A Guide to LaTeX and VIM for Mathematicians and Physicists

Nathaniel D. Hoffman

November 5, 2019

Contents

Preface

What is this guide?

As stated in the title, this is intended to be a starting point and a reference for using LaTeX and VIM, particularly in the fields of mathematics and physics. I do not claim to be an expert in any way, and many of the points in this come from other people who know how these things work far better than I do. The unanimity of their guidance throughout this document motivates me to cite them as a general reference at the end, since it is difficult at this point to tell which bit of knowledge I received from which source. Quite a lot of this document will just be a condensed explanation for something I struggled with and found on the LaTeX StackExchange site. At the end of this document, I will list multiple sources for further exploration, as well as good places to ask questions about both programs.

Who is this guide for?

This is intended to be a guide for people who are both new at LaTeX and VIM. However, because both programs are so extensive, much of this might apply to people who think they have intermediate/expert knowledge of one or both programs. As you will see (and as I continue to find out), it is very easy to believe you have a firm grasp of both while still learning all kinds of new tricks. The intent of this guide is to help get past each program's ridiculously steep learning curve. With this in mind, I don't recommend trying to convert completely to LaTeX and VIM as your only word processing software if you are, for example, in the middle of a semester in school. The time it takes to learn both programs will probably get in the way of doing work. A study from the University of Giessen[1] came to the following conclusion:

> We show that LaTeX users were slower than Word users, wrote less text in the same amount of time, and produced more typesetting, orthographical, grammatical, and formatting errors. On most measures, expert LaTeX users performed even worse than novice Word users. LaTeX users, however, more often report enjoying using their respective software. We conclude that even

[1] Knauff M, Nejasmic J (2014) An Efficiency Comparison of Document Preparation Systems Used in Academic Research and Development. PLoS ONE 9(12): e115069. https://doi.org/10.1371/journal.pone.0115069

> experienced LaTeX users may suffer a loss in productivity when LaTeX is used, relative to other document preparation systems.

It should also be noted that while this was published on December 19, 2014, a correction was made in March of 2015 with the following revision notes:

> This article was republished on March 30, 2015, to correct the sizing and placement of the figures; none of the article content was changed. The publisher apologizes for the original layout errors.

I will counter with three points. First, it is true that the goal of a writing software should be to produce a result with the least amount of errors possible, however, as Marie Kondo might say, LaTeX "sparks joy" more than other writing programs. Second, this study was very general. While WYSIWYG (What You See Is What You Get) software like Microsoft Word and Apple Pages have visual appeal when they are being used, they are not actually very versatile when you want to make big changes to a document without changing a lot of things. Additionally, in the specific fields of math and science, there are many reasons why one might not want to use a WYSIWYG word processor, most obviously because of equation support. Apple has recently added LaTeX equation support to its document suite, and I believe there are some similar features in Word, but it is currently difficult to make the formatting uniform, and equations are restricted to things in the default packages, so some of the most useful commands for scientists are not available. Finally, this study did not take VIM into account. While you can (and should) learn to write LaTeX in other editors, such as Overleaf, which allows for groups to edit documents simultaneously, learning and practicing VIM commands can make writing (especially writing equations) exponentially faster. I will try to convince you that VIM is one of the optimal environments to write LaTeX in, and it is versatile enough to be an IDE for many other coding environments, although they will not be covered in this guide. By proper document organization and some work at overcoming the learning curves for both programs, I believe LaTeX can become your favorite way to typeset your work.

I'll end this introduction with a fourth reason to use LaTeX: Other people already do, and it just looks good. The program is designed around proper typesetting—it is meant to be used in professional settings, and therefore has become a standard of professional typesetting. Many high-level academic journals use LaTeX, and some prefer that you submit papers which are already typeset, so all they have to do is import their own settings to maintain uniformity. VIM is often used as a text editor on remote servers, where you cannot easily open a program with a graphical interface (or none is installed). If these reasons have not convinced you, I will agree that LaTeX is probably not for everyone (VIM certainly is not). However, if you are in a scientific field, there is a rather high likelihood that you will have to use one or both of these programs at some time in your career.

Chapter 1

Getting Started with LaTeX

1.1 Installation

1.1.1 A note on software and hardware

I use a Mac computer. Sorry, I don't have the time or motivation to test out everything in this guide on a non-*nix system. Most of this should be platform independent, and I will try to make a note of the things that aren't. However, when it comes to installation, I will specifically refer to a general install on a Mac running the latest OS. There are online guides for installation on other systems which are easily available, and if this guide does not cover you, a quick Google search should be sufficient.

1.1.2 Downloading LaTeX

The current recommended way to install LaTeXon any system is from LaTeX Project. Scrolling down a bit, you should find a link to MacTeX, for Mac users: MacTeX. Download and install the latest version. This is exactly equivalent to an install from a package manager like HomeBrew or MacPorts. Because they are equivalent, I recommend you just download it straight from the main site, as you will be guaranteed to get a safe, up-to-date build.

Alternatively, if you don't want to install anything and you plan to skip any of the information about VIM, you can use Overleaf for free without any installation. If you wanted to ever download files from Overleaf and compile them on your local computer, you would need to first install LaTeX.

Now that you have LaTeX, you can write its version of the omnipresent "Hello World" script. For now, we will ignore VIM entirely. In fact, I would recommend you also don't use a LaTeX editor at all. The reason for this is that it will take additional time and frustration to make sure it all works correctly. Editors like Sublime Text 3 are nice, but they require you to install additional packages to process and interpret LaTeX. Additionally, these programs will provide you with autocompletion recommendations.

This might sound like a good thing to the LaTeXnewbie, but it will act as a crutch. The best way to be confident in LaTeXis to really understand what the commands mean so you can use them efficiently and without relying on an external program to make guesses for you. I will encourage you to start by writing raw code in a simple text editor, like TextEdit. Also, I recommend creating a new folder in which you can write this document. LaTeX creates a few auxiliary files which we don't need to worry about for now, but having them all in the same place will be helpful and will clutter up your own directories much less. Now would also be a good time to learn some Unix commands, as we will need to use them to compile the document anyway. On Mac, open up the Terminal application (just search for it, it's not hard to find). Once there, type `latex --version`. If you see anything that's not an error, you're probably okay. Check to make sure the version you have is the version you installed. If you get an error, it probably has to do with the way you installed the program, and unfortunately this is not a troubleshooting manual. That's what the internet is for. Now type `ls` (short for "list"). You should see some file names spit out. These are probably the files in your home directory. We can navigate to other directories by using the `cd` ("change directory") command. To get a feeling of how this command works, type `cd ` (with a space after the command), open finder, and drag and drop a directory from the finder window into Terminal. This will insert the "path" of that directory into Terminal in text form. Pressing return will cause Terminal to navigate to that directory. If any of the directories in that path have spaces in them, you must enclose the path in double quotation marks or replace spaces with a backslash-space combination (note that the backslash is the key right above return, \). `ls` will now show you the documents in that directory. Typing `cd ..` will move you to the parent directory, and `cd` by itself will always return you to the original home directory. Typically, your home directory is represented in the path as a tilde (`~`).

Navigate to a directory (like "Documents") and type `mkdir ` (short for "make directory", include the space) and the name of the directory you would like to create. For example, from a fresh Terminal,

```
cd ~\Documents
mkdir LatexTutorial
```

will navigate to your Documents folder and create a subfolder called "LatexTutorial." I will refer to this path in the future, so if you choose a different name or location, make sure you make the adjustment on your side. Sometimes I will just refer to a path as `<path/to/file.tex>` as a generic path to a specific file (`file.tex` here), in which case you should ignore the `<>` characters and insert whatever path gets you to that file (or drag and drop the file into Terminal).

Next, open up the TextEdit application and create a new document. By default, TextEdit will create a "Rich Text Format" (`.rtf`) file. This kind of file includes formatting like font size and type, text color, and other modifications which will screw with the LaTeX interpreter. Just like you wouldn't write Python code in Microsoft Word, you want to write LaTeX code in a plain text file with the extension `.tex`. Note that this is different from the usual plain text format `.txt`. To convert this RTF file to one that can be read by LaTeX, go to the `Format` menu and select `Make Plain Text` or press "shift-command-T". Type the following into this file, and then I'll explain what it means and how to compile it.

```
\documentclass{article}
\begin{document}
Hello World!
\end{document}
```

Save this in your newly minted folder as `helloworld.tex`. Now let's run it. In your MacTeX distribution, you should have `pdfTeX` installed. First, navigate to your tutorial directory. Typing `ls` should show you your `helloworld.tex` file. Now type `pdflatex helloworld.tex` and hit return. If everything is installed properly, there should be a new `helloworld.aux`, `helloworld.log`, and most importantly, `helloworld.pdf` file created in this folder. Type `open helloworld.pdf` to open your new PDF document in Preview (or whatever your default PDF viewer is set to).

Now let's go through some of the parts of this script. First, you'll notice that things that start with a backslash don't end up in the PDF. These are commands. Generally, commands will come in the form:

```
\commandname[options]{input}{input}\ldots
```

In general, commands take an input (or multiple inputs) and do something with them. Sometimes, additional options can be set in square brackets, with each option separated by a comma. Each input goes in its own set of curly brackets (braces).

Note

If you are familiar with programming languages, you might be wondering what the LaTeX policy on "whitespace" is. For the people who have no idea what that means, whitespace encapsulates all the characters that appear as blank spaces on the screen, like spaces and tabs. In LaTeX, the general rule is to minimize whitespace. If you were to put a space after the word "article" in your file, it would give an error (however, it would compile fine if you put a space before the word or before the first brace). Because of the ambiguity, I recommend you try to not have unnecessary spaces in your document. Trailing whitespace (extra characters at the ends of lines) is generally okay, but there are situations where having a blank line will give you errors (we'll see this when we talk about equations).

The first command here is the `\documentclass[options]{class}` command. The basic idea here is that it references a file labeled by the parameter `class` which contains a bunch of information on how LaTeX should set up the document's layout. This include stuff like the margins of the page, the distance between columns (if the text is in more than one column), the size of the header and footer, etc. For this document which you are reading, the document class is `book`, which allows me to create "chapter" structures. Generally, most documents for scientific journals and reports use the `article` class. Longer reports (like a thesis) would use the `report` class. You can also use `letter` to make neatly formatted letters (like letters in the mail), and `memoir` for making less structured documents. It is not important right now what these mean, just know that they exist (and there are more that I haven't mentioned). Sometimes, you will find

scientific journals that use their own custom class file, and for that you might have to download a `.cls` file and reference it in this command.

There are also some (optional) options for this command. There are quite a few, so I'll just go over some of the main ones which will be useful. First, you can set the main font size by adding in `10pt`, `11pt`, or `12pt`. If you don't put in any option, it will default to `10pt`.

You can add the option `titlepage` or `notitlepage` to choose whether or not to put the title information on a separate page at the beginning of the document. The default for an `article` is to not have a separate page, while `report`s and `book`s start with a title page.

You can also add the option `oneside` or `twoside` to change the formatting to work well with single or double-sided printing. This doesn't actually tell your printer that you can only print the thing single or double-sided, but it sometimes (depending on the document class) changes the layout to allow for more space on one side of the paper, which helps readability if that side was bound or stapled together.

Finally, the `draft` option draws boxes by problem areas in your document (things LaTeX thinks you've formatted incorrectly), and it doesn't render images but instead shows just an empty frame where the images would be placed. This can be helpful because documents with multiple images compile slower, so if you don't really care about looking at the pictures while you're working on the document, you can suppress them for quick compilations.

1.1.3 Environments

There are two special commands which you will see a lot when using LaTeX. They are the only other commands in our demo document: `\begin{environment}` and `\end{environment}`. The variable name "environment" tells LaTeX that the stuff between these commands follows an additional set of rules. For our demo file, the only environment is the `document` environment. The PDF output will only contain text that is placed inside this environment. Any text after `\end{document}` will not even be read by the compiler. Generally, every LaTeX file needs to have one (and only one) `document` environment.

Other environments which we will discuss later include math environments, like `equation` and `align` and list environments, like `itemize` and `enumerate`.

1.2 Packages

The first line of your document should always specify the document class. The last section should be the `document` environment. However, there can be commands placed between these, and one of the commands which we will use a lot is the one that brings in additional packages. If you are familiar with programming, packages should seem like second-nature. They are files created by someone else which add additional content to

LaTeX. There are a ton of packages out there for all kinds of specific uses. For example, there are packages which add lots of useful math and physics notation to equations. Some packages add color to text and equations, and some can even be used to draw figures and diagrams inside LaTeX. The basic command for adding a package to your file is

```
\usepackage[options]{package}
```

Not every package will have options, and it is not always necessary to include a space for them. The package name will go in the `package` input. Let's try one out now. Go back to your file and use the package `xcolor`. This package gives us access to a new set of commands which allow us to change the color of text and equations in our document. More specifically, it adds a command called `\color{label}` where `label` can be one of many built in colors. It is worth noting that after this command is invoked, the color will not change back to black again until the end of the current environment. Edit the document to look like the following lines and run it as we did before:

```
\documentclass[12pt]{article}
\usepackage{xcolor}
\begin{document}
\color{blue}Hello \color{red}World!\color{black}

More Text!
\end{document}
```

As you can see, we now have multicolored text in the document. Often, LaTeX will spit out errors about not recognizing a command name. 50% of the time, it's because you didn't import the correct package which contained the command. The other 49% happens because you misspelled the command. I reserve the other 1% for the other reasons which I have yet to experience and cannot anticipate.

Note:

You may have noticed that while there is a whole blank line in our document, there is not a blank line between the text in the PDF. Try deleting the blank line (still keeping the text on separate lines). You'll notice the text now appears on the same line in the PDF!
I'm not sure what the reasoning exactly is for having LaTeX interpret blank lines this way. I'd assume its to allow for readability in the source file, but you just need to know that if you want to start a new paragraph, you need a blank line. If you just want to break up a line at a specific point, you have a few options. The commands `\\` and `\newline` have almost the same exact meaning in most cases, and if you use one of them, the text on the following line will not be indented. Try this with the following text:

```
Hello World!\\ More Text!
Hello World!\newline More Text

Even More Text!
```

Notice the commands are followed by a space. This ensures that the following text is note interpreted as part of the command.

1.3 Top Matter

After you have imported all of the packages you want and set all of your document class options, we can move on to another important part of your document, namely the title page (or, as noted, a title which is not on a separate page, depending on whether or not you set that option). The commands used for this are fairly straightforward:

- `\title{The Title}`: This command sets the title of your document to be whatever you want it to be. Pretty straightforward.
- `\author{Your Name Here}`: This lets everyone know who wrote the document. Again, not a lot to remember.
- `\date{Today, this month, and the year}`: When you wrote the document. You can use `\date{\today}` to have it automatically use the current date (at time of compiling). Notice the `\today` command doesn't have an input. However, if you omit this command completely, LaTeX will insert the current date by default when you use the next command in the list:
- `\maketitle`: This command has no input, but takes all of the information from the previous three commands and formats it nicely in your document. There is some customization you can do, and I might touch on it in a later section, but for now we will use it as is.

The first three commands can be written in any order and can be placed before the `document` environment or inside it. However, `\maketitle` must be placed inside the `document` environment and must come after these commands when reading from top to bottom (otherwise it won't know what to put in the title). Try the following example:

```
\documentclass[12pt]{article}
\title{How to Hello World in Style}
\author{Insert Your Name Here}
\date{February 30, 2020}
\begin{document}
\maketitle
Hello World! Here's some text to fill the void!
\end{document}
```

Note that this is equivalent to

```
\documentclass[12pt]{article}
\begin{document}
\title{How to Hello World in Style}
\author{Insert Your Name Here}
\date{February 30, 2020}
\maketitle
Hello World! Here's some text to fill the void!
\end{document}
```

Now while I said you need to have all of your top matter information in the document before the `\maketitle` command, technically you will only be given a warning in the compiler if you put these commands out of order. Take some time to experiment with the order of these commands to figure out what happens when you misuse them. This will make it easy to fix problems in the future where the problem might not seem so obvious.

1.3.1 The Abstract

One environment that I believe falls under the category of top matter is an abstract. If you're used to reading or writing reports from professional fields or academia, you've probably seen nicely centered abstracts at the beginning of a paper. These can be easily created with an `abstract` environment inside of the `document` environment. Yes, environments can (and usually are) nested within each other. Just make sure you don't end an outer environment before you end all the environments inside of it, or you'll run into some nasty errors. For instance, the following text runs smoothly:

```
\documentclass[12pt]{article}
\title{The Title of the Paper}
\author{John Doe}
\date{June 4, 1989}
\begin{document}
\begin{abstract}
    Here's some text which should summarize the paper.
\end{abstract}
That about sums it all up.
\end{document}
```

However, this code will give you big, scary errors:

```
\documentclass[12pt]{article}
\title{The Title of the Paper}
\author{John Doe}
\date{June 4, 1989}
\begin{document}
\begin{abstract}
```

```
    Here's some text which should summarize the paper.
That about sums it all up.
\end{document}
\end{abstract}
```

1.4 Sections, Subsections, Subsubsections, Subsubsub...

You're probably wondering how I got these nice section headings with different sizes and styles. Although I used a more customized format file for this document (which I will provide and give an extensive tour of later), you can, right now, create similar headings in your file. The following commands are listed in order of largest to smallest in terms of heading size or importance. Note that none of these are available in the `letter` document class.

1. `\part{Part One}`
2. `\chapter{Chapter III}` (only in `book` and `report` document classes)
3. `\section{A Big Category of Information}`
4. `\subsection{A Slightly Smaller Category}`
5. `\subsubsection{An Even Smaller Header}`
6. `\paragraph{This is Just a Paragraph}`
7. `\subparagraph{This is not even a paragraph}`

These commands generally go on their own line inside the `document` environment, but it is important to note that they are not their own environment. Think of them instead as bookmarks which can be used to give a reader some sense of organization in a long paper. They don't necessarily have to come in any order—you can start your document with a subsection, for example. However, it is very important to note that everything except paragraphs and subparagraphs get either numbered, lettered, or have a Roman numeral prefix. You can turn off these prefixes by placing an asterisk (`*`, the object you get from shift-8 on most keyboards) between the end of the command and the brace that begins the input. For example:

```
\section*{This has no number}
\section{This does have a number}
```

You can also add a nice Table of Contents to the beginning of your document, and it will pick up the pages on which your headers are located and organize them neatly into a list at the beginning of your document. Before I show you an example of this, let's learn about a new (simple) package: `lipsum`. This package adds a command, `\lipsum`

which generates text from the Lorem ipsum, a standard placeholder text. This will allow us to see what our document would look like without having to fill in a bunch of space. The command also comes with an option, which represents the paragraphs of the text we want to print. Try the following example:

```
\documentclass[12pt]{article}
\usepackage{lipsum}
\begin{document}
\title{Lorem ipsum dolor sit amet!}
\author{Insert Your Name Here}
\date{January 10, 1938}
\maketitle
\tableofcontents
\begin{abstract}
\lipsum[1]
\end{abstract}

\part{In the Beginning}
\section{First Section}
\lipsum

\section*{Second Section}
\lipsum

\subsection{Subsection}
\subsubsection{Here's a small section}
\lipsum

\paragraph{Does this even count?}
\lipsum

\section{Third Section}
\lipsum

\end{document}
```

Important!

You will have to run `pdflatex helloworld.tex` **twice** to get this to work properly. Why? The first time, it creates a new file called `helloworld.toc` (for Table Of Contents) which contains the information about the table of contents. Once this file is created for the first time, you can compile as usual. Check what happens between each compilation. You'll see that the first one generates the proper document but the Table of Contents is blank!The second run will fill that table with the information from the file it created.

1.5 A Very Useful Command: TexDoc

This seems like a good place to end the very introductory chapter on LaTeX. You now can make a very simple document which doesn't do anything fancy, but works. However, there is still so much more to discuss. You're probably thinking, "this is nice, but I could already do all of this with my old word processor without having to learn all of these extra commands." You would be right, all of the things I mentioned are options available on most conventional word processors. This is the learning curve I discussed in the beginning; It will be difficult to see the benefits of writing in LaTeX until you are good at it. It probably won't be faster to compose a document in LaTeX. Even with VIM, most of your writing will depend on the speed of your typing. One of the key benefits of LaTeX early on is that you don't have to think about any of the formatting. You don't have to worry about choosing a font family or font size—instead, you can focus on the actual content of the document. Additionally, since we are using a very low-level text editor for the code, there is no color-coded highlighting. This will be introduced when I discuss VIM, and it will make your document much more readable. The other benefits will come in the form of special characters and symbols. You no longer need to search for μ, ε, or ö in your word processor. Likewise, LaTeX will give you simple commands for building complex mathematical formulae, powerful methods to build and reference a bibliography, and easy ways to modify the formatting on your entire document without changing any of its contents.

For an exercise, start playing with some of these commands and learn how they work. You can also try looking up environments like `itemize`, although most of the important environments will be discussed in later chapters. In the next chapter, we will focus on math contexts, environments which allow for special mathematical commands that can generate symbolic equations.

An important command to remember (a command I still frequently use) is `texdoc <package>`. This is a command-line command, and if you installed LaTeX properly, it should run fine in a fresh Terminal window. In place of `<package>`, enter the name of a package. This command will open the documentation file for that package. Try it out with `texdoc physics` to see the documentation for the `physics` package, which we will discuss frequently in later chapters. To learn more about all the packages which are available in LaTeX, visit the CTAN website. Your installation of LaTeX should include most packages on this site, and you should be able to access them just by using the `\usepackage{<package>}` command from above. The most important thing right now is that you learn the functions of these commands. Don't just copy paste the necessary code to the beginning of each new file you create. Take the time to type it out, make mistakes, and learn what you can and can't do. As a final exercise in this chapter, try putting text in quotes (single or double, it doesn't matter) and see what happens. See if you can "fix" the problem which arises from this seemingly inconspicuous problem. There's a reason why it happens and a way to fix it in the next chapter. When you learn VIM, there will be a way to not have to worry about it at all.

Chapter 2

Typesetting Equations

2.1 Equation Environments

The first thing you should know about typesetting equations in LaTeX is that they don't look like written equations. There are a multitude of commands which are used to create various mathematical symbols and position elements inside an equation. The other thing is that there are a few different kinds of math environments, just like the environments from the previous chapter, which can support these mathematical commands. Typing most of these commands outside of a math environment will give you errors, and there are some intricacies to even the most basic ones which can be confusing for new users. Let's start with the main distinction— there are inline equations and display equations. Inline equations fit inside lines of text, like so: $y = x^2 + \frac{1}{x}$. These environments are not like the ones I described previously. They aren't wrapped in `\begin` and `\end`, but rather within pairs of dollar signs ($). The equation above was written with the following code:

```
$ y = x^2 + \frac{1}{x} $
```

Another important thing is that, inside math environments, whitespace doesn't matter (except you still can't break up command names). For example, I could write

```
$y=   x^  2+\frac{ 1} {x } $
```

and still get the same result when compiled. However you decide to allocate space in your equations, I suggest you emphasize readability. It is already hard enough reading the equations in this new format without having to visually interpret spacing. There are also other ways of creating inline equation environments. By default, you can also use `\(...\)` or `\begin{math}...\end{math}` as the wrappers for inline math environments. I prefer to use dollar signs, but these will all give you the exact same result.

Next, we have display mode equations. These equations get their own line, like so:

$$y = e^{\imath\pi x}$$

Notice that this equation is also numbered. The numbering will be automatic, so if I was to put another equation before this one, it would receive a new number when compiled. You might be wondering how you can reliably refer to an equation by its number if it is subject to changes. This can be done by using labels, and we will discuss this in a later section.

The default environment wrappers for display math are `\begin{displaymath}...\end{displaymath}`, `\begin{equation}...\end{equation}`, and, for equations without numbers, `\[...\]` or `$$...$$`. Again, these options will have identical results, and I don't think there necessarily is a best choice. I have a preference, which we will get to when I discuss the default packages which I use.

2.2 Mathematics Packages

Before going into specific mathematics commands, we need to talk a few select mathematics packages which include symbols and formats which many mathematicians and physicists will use frequently.

2.2.1 AMSMath

The first of these is `amsmath`, a package developed by the American Mathematical Society (AMS). You can check out all of the features in this package with the Unix command `texdoc amsmath`. This package is light on symbols but provides additional environments which you will find very useful. If you have a blank line in a display math environment, you will receive an error. A similar error will occur if you try to break a particularly long equation into two lines by either putting a blank line between them or using things like `\\` or `\newline`. This package allows you to easily input multiline equations. First, it provides an environment called `multline` (notice the lack of the letter "i") which aligns the first line of the equation to the left, centers intermediate lines, and aligns the last equation to the right. Here is an example:

$$G^{\alpha\beta} = \left(g^{\alpha\gamma}g^{\beta\zeta} - \frac{1}{2}g^{\alpha\beta}g^{\gamma\zeta}\right)\left(\Gamma^{\epsilon}{}_{\gamma\zeta,\epsilon} - \Gamma^{\epsilon}{}_{\gamma\epsilon,\zeta}\right.$$
$$\frac{1}{4}g^{\epsilon\lambda}(g_{\epsilon\lambda,\sigma} + g_{\sigma\lambda,\epsilon} - g_{\epsilon\sigma,\lambda})g^{\sigma\lambda}(g_{\gamma\lambda,\zeta} + g_{\zeta\lambda,\gamma} - g_{\gamma\zeta,\lambda})$$
$$\left. - \frac{1}{4}g^{\epsilon\lambda}(g_{\zeta\lambda,\sigma} + g_{\sigma\lambda,\zeta} - g_{\zeta\sigma,\lambda})g^{\sigma\lambda}(g_{\epsilon\lambda,\gamma} + g_{\gamma\lambda,\epsilon} - g_{\epsilon\gamma,\lambda})\right)$$

The `gather` environment centers each line:

$$G^{\alpha\beta} = \left(g^{\alpha\gamma}g^{\beta\zeta} - \frac{1}{2}g^{\alpha\beta}g^{\gamma\zeta}\right)\left(\Gamma^{\epsilon}{}_{\gamma\zeta,\epsilon} - \Gamma^{\epsilon}{}_{\gamma\epsilon,\zeta}\right.$$
$$\frac{1}{4}g^{\epsilon\lambda}(g_{\epsilon\lambda,\sigma} + g_{\sigma\lambda,\epsilon} - g_{\epsilon\sigma,\lambda})g^{\sigma\lambda}(g_{\gamma\lambda,\zeta} + g_{\zeta\lambda,\gamma} - g_{\gamma\zeta,\lambda})$$
$$\left.-\frac{1}{4}g^{\epsilon\lambda}(g_{\zeta\lambda,\sigma} + g_{\sigma\lambda,\zeta} - g_{\zeta\sigma,\lambda})g^{\sigma\lambda}(g_{\epsilon\lambda,\gamma} + g_{\gamma\lambda,\epsilon} - g_{\epsilon\gamma,\lambda})\right)$$

Finally, the `align` environment allows for preferential alignment of equations using ampersands (`&`). Each instance of an ampersand will align in the same order in which they are placed. For example,

```
\begin{align}
    a + b &= c &x^2 = y\\
    d + e &= f &y = 4x
\end{align}
```

looks like

$$\begin{aligned} a + b &= c & x^2 &= y \\ d + e &= f & y &= 4x \end{aligned}$$

while

```
\begin{align}
    a + b &= c x^2 &= y\\
    d + e &= f &y = 4x
\end{align}
```

looks like

$$\begin{aligned} a + b &= cx^2 & &= y \\ d + e &= f & y &= 4x \end{aligned}$$

There are other variants of this, including the `split` environment, which we might discuss later. Additionally, adding a star (`*`) to the environment name (like `\begin{equation*}...\end{equat` will cause the equation to not be numbered. As mentioned, we will discuss a neater way to choose which equations are numbered in a later section.

2.2.2 AMSFonts and AMSSymb

Two more packages by the American Mathematical Society are `amsfonts` and `amssymb`. These add a ton of mathematical symbols, which can all be viewed on their `texdoc` page.

These packages work together; The `amsfonts` package adds the symbols as a new font while `amssymb` provides convenient commands for many of them, so I recommend including both of them in mathematical documents. Another thing `amsfonts` does is include a way to write "blackboard bold" letters, like $\mathbb{R}$, using the command `\mathbb{<letter>}`. Note that only capital letters work here. There is also a font for calligraphic letters, like $\mathcal{G}$, using `\mathcal{<letter>}`.

The mathrsfs Package

`mathrsfs` is another package which I have occasionally used which contains a font for script letters, like $\mathscr{F}$ (`\mathscr{<letter>}`).

2.3 Common Mathematics Environment Commands

2.3.1 Greek Letters

LaTeX comes with all the typical Greek letters, and all the commands for them follow the same form. Just add a backslash before the English spelling of the letter, and you will get the symbol for that letter. For example `\pi` gives π and `zeta` gives ζ. Changing the first letter from lowercase to capital will give the corresponding capital Greek letter. For example, `\gamma` gives γ while `\Gamma` gives Γ. Note that there is no capital version of the letter "chi" (`\chi` or χ).

2.3.2 Superscripts and Subscripts

There are a few default features of math environments which you will use all the time, and being accustomed to using them without having to look them up will greatly benefit your ability to write in LaTeX. The first of these features is superscripts and subscripts. You might have noticed in some of the above examples the use of the upward carrot `^` and underscore `_`, which are used to denote superscripts and subscripts respectively. Using one of these operators by itself, followed by a single character, will make that character the superscript or subscript of the previous. For example, `x^2` gives x^2. However, if you want more than one character to be included, you need to add braces: `x^{4a}` gives x^{4a}. If you were to just write `x^4a`, it would be interpreted as x^4a. However, this one character rule does not apply to commands, such as Greek letters. For example, `x^\gamma` becomes x^γ instead of $x\backslash gamma$. The behavior of the underscore is exactly the same. Note that you can have multiple levels of super and subscripts, but you need to include braces to specify the order in which they act. For example, $x^{3^2_4}$ can be produced with `x^{3^2_4}`, but `x^3^2_4` will give an error concerning a double superscript.

2.3.3 Fractions

As seen in previous examples, fractions can be created using the `\frac{<numerator>}{<denominator>}` command. This command takes two inputs—this is important!Without the second input, this command will throw an error. It is also important to make sure all of the braces are in the correct place, especially if you have things like fractions with fractional numerators and denominators. As an example,

```
\frac{\frac{x^2 + 2}{\epsilon} + 1}{4 - 2\epsilon_0^2}
```

will result in

$$\frac{\frac{x^2+2}{\epsilon} + 1}{4 - 2\epsilon_0^2}$$

2.3.4 Roots

The square root command is `\sqrt{<argument>}` and is used just like any other regular command. It automatically grows with its arguments, so if you put a fraction inside the square root, it will be scaled accordingly. For example, `\sqrt{\frac{4}{3}}` gives $\sqrt{\frac{4}{3}}$. What if you want a cubed root, or an nth root? You can add this as an option, just like we saw in the `\documentclass[]{}` command. `\sqrt[n]{e^{\pi x}}` will give you $\sqrt[n]{e^{\pi x}}$.

2.3.5 Non-Symbolic Math and Text

Mathematicians and physicists often use trigonometric functions and logarithms, as well as limits. There are commands for these in the AMS packages listed above. Why would we want a separate command for the cosine? Why not just type out `cos`? The answer is formatting. Text in math contexts is slightly italicized, which makes it difficult to distinguish things like trig functions and variables if the functions are just typed out. The commands for these functions are easy to remember; Most of them are just the word prefixed by a backslash, like `\cos`, `\tan`, and `\lim`. To see the difference between these commands and raw text, the command `\sin(x)` gives $\sin(x)$ while `sin(x)` looks like $sin(x)$.

You might now be tempted to type out `\ln`, `\tr`, or `\det`, but unfortunately these will give you errors. Later, I will show you the `physics` package, which adds commands for these, but for now, a useful command to type any text is the `\text{}` command. The usage is straightforward: `a^{\text{test}}` gives a^{test}.

2.3.6 Bracing and Parentheses

Try the following line of code:

```
(1 + \frac{1}{2})
```

You should find that it looks pretty bad:

$$(1 + \frac{1}{2})$$

It would be really nice if the parentheses grew to match the size of the fraction!There are two ways to do this in LaTeX. The first is automatically. We use the commands `\left` and `\right` to scale braces and parentheses relative to what they contain. Note that to create the symbol for braces, we have to use a backslash (since braces are used to signify inputs for commands by default, so we have to "escape" them to use them as the raw symbol). This can also be used for straight lines (like those used for an absolute value) For example,

```
\begin{gather}
    \left( 1 + \frac{1}{2} \right) \\
    \left[ 1 + \frac{1}{2} \right] \\
    \left\{ 1 + \frac{1}{2} \right\} \\
    \left| 1 + \frac{1}{2} \right|
\end{gather}
```

produces

$$\left(1 + \frac{1}{2} \right)$$
$$\left[1 + \frac{1}{2} \right]$$
$$\left\{ 1 + \frac{1}{2} \right\}$$
$$\left| 1 + \frac{1}{2} \right|$$

What if we want to have pair of parentheses to carry over a multiline equation? The `\left` and `\right` commands only work within the context of one line. If you are missing the corresponding opposite command in a line, you will get an error. For example,

```
\begin{gather}
    \left( 1 + 2 + \frac{1}{3}\\
    \frac{1}{4} + 5 + 6 \right)
\end{gather}
```

will generate an error. We can avoid this by using a period as a placeholder:

```
\begin{gather}
    \left( 1 + 2 + \frac{1}{3} \right. \\
    \left. \frac{1}{4} + 5 + 6 \right)
\end{gather}
```

produces

$$\left(1 + 2 + \frac{1}{3} \right. \\ \left. \frac{1}{4} + 5 + 6 \right)$$

The other method is to manually size bracing objects (this can also be applied to most other mathematical objects). You might wonder, if we have the ability to create automatic braces, why we would want to do this manually. The reason is because the method isn't perfect. For example,

```
\begin{gather}
    \left( 1 + 2 + \frac{1}{3} \right. \\
    \left. 4 + 5 + 6 \right)
\end{gather}
```

produces

$$\left(1 + 2 + \frac{1}{3} \right. \\ 4 + 5 + 6)$$

Notice the mismatching size of the parentheses? The second line has no reason to believe it should create a big right parenthesis, since there is no large object in that line. For this, we need to manually create a pair of parentheses. This can be done using a series of commands, which are listed below from smallest to largest:

- `\big`
- `\Big`
- `\bigg`
- `\Bigg`

We can fix the problem above with the following code:

```
\begin{gather}
    \bigg ( 1 + 2 + \frac{1}{3} \\
    4 + 5 + 6 \bigg )
\end{gather}
```

produces

$$\left(1 + 2 + \frac{1}{3} \right. \\ \left. 4 + 5 + 6 \right)$$

2.3.7 Memorizing Commands

With all of these commands for various symbols, it might seem that LaTeX is going to be a waste of time. After all, how can someone be expected to remember all of the commands for the various symbols they have to now learn? I'll let you in on a secret: I don't always remember the commands for symbols. Obviously I have a few that I keep on hand at all times, most of which I've mentioned above, but sometimes I'll run into a symbol I've never seen before written in a paper or used in a class. When this happens, you have two options. The first is to look through the package listings in some procedural manner, scanning through the documentation PDFs until you find a symbol matching the one you need. This could take lots of time, depending on the obscurity of the symbol in question. The second manner is to use an amazing program called Detexify. It was developed by Daniel Kirsch based on an idea by Philipp Kühl. Basically, you draw the symbol you are looking for and Detexify will search through a huge database to find the exact command you need. It's extremely useful, and there is now a Mac app which can run in the background and be accessed with a keyboard shortcut. This is a free software, but if you're a decent person, support the creator by buying a license (its like eight euros) or donate. I'm not being paid to say that, I just know that it's heavily used by the LaTeX community and the software has saved me a lot of time.

2.4 Physics Packages

2.4.1 Tensor Indices

While I should really start this section with the `physics` package, I want to quickly point out that there are a few different packages which can be used to write tensor indices. I know this could also go in the section on mathematics packages, deal with it.

If you search the CTAN website for packages to handle tensor indices, you'll find a few different options. My personal favorite is just the `tensor` package. Notice that without a package like this, writing `M^{ab}_{cd}^{ef}`, you will run into a double superscript error instead of what you probably want: $M^{ab}_{cd}{}^{ef}$. The `tensor` package gives two ways of adding indices onto tensors. The first is with the `\indices` and the second is with the `\tensor` command. The `\tensor` command allows for prefix indices, as seen below. You can read more about the specific uses using the command `texdoc tensor`.

```
M\indices{^a_b^c_{de}^{fg}}
```

$$M^{a\ c\ \ fg}_{\ b\ de}$$

```
\tensor[^a_{bc}]{M}{^d_e^{fg}}
```

$$^{a}_{bc}M^{d\ fg}_{\ e}$$

I should also mention here that the partial derivative symbol can be written using the `\partial` command: ∂.

2.4.2 The (Most Useful) Physics Package

I'm going to walk through the same information given by Sergio C. de la Barrera in his documentation for the `physics` package (which can be found using the `texdoc physics` command). This is because it includes a ton of useful commands which are applicable to other fields besides physics. I personally used to think this package was only useful for typesetting Dirac braket notation, but after reading the documentation, I found that there is a lot more included in it. Most of the documentation is explaining all the intricacies of the commands (such as how to disable automatic sizing on certain symbols), so I will condense it down into just the things that I find useful. Again, with all of the packages I mention, I encourage you to read the actual documentation.

Automatic Bracing

The package introduces another way to create pairs of braces like we had above. There really isn't that much of an advantage to doing this, as far as I can see, since you still can't use it properly across multiline equations. However, there are a few commands which are nice to have and make your LaTeX code more readable. The first two are `\abs` and `\norm`, which generate automatically sized lines and double lines around their inputs, respectively. For example, `\abs{x} - \norm{y}` produces $|x| - \|y\|$. This can generally save a lot of typing. Next, instead of writing `\bigg|_{x=0}` or some equivalent for evaluation limits, use the `\eval` command, which wraps its input with a vertical bar. `\eval{x}_{x=0}^{x=10}` produces a nice evaluation bar: $x\big|_{x=0}^{x=10}$.

Next up is order notation. We could use the symbol `\mathcal{O}` for the $\mathcal{O}$ symbol and follow it with some automatically sized parentheses, or we could use the command given by the `physics` package: `\order{x^2}` produces $\mathcal{O}(x^2)$.

Finally, I have to include the commands for commutators and anticommutators (Poisson brackets). While this could be done with automatically scaled square and curly brackets, the physics package provides short, easy to remember commands: `\comm{X}{P}` for $[X, P]$ and `\acomm{X}{P}` or `\pb{X}{P}` for $\{X, P\}$.

2.4.3 Vectors

There are a few vector commands which I believe are either included in the math packages or in default LaTeX. These include symbols like ∇ (`\nabla`), vector arrows like $\vec{x}$ (`\vec{x}`), the dot product symbol $A \cdot B$ (`A \cdot B`), and the cross product symbol $A \times B$ (`A \times B`). Additionally, unit vectors can be written with `\hat{x}` ($\hat{x}$). However, the `physics` package adds some useful features for writing better-looking vector math.

Let's start with the fact that bold letters are often used for vectors. While I'll show you another way to write bold math using a separate package, for single vectors, the `\vb` ("vector bold") command works well. We can also use `\va` ("vector arrow") and `\vu` ("vector unit") for arrowed and unit vectors, respectively. They appear as $\mathbf{x}, \vec{\mathbf{x}}, \hat{\mathbf{x}}$

as compared to the default $x, \vec{x}, \hat{x}$. These commands save you from having to "bold" your letters and then encapsulate them with the vector commands, which would give you $\boldsymbol{x}, \vec{\boldsymbol{x}}, \hat{\boldsymbol{x}}$. I'll show you how to make these bold italic letters later.

Next, the package introduces a matching set of nicely bold-ed inner and outer products. For a dot product, use `\vdot` instead of `\cdot` and for cross products, you can use `\cross` or `\cp` instead of `times`. You can see the difference here (I have the default version followed by the bold): $\cdot / \vdot$ and $\times / \cross$.

You're probably still wondering why these commands are at all necessary. The reason is not just because the bold characters look nicer, it's also to match with some simple shortcut commands provided by the package to do vector calculus. These commands are `\div`, `\grad`, `\curl`, and `\laplacian` for the respective operations. They can either take a single object as the input, or, instead of using braces, you can use parentheses or square brackets to enclose multiple objects. Below is a demonstration of these features:

```
\begin{equation}
    \div{\va{a}} + \curl(\va{a} + \va{b}) \vdot \grad{\Phi} \cross \laplacian{\psi}
\end{equation}
```

yields

$$\vec{\boldsymbol{\nabla}} \cdot \vec{\mathbf{a}} + \vec{\boldsymbol{\nabla}} \times \left(\vec{\mathbf{a}} + \vec{\mathbf{b}}\right) \cdot \vec{\boldsymbol{\nabla}} \Phi \times \nabla^2 \psi$$

Important!

To get arrows above the "del" operator, you need to import the `physics` package in a special way. You have to add the option `arrowdel` when importing it, so you will use `\usepackage[arrowdel]{physics}` instead of the typical import. Many packages have options like this to turn certain features on or off. I personally like this one because the operator is technically a vector, but if you don't like using arrows on your vectors and prefer just using bold letters, you might choose to load the package without this option.

2.4.4 Trigonometry

This is actually quite an overlooked feature of the package. Typically, parentheses after trig functions won't automatically scale with the argument. `\sin( \frac{\pi}{2} )` will just look like $\sin(\frac{\pi}{2})$. However, by default, the physics package causes the parentheses and square brackets following the command to scale, so the same command as before will give you $\sin\left(\frac{\pi}{2}\right)$. This makes it really convenient to write these functions without having a bunch of `\left` and `\right` commands. You can also add a power to the functions as an optional argument: `\sin[2](\theta)` results in $\sin^2(\theta)$.

2.4.5 Additional Functions

The same automatic bracing is added to the functions `\exp`, `\log`, `\ln`, `\det` and `\Pr`, which are just shorthand versions of displaying the text of the command. Additionally, the package adds a `\tr` and `\Tr` function (capitalization of the trace operation is preferential) as well as `\rank`, `\erf`, `\Res`, `\Re` and `\Im`. These last two will receive free curly braces when used, so `\Re{ \frac{z}{2} }` will give you $\mathrm{Re}\left\{\frac{z}{2}\right\}$.

2.4.6 Derivatives

Say you want to write a derivative or differential. Simply writing the text `dx` will give you dx, where both letters are in the same italic font. This is rather ambiguous—what if d was a variable? dd is not a clear way of denoting the differential; It looks like the letter was accidentally typed twice. While you could argue that the ambiguity is solved by context, we can eliminate it entirely by writing `\text{d}x` to get $\mathrm{d}x$. However, this is too much work, and it makes your code difficult to read. The `physics` package comes to the rescue again!It provides a command, `\dd` which creates automatic spacing and bracing for its arguments, and like the trig functions, it has an optional power argument. It seems like now is a good time to mention that integrals are just written with the `\int` command, so

```
\begin{equation}
    \int x \dd{x} = \int y \dd[3]{y} = \int \theta \dd(\cos(\theta))
\end{equation}
```

will give you

$$\int x \,\mathrm{d}x = \int y \,\mathrm{d}^3y = \int \theta \,\mathrm{d}(\cos(\theta))$$

Derivatives can be formed with a similar command: `\dv`. Of course, there are a few more options, since this now produces a fraction: `\dv{x}` $\rightarrow \frac{\mathrm{d}}{\mathrm{d}x}$. Additionally, you can specify a function in the numerator with `\dv{f}{x}`, and you can add a power in the same way as previous commands: `\dv[2]{f}{x}` $\rightarrow \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$. For partial derivatives, replace `\dv` with `\pdv`. An additional feature of partial derivatives is the ability to have multiple derivatives in the same fraction. For example, `\pdv{f}{x}{y}` automatically creates $\frac{\partial^2 f}{\partial x \partial y}$. This doesn't work if you add more arguments, but at that point you should really be using another notation anyway.

2.4.7 Dirac Braket Notation

The reason most people probably use this package is the commands for braket notation. Technically this notation can be achieved without any packages using `\langle` for $\langle$, `\mid` or `|` for $|$ and `\rangle` for $\rangle$, but these symbols are difficult to scale (if you have fractions inside the brakets, for instance). The `physics` package provides the `\bra` and `\ket` command for the corresponding sets of symbols. Placing these next to each other

with no space automatically contracts the pair into a single braket: $\langle\psi| \cdot |\phi\rangle = \langle\psi|\phi\rangle$ (`\bra{\psi} \vdot \ket{\phi} = \bra{\psi}\ket{\phi}`). There is also a command called `\braket` which takes two arguments, but I prefer its shorter alias `\ip` (short for "inner product"). For example, `\ip{a}{b}` gives $\langle a|b\rangle$ and `\ip{a}` gives $\langle a|a\rangle$. The commands `\ketbra`, `\dyad` and `\op` all give same (reversed, or outer product) result, and as you might have guessed, I prefer the shortest command.

Finally, you can use `\ev` for the expectation value of an operator, $\langle A\rangle$, and adding an additional command will give you the explicit form for a particular wave function: $\langle\Psi|A|\Psi\rangle$ (`\ev{A}{\Psi}`). For a generalized matrix element, use the `\mel` command, which takes three arguments and displays them in the same order: $\langle a|B|c\rangle$ (`\mel{a}{B}{c}`).