Some Useful LaTeX Knowledge

Tristan Lewis - tlewis@ugastro.berkeley.edu*

September 11, 2006

Foreword

The purpose of this paper is to help you understand some of the fundamentals behind editing a LaTeX document. One motivator behind this is that both in my experience of learning LaTeX on the fly as well as seeing others do the same thing, I realized that we rarely understood how and why the various commands that we commonly use worked. This led to the practice of memorizing each and every command that we ever wanted to use—a task that is inefficient as well as inflexible. I hope that by grasping a more general understanding of the way LaTeX operates, you will be able to avoid the daunting task of trying to memorize every command you will want to use in your documents and also be able to apply your generalized knowledge in order to figure out new tasks that you might want to tackle. The important thing to remember is that LaTeX really is a language; one that has its own vocabulary, punctuation, and syntax which for the most part are very consistent and systematic. Understanding this systematic behavior is one of the best ways to become a true LaTeX guru.

1 Starting Your Document

Every document in LaTeX needs to start off (and end with) several basic commands. There are also a few optional commands that can make your life easier by adding titles and dates to your paper. For example, the document you are now reading starts and ends with the following lines:

```
\documentclass[12pt]{article}
\title{Some Useful \LaTeX\, Knowledge}
\author{Tristan Lewis - tlewis@ugastro.berkeley.edu} \date{}
```

*Please e-mail me with any questions, comments, corrections, or suggestions for future versions of this document.

```
\begin{document}
\maketitle
[body of document here]

⋮

\end{document}
```

The `\documentclass` command sets up what kind of document this will be; in this case, the supplied argument `article` is provided within a set of curly brackets. There is also an optional parameter in square brackets to make the body text size 12pt. The next three commands, `\title`, `\date`, and `\author`, define the title, date, and author, which will go into making the title. Then there is the all-important `\begin{document}` call, which every document needs. This lets the compiler know that your document has actually begun, and it should be entered before any of the document's text makes its way into the file. The call to `\maketitle` will use the supplied title, date and author to create a large title for your paper on the first page. Finally, `\end{document}` finishes off the environment that was started all the way at the beginning with the `\begin{document}` and tells the compiler what should be obvious from the command: that the document has ended. Remember that any environment you enter with a `\begin{}` command must be closed off by a subsequent `\end{}` command.

2 Counters and Environments

There are a few LaTeX counters and environments that will be necessary to master in order to write lab reports which effectively communicate what you have learned and accomplished over the course of your lab assignment. Counters are things are counted and kept track for you by LaTeX , such as sections, subsections, and figures. Environments are specific modes of operation within a given portion of your document. What follows is an overview of the essential counters and environments you'll need to become familiar with.

2.1 Sections

Organization is key in any paper, and a scientific paper is no exception. Splitting your document into smaller, cohesive elements which focus on specific topics is beneficial to both the reader and the author. LaTeX makes it easy to to this with sections. To begin a section, you simply need to use the `\section{}` command, where the argument to be placed in the curly brackets is the name of the section. If you need further levels of detail within a single section, you can also make use of subsections,

which you call predictably enough with the `\subsection{}` command. Sections and subsections are numbered automatically by the compiler for you[1].

2.2 Math Mode

Let's admit it: easy and professional-looking typesetting of mathematical expressions is one of the biggest reasons that LaTeX is useful to people like mathematicians and astronomers. This great capability can't be taken advantage of in normal paragraph mode though—you must first enter something called *math mode*. The most basic way you can do this is by placing such expressions *in-line* in a normal paragraph. To do this, you need to simply surround the mathematical expression with a dollar sign, $, on either side. So, placing `$\beta$` in my otherwise normal sentence will yield a β, and so on. We can also place more complicated mathematical expressions in as well, such as: $\bar{x} = \frac{1}{N}\sum_i^N x_i$. However, this example illustrates that when adding some expressions in-line, especially ones containing fractions or integral signs and summations with specified limits, it comes out a little squished together and can even affect the line spacing of your document. To avoid this you can utilize the next flavor of entering math mode: math expressions.

Entering a math expression is just as easy as adding in-line math. The only difference is you need to surround your expression by double dollar signs, $$. Let's see what the math expression code looks like for $\bar{x}$ equation entered above:

```
$$
\bar{x} = \frac{1}{N}\sum_{i}^{N}x_{i}
$$
```

This new way of inserting math looks like this when compiled:

$$\bar{x} = \frac{1}{N}\sum_{i}^{N}x_{i}$$

The expression receives its own space on the page, which really allows it look as professional and elegant as it possibly can. Also, summation or integration limits are placed on the top and bottom of the summation or integral sign where they belong.

The final method involves the creation of a *formula*, and yields a result very similar to the previous method of math expression. The only difference in syntax is that you must surround the expression with `\begin{equation}` and `\end{equation}` instead of double dollar signs. The difference in output is that your equation is given an numerical identity in your document which you can later use to reference it. First, let's see the code:

[1] If you want to disable the automatic numbering for any counter, place an asteric directly after the counter's type when you declare it (eg: `\section*{Foreword}`).

```
\begin{equation} \label{meaneq}
  \bar{x} = \frac{1}{N}\sum_{i}^{N}x_{i}
\end{equation}
```

And now for the result:

$$\bar{x} = \frac{1}{N}\sum_{i}^{N} x_i \tag{1}$$

I included a new command, `\label{}` to give this equation a name that I can refer to later in the document with another command, `\ref{}`. Hence now that I've labeled this equation as `meaneq`, I can later type `\ref{meaneq}` in my document to render whatever equation number it happens to be (in this case, 1). The best part of this is that if I end up going back and putting another equation before this one, LaTeX will do all the work in renumbering the equations and update all instances of `meaneq` with its new equation number.

2.3 Figures

Figures are nice. Figures break up text, add eye-candy, and (hopefully) convey useful information that would be impossible to get across with the use of words alone. The best way to see how to add figures is simply to look at a sample of code, such as the following:

```
\documentclass[12pt]{article}
\usepackage{epsfig}

⋮

\begin{figure}[htb]
  \begin{center}
    \epsfig{angle=90, width=.65\textwidth, file=fig1.eps}
    \caption{This plot shows the line y=x} \label{linplot}
  \end{center}
\end{figure}
```

The first thing to take from this example is that we need to tell the compiler that we're going to be using a special package that allows the insertion of `.eps` files into our document. To do this, you need to use the `\usepackage{}` command after you declare your document class at the beginning of the file. For `.eps` files, you'll use the `epsfig` package.

Now let's move on to the placement of the image. First, we see that like other environments, this figure environment is being delimited with `\begin{figure}` and `\end{figure}`. The argument to `\begin{figure}` in square brackets has to do with the figure placement. LaTeX likes to have its own say as to exactly where your figures

get placed, but you can give it some priority by adding a list of locations in square brackets, where `h` means **here**, `t` means **top** of page, and `b` means **bottom** of page. Furthermore, you can stress your preference by adding an exclamation mark, `!`, in the first spot. So, if you really want LaTeX to put your graphic exactly where it appears in the `.tex` code, you can use `[!h]` and hope it listens to you.

Next, we begin a new environment, `center`, which will center everything within itself. Then comes the actual call to `\epsfig{}` which places the graphic in the output file. The only required argument is `file=`, but there are other optional ones such as `height, width,` and `angle`. The units for height and width must be specified—such as 5in or 10cm. Another useful unit you can use is `\textwidth`, which is the document's global variable that describes what width the text spans on paper. In my example above, I chose to make the graphic span 65% of the text width, so I used `width=.65\textwidth`. The angle units are degrees counter-clockwise from the image's original orientation[2].

Two more commands you can throw in before you end your figure environment are `\caption{}` and `\label{}`. Caption places a caption underneath the figure, and labeling the figure works just as it did with equations, where you can later refer back to the figure's label name (in this case, `linplot`), with the `\ref{}` command. The last two commands simply end the environments that were started here in the reverse order that they were started. Figure 1 shows how our figure came out.

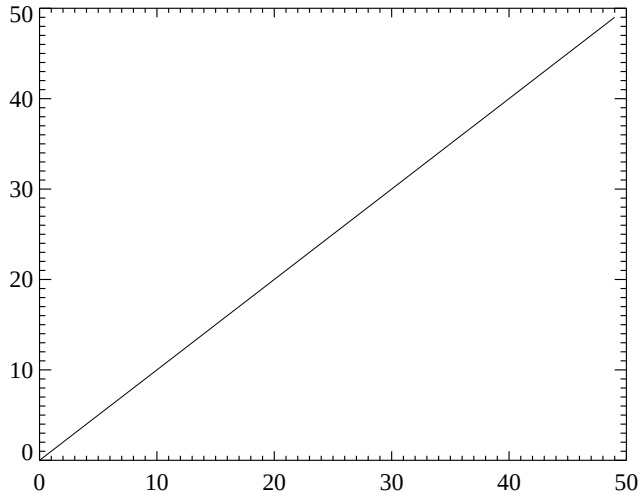

Figure 1: This plot shows the line y=x

[2]The order in which you specify the height, width, and angle transformations to your graphic is the same order that the compiler applies them. Thus, if you change the image's width and *then* rotate it, what you'll have changed is actually the height of the inserted image.

LaTeX also has the capability to work with normal post-script files (non-encapsulated). The first difference in this case is that instead of using the `epsfig` package, you will need to use one called `graphicx`. Then to insert an image, you will use the `\includegraphics{}` command. Let's look at what the code to insert a post-script file of the same parameters as the previous example would look like:

```
\documentclass[12pt]{article}
\usepackage{graphicx}

⋮

\begin{figure}[htb]
  \begin{center}
    \includegraphics[angle=90, width=.65\textwidth]{fig1.ps}
    \caption{This is also a plot of the line y=x} \label{linplot}
  \end{center}
\end{figure}
```

Using `\includegraphics{}` is slightly different than using `\epsfig{}` because now the only argument to go into the curly brackets is the filename of the post-script file. To specify optional argument that control the sizing and rotation of the file, put those arguments in square brackets right after the `includegraphics`. Note that you can easily use both `.eps` and `.ps` in a document by including both the `epsfig` and `graphicx` packages (`\usepackage{epsfig, graphicx}`).

2.4 Tables and Tabular

Tables can be an effective way to present data (especially numerical data) in a condensed, easy-to-read fashion. Let's say I wanted to compare the theoretical values for a quantity, $\Gamma(\alpha)$, with the measured values at different values of α ranging from 0-10. Here's what a table illustrating these values might look like:

α	Γ_{meas}	Γ_{theor}
0	3.4	3.2
2	5.6	5.3
4	10.5	10.8
6	24.6	22.5
8	30.1	28.9
10	37.8	34.0

First let's examine this table's code:

```
\begin{center}
  \begin{tabular}{r|c|c}
    $\alpha$ & $\Gamma_{meas}$ & $\Gamma_{theor}$ \\ \hline \hline
    0        & 3.4             & 3.2 \\
    2        & 5.6             & 5.3 \\
    4        & 10.5            & 10.8 \\
    6        & 24.6            & 22.5 \\
    8        & 30.1            & 28.9 \\
    10       & 37.8            & 34.0
  \end{tabular}
\end{center}
```

We're already familiar with the `center` environment so let's start at the second line, which introduces us to the `tabular` environment. The second set of curly brackets after the environment begin statement defines the format of the tabular environment's columns. Each letter represents a single column, and each vertical pipe (|) defines a vertical line drawn between the columns. The most common letters to choose from for defining columns are r, l, and c, which determine whether the column data will be **r**ight justified, **l**eft justified, or **c**entered.

After specifying how the columns behave, the table content can then be entered. We do this row-by-row, using the ampersand (&) to switch over to the next column for each row. In this way I specify my first row of values, α, Γ_{meas}, and Γ_{theor}, which act as my column headers. With that row's three columns specified, it's time to tell the compiler that I want to move down to the next row, which is accomplished with double backslashes, `\\`. For this table, I've chosen to add two horizontal lines with `\hline` after the column headings. Then I proceed with the subsequent rows' entries, making sure to include a `\\` after each one until I reach the end. At this point all that's left to do is end the `tabular` and `center` environments that I began above. An important note is that the way I decided to space out the columns in my code has no effect on how the table is generated—I chose to line up the column dividers simply for readability in the `.tex` file.

Tables are usually fairly important components of lab reports, and for this reason you'll likely want to have your table numbered and captioned just like your figures and equations. To do this, the only thing you need to do is place your `tabular` environment within a `table` environment like the following[3]:

```
\begin{table}[htb]
  \begin{center}
    \begin{tabular}{c | c | c} [table entries]
```

⋮

[3]Remember that like the Equation and Figure environments, you should include a placement specifier when declaring any Table environments.

```
\end{tabular}
\end{center}
\caption{This is my table caption.} \label{examptab}
\end{table}
```

The `tabular` environment is very flexible, and isn't limited to being used just with the `table` environment. An example of this is using `tabular` to place multiple graphics together within a single figure. Here's how you might do this to create a figure with 2 rows and 2 columns of graphics:

```
\begin{figure}[!htb]
\begin{center}
\begin{tabular}{c c}
\includegraphics[angle=90, width=.5\textwidth, height=.4\textwidth]{fig1.ps} &
\includegraphics[angle=90, width=.5\textwidth, height=.4\textwidth]{fig1.ps}\\
(a) & (b) \\
\includegraphics[angle=90, width=.5\textwidth, height=.4\textwidth]{fig1.ps} &
\includegraphics[angle=90, width=.5\textwidth, height=.4\textwidth]{fig1.ps}\\
(c) & (d)
\end{tabular}
\end{center}
\caption{(a)---A plot of y=x; (b)---Another plot of y=x; (c)---Yet another
    plot of y=x; (d)---The last plot of y=x, I swear} \label{quadplot}
\end{figure}
```

While this table code is much more complicated than cases where you are just entering numeric data, it should turn out as expected as long as you keep track of your column and row divisions. Figure 2 shows how the example four-graphic figure above is rendered.

3 Font Families, Series, and Shapes

LaTeX contains three main font families, which are analogous to the different fonts that are accessible from your run-of-the-mill word processor (i.e. Word). These are Roman, Sans Serif, and `Typewriter`, with Roman being the default. To apply these families to text, you can use the `\textrm{}`, `\textsf{}`, and `\texttt{}` commands, respectively, with the target text appearing between the curly brackets. While in a lab report you will probably not have any use for the two other fonts, it is useful to know how to manually set the font to Roman in cases where you are in math mode. For example, if I want to typeset a formula that provides the number of angular degrees in 2π radians (and include the word radians in the equation), I could code it as following:

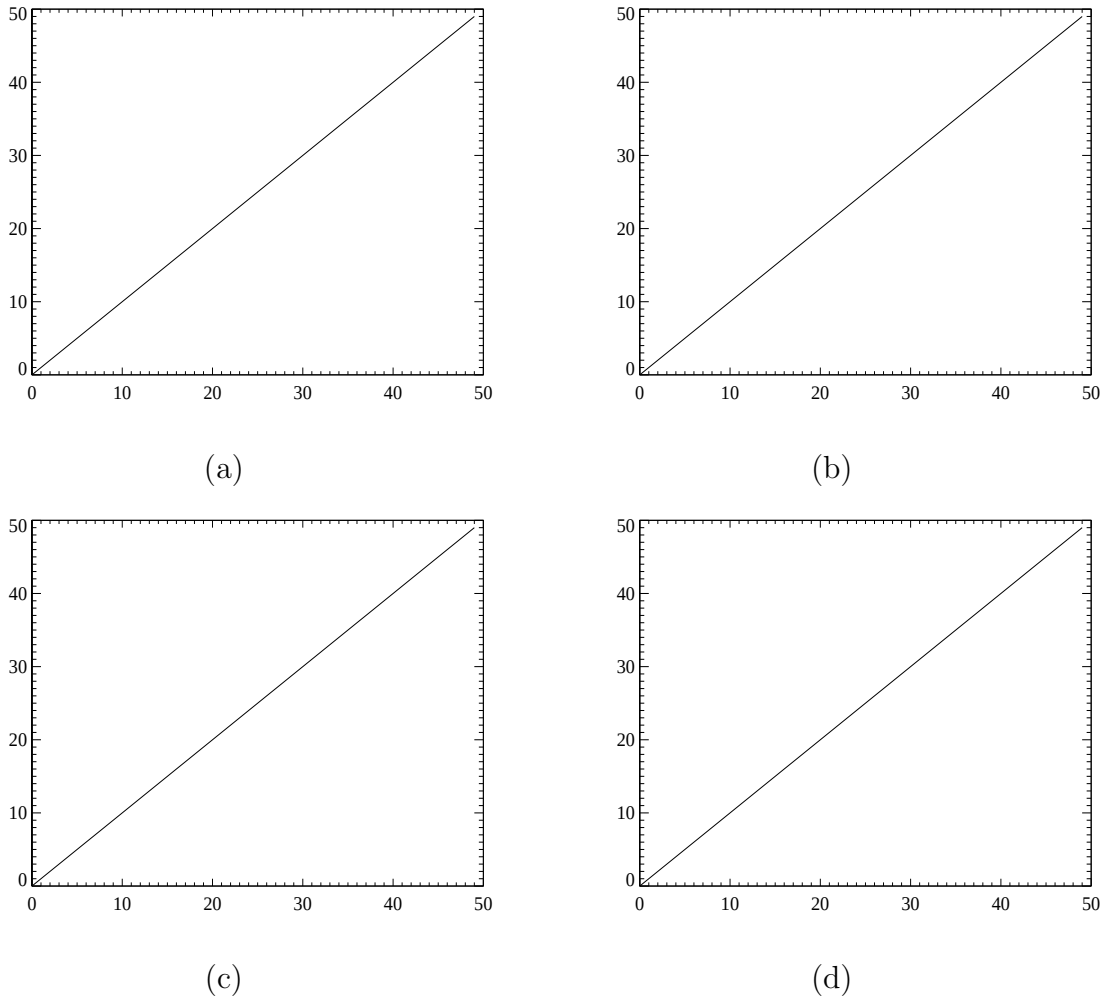

Figure 2: (a)—A plot of y=x; (b)—Another plot of y=x; (c)—Yet another plot of y=x; (d)—The last plot of y=x, I swear.

```
\begin{equation}
  360^{\circ} = 2\pi radians
\end{equation}
```

and get this out from the compiler:

$$360^\circ = 2\pi radians \tag{2}$$

We quickly notice that while in the equation environment, *every* character is considered a mathematical expression, making normal words appear in the strange curly math font and ignoring spaces completely. We can get around this by explicitly

telling LaTeX to make the word radians (along with a proceding space, since spaces are ignored in math mode) to be typeset in the Roman font family like this:

```
\begin{equation}
  360^{\circ} = 2\pi \textrm{ radians}
\end{equation}
```

to achieve a much better result:

$$360^\circ = 2\pi \text{ radians} \tag{3}$$

Also present in LaTeXare font series and shapes. Essentially, these allow you to make your text **bold-faced**, *emphasized*, *italicized*, or *slanted*. These can be useful if you wish to add the occassional stylistic touch to your words. The `\textbf{}`, `\emph{}`, `\textit{}`, and `\texts{}` commands, respectively, let you use these features.

4 Special Symbols

There are oodles are special symbols available for use in LaTeX—so many that it would be futile for me to try to outline them all here. Luckily, there are some good sources on the internet for you to reference when you're just aching to stick that proportionality sign or real numbers symbol into one of your equations. An extensive list can be found at `http://omega.albany.edu:8008/Symbols.html` or just Google "latex symbols" for all the information you'd ever need.