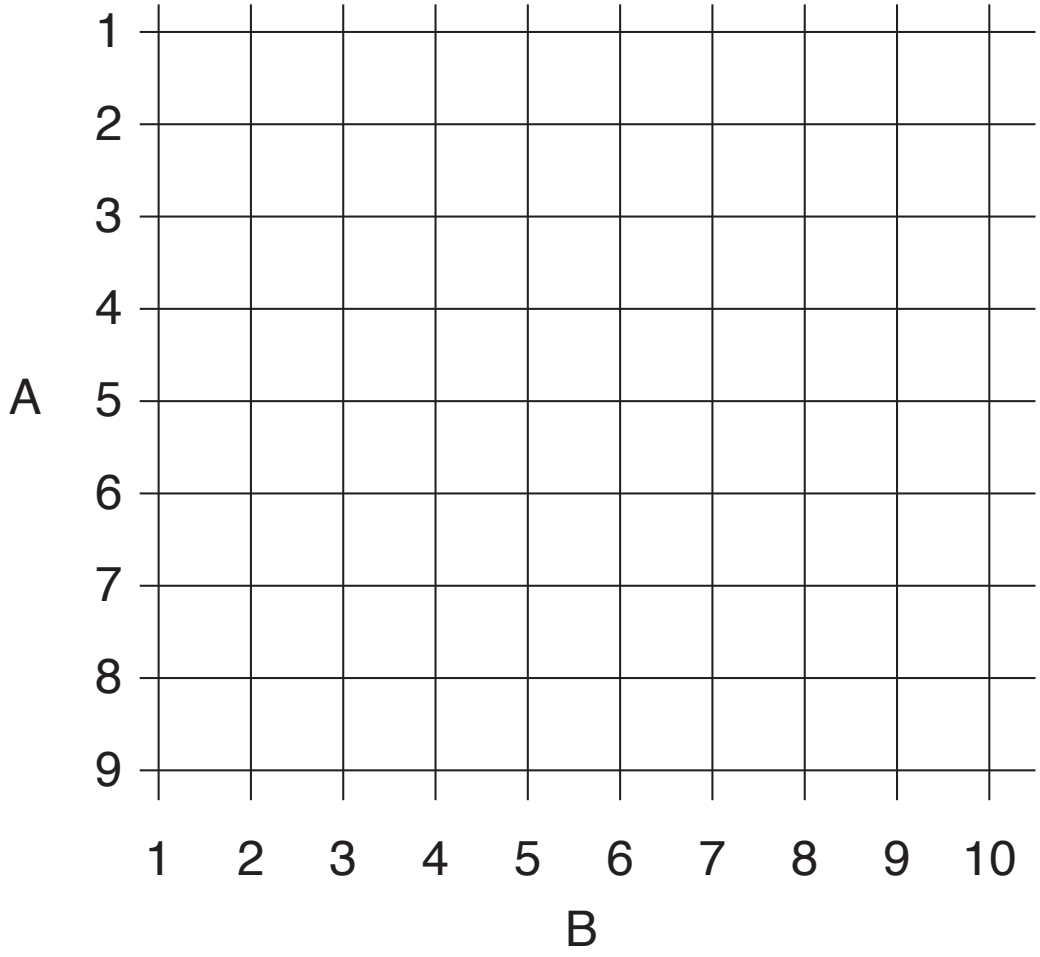

Figure 6: Coordinate system “Cartesian IJ”

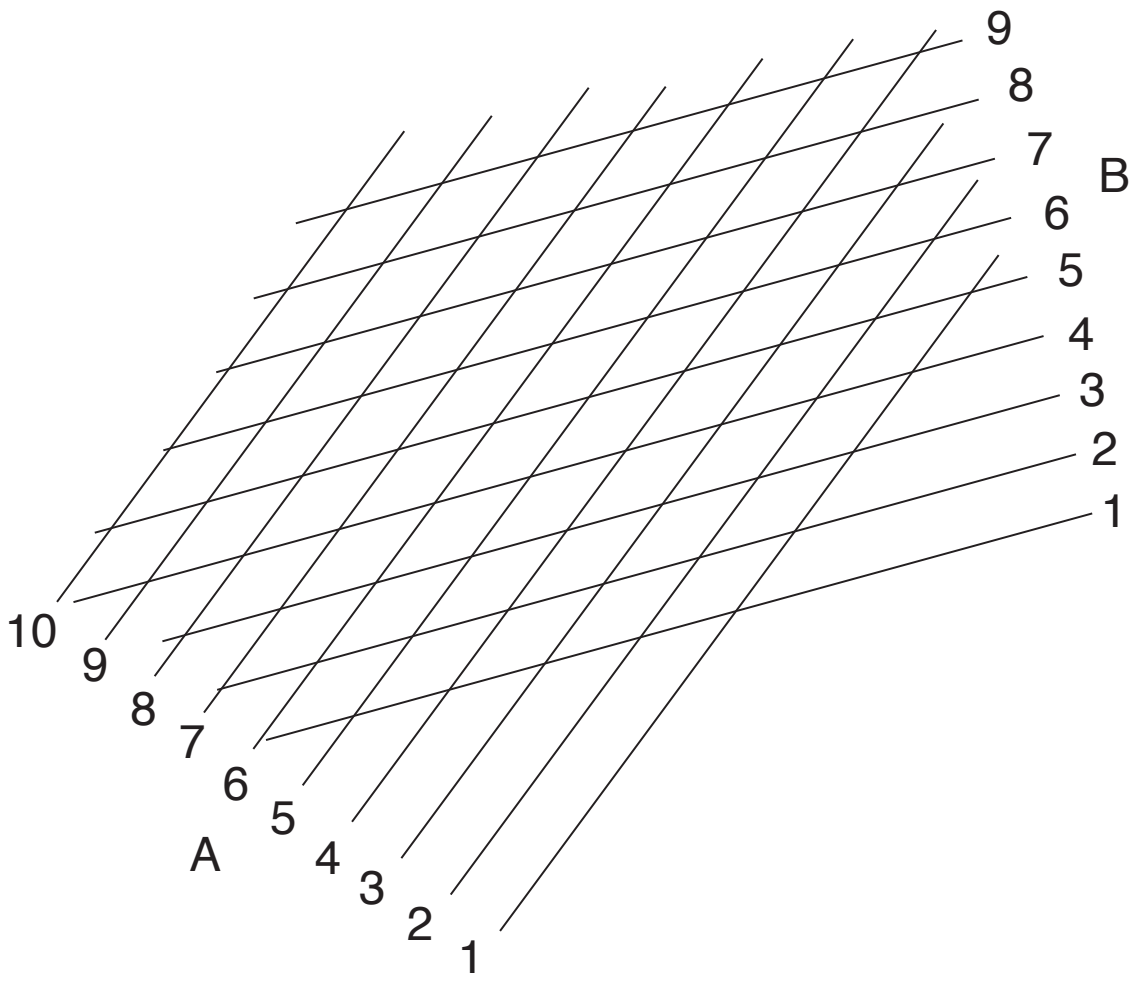

Figure 7: Coordinate system “Skew”

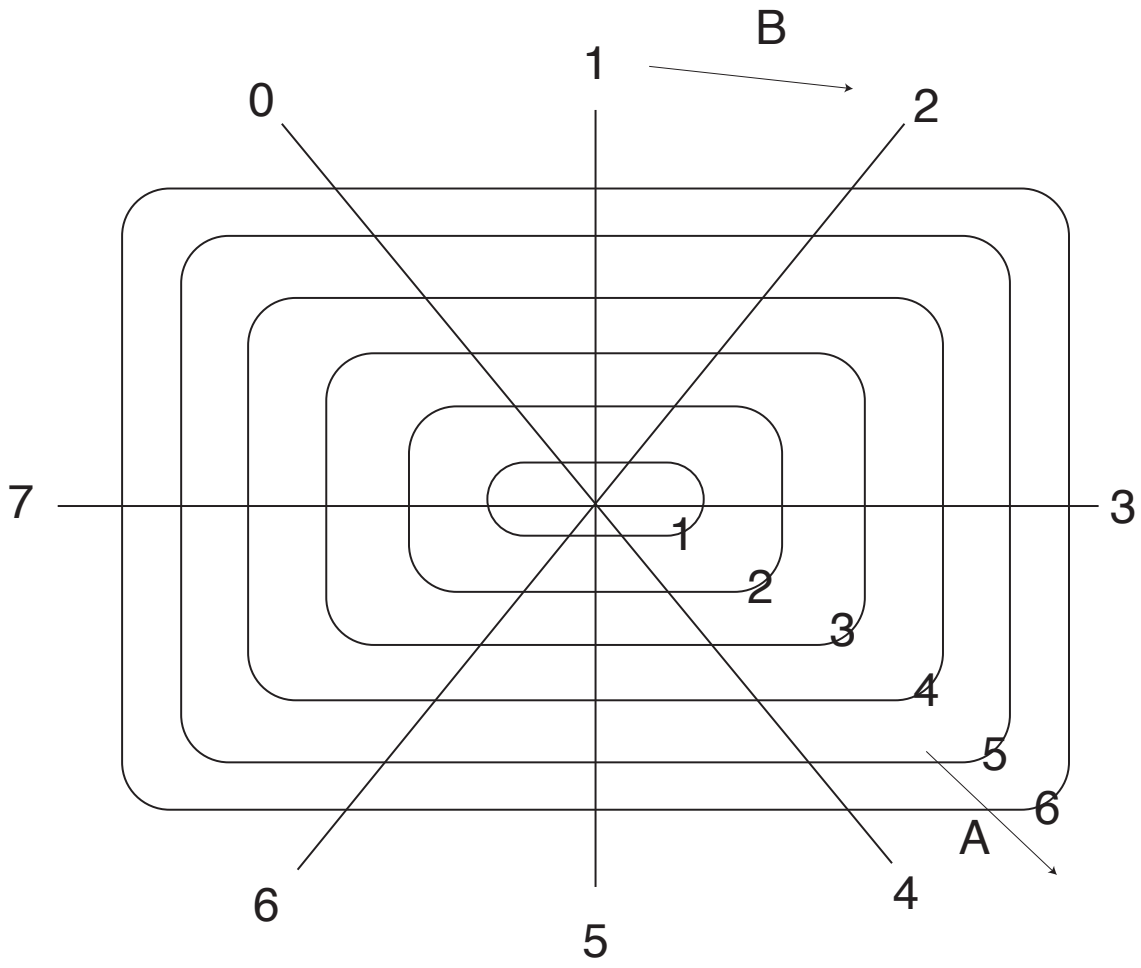

Figure 8: Coordinate system “Polar rectangular”

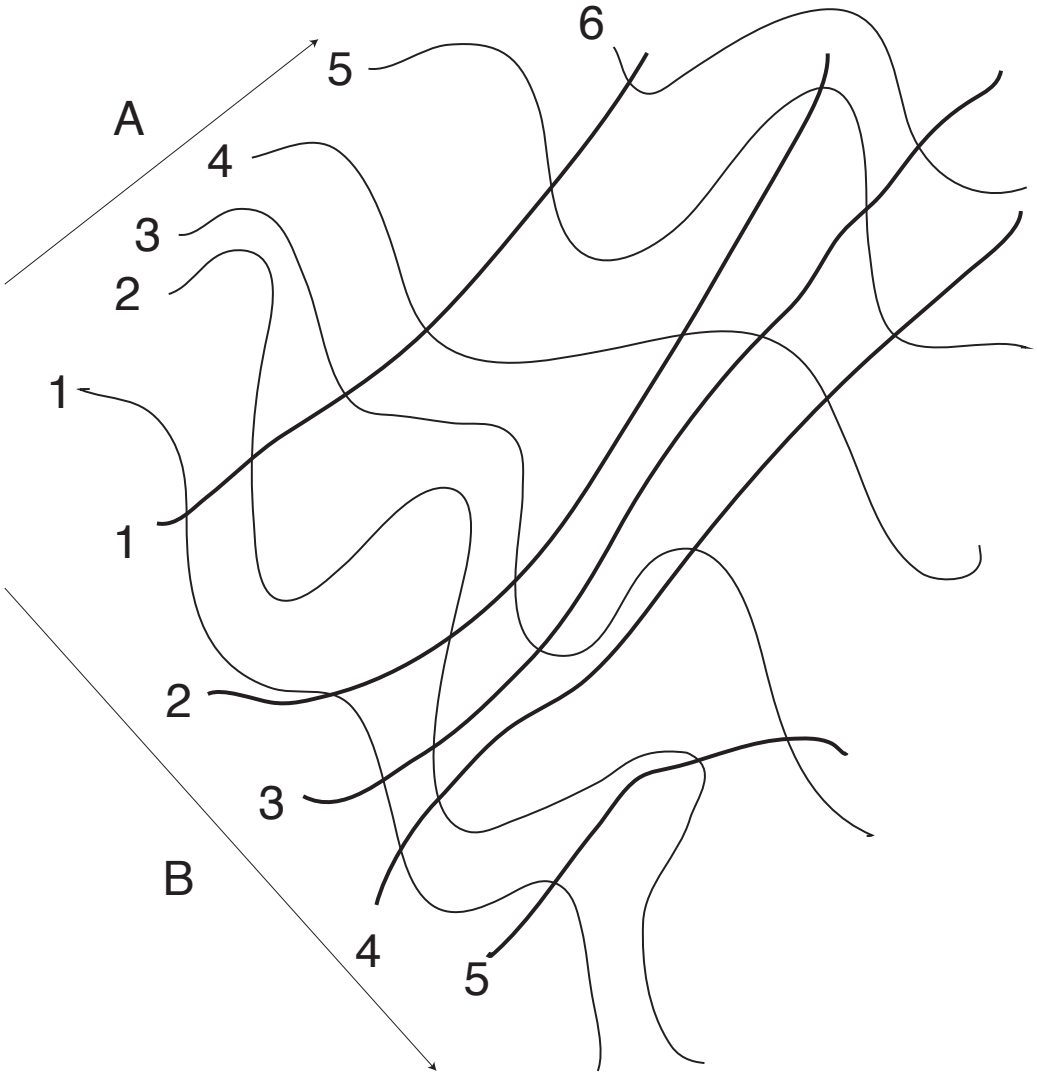

Figure 9: Coordinate system “Seuss”

Python Orientation

Luna Yue Huang

You don't have to turn in any write-up for completing this section, but doing it will help you set up your environment and prepare for future lab assignments.

How to install Python on my laptop? In order to install Python, we need to install its good friend Anaconda. Anaconda is a package management tool for Python, and by installing Anaconda, we are installing most of the common packages in Python (including core Python, `numpy`, `scipy`, `matplotlib`, Jupyter Notebook, etc.).

1. Go to this website and choose an appropriate installer. Since we are teaching Python 3 in this class, please choose the Python 3.7 version.
2. For macOS users, choose the 64-Bit Graphical Installer.
3. For Windows users, you need to first determine your system type (64-Bit versus 32-Bit, see this link for instructions). Choose the installer with your system type.
4. Download the chosen installer. Since these are graphical installers, the installation process is pretty much the same as any other application. Double click the downloaded installer file to start.
5. Follow the instructions. If you are confused, see more instructions here and keep in mind that (1) if asked about whether you want to install Anaconda for "Just Me" or "All Users", choose "Just Me"; (2) in general, do not change the default options (including where anaconda is installed); (3) the installer will prompt you to install other stuff (such as the PyCharm IDE), you can do that if you want to, but we won't be using that in this class.
6. Done!

But where is my Python? - Command Line Basics To answer this question, we have to get you to talk to your laptop via command line first. This will be a useful skill not only for installing Python and various Python packages, but also for installing/using many other applications or packages (e.g., `git`) that you may need.

1. For macOS users, open Launchpad, then click the Terminal icon.
2. For Windows users, from the Start menu, search for and open "Anaconda Prompt".
3. If you are confused, see more instructions here. If you successfully launched your Terminal, you should see something like this (subject to some variations across different operating systems).
   (for macOS users) `YourName@YourLaptopsName:~$`
   or
   (for Windows users) `C:\Users\YourName>`
   Let's talk about what these are. For macOS, the tilde (~) tells you which folder you are currently in. The tilde is a shorthand for "home directory", which usually defaults to `/Users/YourName`. For Windows, the text before > also represents the current directory that you are in. The dollar sign ($) on macOS and the greater than sign on Windows (>) are called the "prompt". A prompt tells you that you should write your codes here. There are no prompts before outputs of your program. People use prompts to distinguish codes from outputs when they post code examples. For example, in the following example
   ```
   $ pwd
   /Users/Me
   ```

this means that on the first line (with a prompt) is my codes (I typed `pwd` and hit Enter) and the output is on the second line. `pwd` is a command that shows you the current directory that you are in (try it yourself!).

4. Try the following:
   `ls` lists files in the current directory (don't do it when you have thousands of files in the directory!);
   `cd directory/where/you/want/to/go` allows you to change your current directory.
   You can't mess anything up by doing these, so go ahead and play with them.

5. Tips: When using the command line, if you press the "up" arrow-button, then Terminal brings up the last command you typed as if you just retyped it. Hitting the up button multiple times brings up your history of commands in reverse order. This can make it quicker to type repetitive commands or to debug a command you are trying to use. Hit the "tab" key and Terminal will attempt to auto complete your commands (specifically, it will auto complete the file names that you typed). Hit "tab" twice and Terminal will list all the file names (in the current directory) while preserving the commands that you just typed. These tips can be very productivity-enhancing if you use Terminal often. (Try it!)

6. Now that you know how to talk to your laptop via command line, end this session by typing `exit`.

The tool that you just used (Terminal or Anaconda Prompt) lets you send commands to your laptop directly, rather than by clicking buttons on graphical user interface (GUI). These tools mostly all use the same language, `Shell`, or more commonly `Bash` (for the purpose of this class, think of them as essentially the same). What you just did (`pwd`, `ls` and `cd`) are in the Shell/Bash language.

But where is my Python? - `which python` In the command line, we can call programs, or executables (think `.exe`), or binaries (for the purpose of this class, think of them as the same). The `pwd` command that you just called is one of these programs. `python` is another one of these programs. The Shell keeps track of which keyword maps to which program, i.e., which executable file you are actually calling when you type a certain keyword.

1. Open another Terminal/Anaconda Prompt session.

2. Ask where your Anaconda is by doing
   ```
   $ which conda
   /Users/Me/anaconda3/bin/conda
   ```
   This is as expected. My anaconda is healthily living in my default home directory (this is the default and recommended setting for macOS users).

3. Now this is the interesting part. Like we discussed, python is just a program/executable. As a matter of fact, you can have several pythons living on your laptop at the same time (this will be the case for most macOS users). The reason is that the macOS system uses Python (we call that the "system Python"). We do NOT want to use or mess with (e.g., update or delete packages) this version of Python on your laptop. Do the following to find out which python is currently in use, if you see the following, you are currently using the system Python (usually Python 2).
   ```
   $ which python
   /usr/bin/python
   ```
   If you are seeing the following, you are using the python living in harmony with your anaconda, which is what we want.
   ```
   $ which python
   /Users/Me/anaconda3/bin/python
   ```
   So how do we activate the python in our anaconda? It's simple.

`$ conda activate`
For macOS users, when anaconda is activated, you should see
`(base) YourName@YourLaptopsName:~$`
where `base` is how anaconda calls its default environment. Also, anaconda should be automatically activated in most cases.

How do I launch my Python? We can use python in many ways. Here are three ways that I find the most useful.

1. We can launch our python by doing
   `$ python`
   You will see that the prompt changed to >>>, which is Python's prompt. You are now in the python environment (and can write python codes and execute them line by line). You can exit the Python environment (and go back to Shell) by typing `exit()`. Note that each Terminal session is independent, so the variables that you define in one session is not "seen" by other sessions. Each time you enter a python environment in Terminal, everything is "reset", and every time you exit by `exit()`, you lose all the variables/outputs that are defined during the session (since you entered the python environment).

2. We can also run our python scripts directly. Open a text editor (what you would use to edit `.txt` files). Type the following into the file
   `print('hello world')`
   and save it as `test.py`. In Shell, `cd` to the directory where `test.py` is saved, and do this
   `$ python test.py`
   `hello world`
   This is to demonstrate two things: (1) there is nothing magical about python scripts, they are just `.txt` files with a different suffix `.py`; (2) the output is generated and shown in Terminal.
   Note: if you want to download a fancier and nicer text editor, Sublime is a good choice. This is of course optional and you won't need it for this class.

3. We can use jupyter notebook. You will be doing your labs in jupyter notebook. Launch jupyter notebook by doing
   `$ jupyter notebook`
   You will see that a jupyter notebook window opens up in your browser. You will see your file system first. To start a new notebook, go to a directory and choose New > Notebook: Python 3. To open an existing notebook, go to its directory and click on it.
   If you are asked to select a kernel amongst a few kernels, that means you have multiple pythons installed on your laptop. Select the anaconda one (Python 3). You don't have to `conda activate` before running Jupyter Notebook.
   To close a jupyter notebook (which is what you should do when you are not using it), go to the file system, find the file, select the file by checking the box in front of it, and click on "Shutdown". One thing to note is that if you close the browser, you don't actually close anything. You can look at your Terminal session, and copy-paste the URL into a browser to launch a new window, and nothing would change from when you closed your browser. To close your jupyter notebooks, press `Ctrl+C` once (or twice, if you want to skip confirmation). `Ctrl+C` is the "KeyboardInterrupt" shortcut in Shell/Bash and it stops whatever program you are running. You should close jupyter notebook when you are not using it, because they do run in the background and use memory.

How to install new packages for my Python? There are two commands that you will use for installing new packages, `conda` and `pip`. In case you are interested, here is how they are different. The TL;DR is that (1) use `conda` whenever possible, if a package is not on `conda`, use `pip`; (2) some packages may not be on `conda`'s default channels, but they are on the `conda-forge` channel.

1. When packages are written, they very often have "dependencies", meaning that they use some functionality from other packages. In order to install these packages, we need to install their dependencies first, and in order to install their dependencies, we need to install their dependencies' dependencies...You can see that we would not want to do this manually, and this is where `conda` and `pip` come in. They both automatically install all the dependencies, and they also keep track of the versions of the packages installed, which can be important. Start a Terminal/Anaconda Prompt channel, activate your conda environment if it's not yet activated (`conda activate`).

2. Type `conda list`, which will show you all the packages that you have now.

3. Type `conda list | grep numpy` (or change numpy to be any other package name of interest). This is asking conda to show all the packages that contain the numpy keyword.

4. Type `conda search gdal` to ask conda if the gdal package is available on one of its channels (an important remote sensing package, many packages, e.g., `rasterio`, depend on it), you will see that it is available and all the available versions are listed.

5. Type `conda install gdal` to install gdal. The installation could take a while.

6. If all goes well (fingers crossed!), no errors should be thrown, and you can type `python` to enter a python environment. Type `from osgeo import gdal` to try to import the package. If you don't encounter any errors, congratulations! The installation is successful.

7. The steps above can help you install packages on conda's default channels. Sometimes the packages of interest may not be in the default channels, but they are likely to be in `conda-forge`. To search in conda-forge, type `conda search -c conda-forge gdal`. To install, type `conda install -c conda-forge gdal`. If I could find a package in the default channels, I would usually install that version.

8. *[optional]* ***`pip`*** *is another program that helps you install Python packages. You may need to use this at some point if a package that you are looking for is not on* ***`conda`****. One of the reasons why* ***`conda`*** *is "better" than* ***`pip`*** *is that conda takes care of dependencies that are not in Python (part of the GDAL packages are in C, for example). Many smaller packages with less sophisticated dependencies are only available on* ***`pip`****, and you can pip install them without getting into trouble. However, I do NOT recommend install gdal with pip.*

9. *[optional] Before you do things, make sure that you are installing the package to the desired conda environment.* ***`which pip`*** *and make sure that pip is inside the anaconda folder.* ***`pip list`*** *to see all your packages.* ***`pip search numpy`*** *to search for packages with the keyword numpy. Finally,* ***`pip install numpy`*** *to install the package numpy (if you already have it, pip won't install a new one).*

If you encounter errors during this process and if they look more complicated than you can handle, get help! You could (1) copy-paste the error message and search for that in Google or Stack Overflow, make sure to follow instructions only if you know what the commands are doing; (2) come to office hours for hands-on help.