Introduction to Overleaf, LaTeX, and Tikz

Contents

1. Intro and First Login

When it comes to writing up math, there's nothing better than LaTeX (unless you do stat, and R Markdown is your jam, but even that uses LaTeX!). It is bar none the best typesetting software there is. It's what was used to print our textbook. That being said, it can be intimidating for first time users (and second time users). This guide should help you get on your feet when it comes to LaTeX.

This primer is written for Overleaf, an online TeX compiler. There are other options for using LaTex (including CoCalc, ShareLaTeX, and many applications for using LaTeX locally), but Overleaf is probably the most well known. To get started with Overleaf, first create an account. You will receive a confirmation email, which will prompt you to create a password for your account. Once you do this, you'll be ready to go!

2. Overleaf

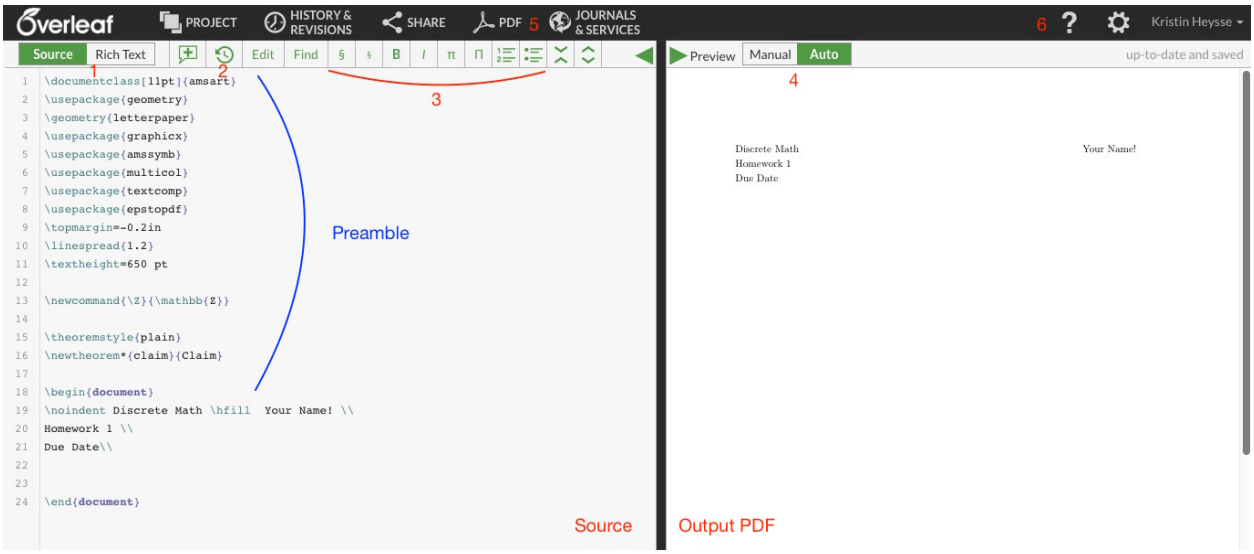
When you want to create a new file, you've got two options with Overleaf. The first is to use one of the built in templates. Overleaf has a whole bunch for you to choose from, and you can find one that you like. The second option may be easier for your first few assignments, and that's to use the Homework Template from Moodle.

2.1. **Setting up files.** You can certainly feel free to organize your Overleaf files however you want, but I would recommend that you make use of its *projects* interface. A project is where you can store all your LaTeX files of a certain type. You can do this by content (like daily questions, homework sets, etc) or maybe by unit (logic, proofs, counting, etc).

(1) The first thing you'll want to do is download the LaTeX Homework Template from the Moodle page. You need not open it locally, but you might want to save it in a place where it can be easily accessed.
(2) In the top left part of Overleaf dashboard, you'll see a new project button. Pick the blank paper option.
(3) When you get to where you can edit (see the picture in Section 2.2), click on the Project header button. It should turn green and open a sidebar.
(4) Next to files, you'll find a drop down menu. Pick upload from computer, and select the homework template. (Feel free to delete the main.tex file.)
(5) Rename your file to reflect the assignment! You don't want a whole bunch of Homework Template files.

When you want to create a new assignment, you can either upload the homework template again or copy and paste the preamble into a blank file.

2.2. **The Overleaf window.** Once you've opened your file, you'll be shown a window like the one below.



What we see are two files, side by side. The one on the left is the *source*. This is where you do all your typing and editing. On the right is the *output pdf*. LaTeX builds files by compiling the source into a nice looking pdf.

Everything before the line `\begin{document}` is called the *preamble*. It's where we specify all the style elements of the document. Once we begin the document, that's what shows up in the pdf. Try typing a line of text before the `\end{document}` command. If you wait a few seconds, the line you typed will show up on the pdf side.

Overleaf has a lot of great features, feel free to explore them when you have time. I've highlighted a few below:

(1) Source versus Rich Text: Overleaf has a rich text mode which essentially is like a "half compiled" version of your source. It'll compile some of your code but not all of it. I personally like using the source, but find what works for you!
(2) History: If you made a change or you need to go back and rescue a deleted block of code, this will help track and restore your changes!
(3) Some macros: Here are some buttons for common TeX commands, including adding sections and math environments (more about those in the LaTeX section).
(4) Compilers: Overleaf has manual or automatic compiling of your source to your PDF. You can pick it here. If you are making a lot of changes and you don't want it updating every five seconds, you can pick manual. Just make sure you compile before you download your PDF!
(5) PDF: here's how you'll get your PDF to your computer to print or digitally hand in.

3. LaTeX

3.1. **The preamble.** The preamble is where we set all the stile elements of the file. Let's go through it line by line for the homework template.

- Lines 1-3: These set the document class and the type of paper. We'll use *amsart* which is short for AMS (American Mathematical Society) article. It has lots of good built in commands we can use for our homework.
- Lines 4-8: These are the packages we'll need. You don't need to keep track of these unless you need to add some later. Essentially what these do is make more options available to us in our work. That includes multiple column environments (multicol), adding graphics (graphicx), and using AMS symbols (amssymb). You can google these if you want.
- Lines 9-11: These set the dimensions for the text and the line spread (or the spacing between lines).
- Line 13: This is cool! This is called a macro, and it defines a new command in my file. Suppose I am using the symbol $\mathbb{Z}$ a lot. Well, each time I do that, I have to write out `$\mathbb{Z}$` every time. Maybe instead I'd rather just write `$\Z$`. That's what this line allows me to do! You don't have to use these, they just help save time.

- Lines 15-16: This builds my claim environment (more on those later). It just means I can make a claim in a certain style.

Alright, we've made it to begin document. Here's where we'll start typing our work.

3.2. **Commands versus environments.** When working in LaTeX, you have both *commands* and *environments*. Commands are for fairly simple tasks: making a mathematical symbol, bolding some text, etc.

Environments are for more involved jobs, like making a list or typesetting equations. Environments often have commands that are specific to them. For example, the `\item` command doesn't work outside a list building environment. Similarly, a math symbol like `\frac` doesn't work outside of a math environment. You will need to begin and end environments with the `\begin{}` and `\end{}` code.

As soon as you start typing a command or environment, Overleaf will give you some options that you might want. This is great for if you don't know exactly what you're looking for! It will also insert complete environments for you, which lessens the probability that you accidentally forget to close your environment.

3.3. **Typesetting Text.** For most text, you'll be able to type normally. There are a couple of things you may want to know, though.

- *I want to make a new line.* Either do a double enter (leaving a blank line in your tex file) or use the double backslash: `\\`.
- *I want to make something bold.* Use the command `\textbf{}`. Put the text you want bolded in the curly braces. For example, `\textbf{I love math}` builds as **I love math**.
- *I want to make something italic.* Similar to bolding, use `\emph{}`.
- *I want to make a bulleted list like this one.* Getting fancy! What you want is an *itemize* environment. We make one with the code `\begin{itemize}`. Then, every item in the list starts with the command `\item`. When you're done with your list, make sure to close out the environment with `\end{itemize}`.
- *I want to make a numbered list.* Same sort of rules apply to numbered lists as with itemized lists, but the environment is called *enumerate*. What's great is that you don't need to track the numbering, you will still use `\item` and LaTeX does the numbering for you.
- *I need some blank space.* You'll want either `\hspace{}` or `\vspace{}`, depending on if you want horizontal or vertical space, respectively. Inside the curly braces you'll put the amount of space you want. These commands takes lots of parameters, like in for inches, cm for centimeters, or pt for points. For example, `\vspace{1 cm}` makes the space below.

- *I want to make a table.* This is one of the things that LaTeX makes really complicated. I suggest using a online generator. Tables Generator (http://www.tablesgenerator.com/) is the one I use, essentially you fill it like a spreadsheet and put the generate button. The website will generate the LaTeX code for you, which you can then copy and paste into your source.
- *I want to center some text.* Use the *center* environment.

3.4. **Adding Images.** Sometimes it's too hard to make a picture in TeX (there are tools we will discuss later) and it's just better to just insert an image. TeX makes this pretty easy with the following code:

`\includegraphics[parameters]{imagename.extension}`

Parameters help you set the image size (width = something, scale=a decimal in [0,1] are examples) while the image name and extension are how you call the image.

Note! your image needs to be stored where your source code is, meaning you need to upload it to Overleaf and keep it with your document.

3.5. **Typesetting Math.** This is where LaTeX really does well. When you want to write math, you will first need to pick the right math *environment*. Below are the most commonly used:

- *Inline Math:* This is for when you want your math to be in line with the text. Any time you talk about a variable, you will use inline math mode. It's also good for small equations that aren't worth taking a large line.

  How to: Use single dollar signs or `\( \)` around the mathematics.

  Example: `$n$` produces n, `\(a+3 \geq 4\)` produces $a + 3 \geq 4$.
- *Centered equations:* When you want an equation to sit by itself on its own line, use this type of environment. This is best for very involved equations or anything that needs to referenced later in your proof. If you have a summation, an integral, or anything else big, you really should use a centered equation.

  How to: There are a few options here. If you want a centered single line of math use `\[ \]` with the math inside the brackets. If you want to make a numbered equation that you can reference later, use the equation environment `\begin{equation}`.

  Example: `\[ \int_a^b f(x)dx \equals F(a)-F(b)\]` produces

$$\int_a^b f(x)dx = F(b) - F(a)$$

  while `\begin{equation} \frac{d}{dx} f(x) \equals f'(x) \end{equation}` produces

$$\frac{d}{dx} f(x) = f'(x) \tag{1}$$

  (You can just use an equals sign instead of the equals command).
- *Align*: An align environment is great for when you have multiple lines of equations showing your work.

How to: Using an align environment, put an ampersand & at the parts you want to align and break your lines with a double backslash \\. If you add an asterisk to your align environment, the lines will not be numbered.

Example: The code

```
\begin{align*} a+b &\leq a+c \\
&\leq a+\frac{1}{2} \\
b &\leq \frac{1}{2}
\end{align*}
```

produces

$$\begin{aligned} a+b &\leq a+c \\ &\leq a+\frac{1}{2} \\ b &\leq \frac{1}{2} \end{aligned}$$

Notice how the fronts of the less than or equal to signs are aligned because that's where we put the ampersand.

3.6. **Typesetting Symbols.** Now that we have a math environment in mind, we'll want to start writing some math. There's no way that I could possibly put all the symbols you would need in this guide, but luckily someone has already done that for us! You've got a few options when it comes to symbol hunting.

- Detexify (`http://detexify.kirelabs.org/classify.html`) is a website which allows you to draw mathematical symbols and it returns to you a few options that you might be looking for.
- The online LaTeX wikibook (`https://en.wikibooks.org/wiki/LaTeX/Mathematics`) has pretty good list of symbols and operators and how to use them.
- If you need to find something really particular and have a lot of time on your hands, go to the Comprehensive LaTeX Symbols List (a 338 page pdf you can find through Google). Beware, you may need to install a package to use the symbol you want!

3.7. **Typsetting Proofs.** When we get to our proof writing, we can use LaTeX's built in environments. The biggest things will be the *claim* and *proof* environments. They work just like any other environments in that you need to begin them and end them. For example, suppose I ask you to prove that if a is an even integer, then a^2 is an even integer.

(1) First thing we want to do is write out the claim. We'll do that in a claim environment with the code

```
\begin{claim}
If $a$ is an even integer, then $a^2$ is an even integer.
\end{claim}
```

which produces

Claim. *If a is an even integer, then a^2 is an even integer.*

(2) Now we'll go to a proof environment to write up our proof. We'll use the code below

```
\begin{proof}
Let $a$ be an even integer. Then $a$ can be
written as $a=2k$ for some integer $k$.
It follows that
\[a^2=(2k)^2=4k^2=2(2k^2)\]
which is even. This completes the proof.
\end{proof}
```

That code produces:

Proof. Let a be an even integer. Then a can be written as $a = 2k$ for some integer k. It follows that

$$a^2 = (2k)^2 = 4k^2 = 2(2k^2)$$

which is even. This completes the proof. □

Notice how I mixed inline math with centered equation math. That string of equalities really needed it's own line to be clear. I also get a proof box for free at the end of my proof environment.

(3) The final result is

Claim. *If a is an even integer, then a^2 is an even integer.*

Proof. Let a be an even integer. Then a can be written as $a = 2k$ for some integer k. It follows that

$$a^2 = (2k)^2 = 4k^2 = 2(2k^2)$$

which is even. This completes the proof. □

3.8. **Troubleshooting.** Without fail, you will have problems with your TeX which will make it not be able to compile (or typeset). Compiling your pdf often will help you find those errors! Sometimes the compiler will give a helpful error message, but sometimes it will not. Here are a few things to check:

- Ended all your environments in the appropriate spots.
- Close all your curly braces. It helps to count up when they open and count down when they close.
- Check the spelling on your commands and environments.
- Add any needed packages (see me if you need help with this one).
- Don't worry about underfull or overfull boxes unless your words go off the edge of the paper.
- If you're working locally, delete your auxiliary files and try again. These are the files that LaTeX generates when it builds your document.

4. Tikz

If you're reading this, we've probably started to learn about graphs. While graphs are basically the coolest thing ever (bias: I'm a graph theorist), LaTeX isn't great at making them. We have to use a package called Tikz, and it has an even steeper learning curve than LaTeX.

Disclaimer: you are not required to make your images in Tikz! An image made on the computer or drawn, scanned, and inserted is just as good. Tikz is just a tool that graph theorists like.

4.1. **Preamble.** Tikz needs its own package that does not come default with Overleaf. This is easy enough to fix, add the following line to your preamble:

```
\usepackage{tikz}
```

Tikz also has lots of libraries that you can call on if necessary. I'll mention a few in the rest of this guide, but if you need more help with it, come see me.

4.2. **Setting up a Tikz Picture.** Tikz images are built in the environment tikzpicture. Let's start with a really boring image, a single vertex (or K_1, if you're feeling mathy).

```
\begin{center}
\begin{tikzpicture}
\tikzstyle{knode}=[circle,draw=black,thick,inner sep=3pt]
\node (1) at (0,0) [knode] {};
\end{tikzpicture}
\end{center}
```

This code creates the image

Let's go through this:

- The first two lines (and the last two lines, for that matter) are pretty straightforward if you've been using LaTeX a lot. These create a centered tikzpicture.
- The command

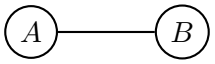
  ```
  \tikzstyle{name}=[parameters]
  ```

  tells Tikz how I want it to draw my vertices. I name the style knode (because I tend to name things after myself) and then I give it some parameters. This includes the shape (circle), how I want it drawn (black, thick) and how big I want it to be (inner sep=3pt). You can try to putz with some of the numbers to see what changes. There are lots of different parameters that can be helpful, and Google is your best friend with this. You can also define lots of styles with different names and call on them when you need them.
- The `\node` line tells Tikz to draw a vertex. Here's how it works:
  - we declare the node, `\node`
  - give it a name: (1)
  - give it a location: at (0,0) (Tikz will take a lot of options here, which we'll get into later)

- tell Tikz how to draw it: [knode], the style we defined earlier
- and what you want to put in it: we put nothing.
- Always end with a semicolon!

4.3. **Edges and for loops: from K_1 to K_2.** Now suppose I want to create K_2, or the complete graph on two vertices. This means I need two vertices and an edge between them. Here we go!

```
\begin{center}
\begin{tikzpicture}
\tikzstyle{knode}=[circle,draw=black,thick,inner sep=3pt]
\node (1) at (-1,0) [knode] {A};
\node (2) at (1,0) [knode] {B};
\foreach \from/\to in {1/2}
\draw[thick] (\from)--(\to);
\end{tikzpicture}
\end{center}
```



So let's discuss the differences in the code. I've now got two nodes, (1) and (2), which I've labeled A and B by putting text inside the curly braces at the end of the node command. The biggest change is the for each command, which draws the edges. We'll go through it a piece at a time:

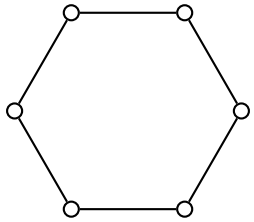
- The command `\foreach` starts the loop
- `\from/\to` tells the loop what sort of objects it's going to find in the list. Essentially I'm saying I'm going to give you a pair, separated by a front slash, which I will reference as from and to.
- `in {1/2}`: this builds the list of vertices you want to draw edges between. I reference the nodes by their *given label*, what we put in to the command, not what we see inside the node.
- `\draw[thick] (\from)--(\to)` : this draws the edge. The draw command takes parameters (thick) just like nodes do, and the double dashes give a straight line. We need the parenthesis around from and to because that is how the nodes are actually labeled.
- Don't forget the semicolon at the end, that closes the loop!

For loops are incredibly helpful, and this only scratches the surface of what they can do for you in Tikz.

4.4. **Node placement: cartesian versus polar.** Sometimes it's more helpful to define your node location in *polar coordinates* than cartesian ones. For example, suppose you wanted to draw a nice C_6. You could break out your trig and figure out what the x and y coordinates need to be for every one, but in polar it's easy! *The biggest difference: instead of using an ordered pair (x, y) to define your node placement, we use a pair $(\theta : r)$.* For example:

```
\begin{center}
\begin{tikzpicture}
\tikzstyle{knode}=[circle,draw=black,thick,inner sep=2pt]
  \node (n1) at (0:1.5cm) [knode] {};
  \node (n2) at (60:1.5cm) [knode] {};
  \node (n3) at (120:1.5cm) [knode] {};
  \node (n4) at (180:1.5cm) [knode] {};
  \node (n5) at (240:1.5cm)  [knode] {};
  \node (n6) at (300:1.5cm) [knode] {};

\foreach \from/\to in {1/2,2/3,3/4,4/5,5/6,6/1}
\draw[thick] (n\from)--(n\to);
\end{tikzpicture}
\end{center}
```



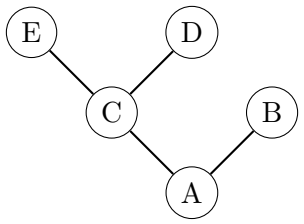
You'll also notice that I give my radius in centimeters. The same goes with cartesian (you can use centimeters, inches, pt, etc).

4.5. **Calculated coordinates.** A great thing Tikz does is calculated coordinates, or essentially defining vertex placement in terms of other vertices. I find this super helpful, particularly when making more complicated graphs. You'll need to add the following line to your preamble:

```
\usetikzlibarary{calc}
```

Here's an example:

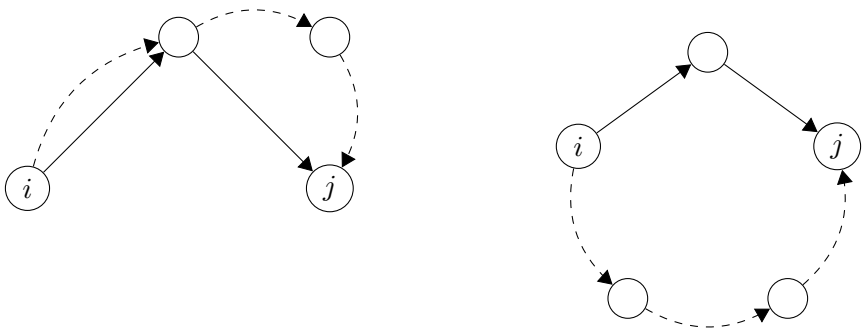
```
\begin{center}
\begin{tikzpicture}
\tikzstyle{knode}=[circle,draw=black,align=center, inner sep=3 pt]
\node (1) at (0,0) [knode] {A};
\node (2) at ($(1)+(45:1.5)$) [knode] {B};
\node (3) at ($(1)+(135:1.5)$) [knode] {C};
\node (4) at ($(3)+(45:1.5)$) [hnode] {D};
\node (5) at ($(3)+(135:1.5)$) [hnode] {E};
\foreach \from/\to in {1/2,1/3,3/4,3/5}
\draw[thick] (\from)--(\to);
\end{tikzpicture}
\end{center}
```



Look at the code for node (2). We want it to be up 45 degrees and 1.5 units from node (1), so we define it's location with a calculation. Notice the notation here: we use parentheses furthest out, then dollar signs, then the calculation.

4.6. **Example.** Just a few more things you can do in Tikz! You'll need to add the Tikz library arrows for the first image.

```
\begin{center}
\begin{tikzpicture}[>=triangle 60]
\tikzstyle{knode}=[circle,draw=black,text width = 15 pt,align=center,inner sep=0pt]
\tikzstyle{hnode}=[circle,draw=black,inner sep=0pt]
\node (l) at (-4,0) {};
\node (r) at (3,0) {};
\node (1) at ($(l)+(-2,0)$) [knode] {$i$};
\node (2) at ($(l)+(2,0)$) [knode] {$j$};
\node (3) at ($(l)+(0,2)$) [knode]  {};
\node (4) at ($(l)+(2,2)$) [knode]  {};
\path (1) edge [->,above] (3)
(3) edge [->,above] (2);
\path (1) edge [bend left, dashed, ->,above] (3)
(3) edge [bend left, dashed, ->,above] (4)
(4) edge [bend left, dashed, ->,above] (2);
\node (5) at ($(r)+(18:1.8cm)$) [knode] {$j$};
\node (6) at ($(r)+(90:1.8cm)$) [knode] {};
\node (7) at ($(r)+(162:1.8cm)$) [knode] {$i$};
\node (8) at ($(r)+(234:1.8cm)$) [knode] {};
\node (9) at ($(r)+(306:1.8cm)$) [knode] {};
\path (7) edge [->,above] (6)
(6) edge [->,above] (5);
\path (7) edge [bend right, dashed,->,above] (8)
(8) edge [bend right, dashed,->,above] (9)
(9) edge [bend right, dashed,->,above] (5);
\end{tikzpicture}
\end{center}
```




```
\begin{center}
\begin{tikzpicture}
\tikzstyle{knode}=[circle,draw=black,thick,text
width = 12 pt,align=center,inner sep=1pt]
\node (G) at (30:1.5) [knode] {$G$};
\node (F) at (150:1.5) [knode] {$F$};
\node (H) at (270:1.5) [knode] {$H$};
\path (G) edge [above,in=20,out=70,loop,
every loop/.style={looseness=10}] node[swap] {$sn$} (G);
\path (F) edge [above,in=110,out=160,loop,
every loop/.style={looseness=10}] node[swap] {$rm$} (F);
\path (H) edge [below,in=295,out=245,loop,
every loop/.style={looseness=10}] node[swap] {$tp$} (H);
\draw (F) to node[midway,sloped,above] {$mn$}  (G)
(F) to node[midway,left] {$mp$}  (H)
(H) to node[midway,right] {$np$}  (G);
\end{tikzpicture}
\end{center}
```