NAVIGATING UNIX

Most scientific computing is done on a `Unix` based system, whether a Linux distribution such as Ubuntu, or OSX on a Mac. The terminal is the application that you will use to talk to the machine - to tell it to compile and run programs, to copy data, to login to remote supercomputers, to compile LaTeX, etc. An instance of the terminal is known a shell, and the language most people use in the shell is known as `bash`, with a mix of `Unix` commands. You can find out what any `Unix` command, such as `cd` does in the terminal by typing `man cd` and pressing enter. Googling commands is also helpful. Try to use the following commands:

UNIX COMMANDS

1. `man <command>` - what does `<command>` do?
2. `pwd` - where am I?
3. `ls` - list all of the files in this directory (aka folder)
4. `cd <directory>` - go into `<directory>`
5. `cd ..` - go back a directory, `cd` - go to home directory
6. `mv <one> <two>` - rename `<one>` to `<two>` or move `<one>` to `<two>`
7. `cp <one> <two>` - copy `<one>` to `<two>`

Other useful commands, with more extensive documentation, are

1. `scp` or `rsync` - to copy from remote servers
2. `cat` - to concatenate files
3. `more` or `head` or `tail` - to look at parts of a file
4. `find` - find a file
5. `grep` - search for a string in a group of files

Look up the documentation for these using `man` or Google.

Things that we will be doing in the terminal also include

1. `python program.py` - running `Python` programs
2. `ipython` - starting an interactive `Python` session
3. `pdflatex writeup.tex` - compiling LaTeX

Some people also choose to use terminal-based text editors such as `vim` or `emacs` to write `Python` code, LaTeX, and other things. Typing `vimtutor` into your terminal will bring up a tutorial program for `vim`, if you wish to use `vim`.

MAKING A DOCUMENT USING LaTeX

Most of the physics papers you will come across (including this assignment for what it's worth) have been written up using LaTeX, which is a document preparation system which makes writing mathematical expressions very easy. The flow of writing a LaTeX document includes editing a `.tex` file, and compiling it with `pdflatex` to generate a `.pdf` file (like this document).

You can edit a `.tex` file using a text editor like `vim` or `emacs`. Certain applications which can make dealing with errors easier also exist (for my mac I like TeXShop, for example). Usually people start a `.tex` file using a template, and there are many to choose from. Some good ones can be found online, but we have also provided one here. Math can be inserted into the text using dollar signs around symbols like `$\pi$` for example to generate π. Equations can be inserted using `\begin{equation}` and `\end{equation}`. Here is a piece of example LaTeXcode:

```
% This is how you make a comment
\documentclass[8pt]{article}
% For pictures
\usepackage{graphicx}
% For math
\usepackage{amssymb}
\usepackage{amsmath}

\begin{document}

% Title
\title{Assignment Title}
\maketitle

% Sections and text
\section{Example Section 1}

I can write symbols like $\pi$, $\alpha$, $\rho$

\section{Example Section 2}

This is my second example section, with my example equation:
% This is an equation
\begin{equation}
\dfrac{\pi}{4} = \sum_{k=1}^{\infty} \dfrac{(-1)^{k+1}}{2k - 1}
\end{equation}

\end{document}
```

The body is between `\begin{document}` and `\end{document}`, and it's where you'll put all of your text and equations. Note that we have included the `amssymb`

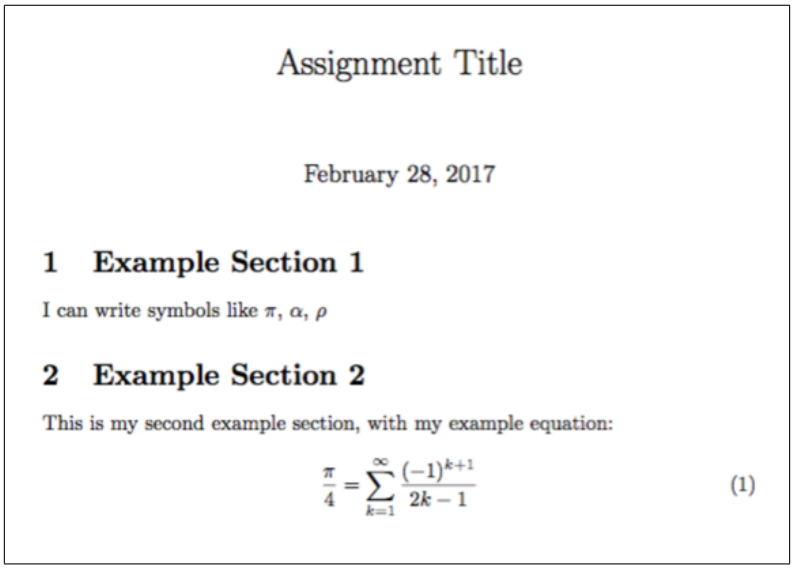

FIG. 1: This is a screenshot of the pdf generated from the above `.tex` code.

and `amsmath` libraries for math write-up, and the `graphicx` library to include figures.

Google is a very useful resource when trying to figure out how to LaTeXsomething. There are many useful responses on the TeX stack exchange, and a useful list of mathematical symbols can be found on ShareLatex (you can click this link!).

Now that we have a `.tex` file, let's say `hw0.tex`, we can compile it in the Terminal using the command

```
pdflatex hw0.tex
```

This should generate a `hw0.pdf` file. You can then open this file using `open hw0.pdf` on a mac or `evince hw0.pdf` on the lab Unix computes, for example. The example code above generates the pdf screenshotted in Figure 1.

One final thing to start with is how to include figures in your document. If you have a figure called `MyFigure.png` in the directory you're compiling the `.tex` file in, you can add the following to the body:

```
\begin{figure}
\centering
\includegraphics[width=\textwidth]{MyFigure}
\caption{This is the caption}
\label{fig:MyFigureLabel}
\end{figure}
```

The `\label` will let you reference the Figure, using `\ref{fig:MyFigureLabel}` to produce something like the Figure 1 reference I had before.

COMPUTING AND PLOTTING WITH PYTHON

Python is a scripting language (meaning it's not compiled like C++, for example) that's ubiquitous in scientific research because it's easy to use and has a lot of nice libraries written for it. It isn't the fastest language, so languages like C++ are better for running high-tech simulations, but Python is nice for data analysis, making plots, and doing various other scientific computing. In this course, we will be using Python 2.7.

A Python program will have a .py extension, and can be run from the terminal using python myprogram.py. Python can be written in emacs, vim, or whatever text editor you prefer. A good flow to have is one terminal window where you will write your program (or a text editor window), and another where you will test running it, and then switch between these windows to debug. You can also write ipython notebooks, which are closer in style to a mathematica instance.

Official Python documentation, including a tutorial, is found on python.org. In this course, we will also be using numpy for numerical computation with arrays, scipy for some data analysis features, and matplotlib to generate plots.

Here is an example Python program, that generates a figure of a sine wave. Try writing something similar and tweaking it, or looking at documentation, to see what various things do.

```
# Comments are written like this
# Including the libraries that we will need
import numpy as np
# now I can use functions, and write 'np' instead of 'numpy'
import matplotlib.pyplot as plt

# Here is a function
def myfunction(x):
  """ This is a function comment
      it can span multiple lines """
  return np.sin(x) + 1.0

# Generating my x array with numpy
x = np.linspace(0.0, 6.0, 100)
# Applying my function
y = myfunction(x)

# making my plot
plt.figure()

# plotting my data using fancy keyword arguments
# to set the color and linewdith.
plt.plot(x, y, linewidth=2.0, color='red', label='mydata')

# Set labels and title
```

```
plt.xlabel('X', fontsize=24)
plt.ylabel('Y', fontsize=24)
plt.title('My Experiment', fontsize=30)

# Save my plot
plt.savefig('MyExperiment.png')

# Show my plot!
plt.show()
```

Since the documentation for `Python` is pretty good, below we have only included a list of useful commands - feel free to explore these yourself!

USEFUL NUMPY Assuming we have `import numpy as np`

1. `np.multiply`, `np.divide`, `np.add`, `np.subtract`, `np.sum`, `np.cos`, `np.sin` - various math functions on arrays
2. `np.zeros` - array of all zeros
3. `np.linspace` - array of numbers in a given range
4. `np.random` - random numbers
5. `np.loadtxt` - reading from a file

USEFUL MATPLOTLIB Assuming we have `import matplotlib.pyplot as plt`

1. `plt.figure` - make a figure
2. `plt.plot`, `plt.scatter` - plot
3. `plt.xlim`, `plt.ylim` - set the limits
4. `plt.xlabel` - set axis labels
5. `plt.legend` - make a legend if you have `'label = somelabel'` in your `plt.plot` call
6. `plt.xscale('log')` - set a log scale
7. `plt.show` - show the plot
8. `plt.savefig` - save the figure
9. `np.loadtxt` - reading from a file

Note that all of the `matplotlib` commands can have plenty of keyword arguments (if you Google for the documentation of each). For example, you can set the fontsizes of axis labels, the frame on the legend, etc.