Disclaimer

I am not an expert in \LaTeX. The advice given here is based on nothing more than experience. It is likely that there are better ways to do some of these things but the techniques here are what works for me.

All packages are cited so that you can look up the documentation yourself and find what works best for you. The bibliography always references the package on the CTAN website [1]. There you will find the package itself (if you need to download it) as well as package documentation (usually) provided by the author. I have checked that all of the packages I’ve referenced are well-documented at the time of writing this document.

The code for this document is available upon request.
## Contents

1 Basic \LaTeX

1.1 Paragraph Behavior .......................................................... 3
1.2 Horizontal Spacing ............................................................ 3
1.3 Multi-Line Equations .......................................................... 4
1.4 Units ................................................................................. 5
1.5 Custom Commands ............................................................... 6
1.6 Modular Documents .............................................................. 8

2 Quizzes and Exams ................................................................. 9

2.1 Margins, Headers, and Footers ............................................... 9
2.2 Question Numbering ............................................................. 10
2.3 Spacing ............................................................................... 12
2.4 Answer Keys ........................................................................ 13

3 Handouts .............................................................................. 14

3.1 Answer Keys (Again) ............................................................. 14
3.2 Clickables ............................................................................ 15
3.3 Equation Numbering and Referencing .................................... 15
3.4 Common Preamble ................................................................. 17

4 Images and the Like ............................................................... 18

4.1 Importing Images ................................................................. 18
4.2 Tikz .................................................................................... 19
4.3 Graphing Functions .............................................................. 25
4.4 Advanced Tikz Examples ...................................................... 28
4.5 Figures ............................................................................... 30

5 Final Thoughts ....................................................................... 31
1 Basic LaTeX

This guide assumes that the user is compiling documents using the `pdflatex` command. This is the default in most editors and has become a common practice. If you are still compiling documents with the `latex` command I urge you to try `pdflatex`.

At several points throughout this document we will refer to external files. Whenever a compiled document uses other files (perhaps using the `\include` command or the `\includegraphics` command) it will be assumed that those files are in the same directory as the file that refers to them. It is easy to define relative paths to documents that aren’t in the same directory, but we won’t get into that here.

1.1 Paragraph Behavior

By default, LaTeX will place an indent in front of each paragraph and no space between paragraphs. This can be inconvenient for a syllabus or for an exam. The quick fix for this is to load the `parskip` package [11]. Putting the following line in the preamble will remove the indent and put a small amount of space between paragraphs.

```latex
% Putting this line in the preamble changes the behavior of paragraphs.
\usepackage{parskip}
```

Use of the `parskip` package is nothing more than a quick fix. The proper way to do this is to change the `\parindent` and `\parskip` variables (ideally by using or defining a new document class).

1.2 Horizontal Spacing

There are two situations where it is important to be mindful of spacing. The first occurs after a period. By default, LaTeX assumes that every period is at the end of a sentence and it provides a small amount of extra space. To fix this, simply put the `\` character after the period when the period is not at the end of a sentence. It is subtle, but there is a difference in the result of the following two lines of code:

```latex
% This is wrong.
\Mr. Michael Price is the world’s best boss.

% This is right.
\Mr.\ Michael Price is the world’s best boss.
```

Another problem in spacing occurs after the use of a command. To typeset “LaTeX” I use the `\LaTeX` command. The compiler needs a space after a command to tell it that the command is over. Hence the compiler always assumes that the space after a command is used to end the command and that it should never typeset a space directly after the command. To tell the compiler that it should typeset a space after a command, put a `\` after it. The difference is shown below:

```latex
% There will be no space after the command.
\LaTeX is the best.
```
There are also commands for custom horizontal spacing. This can be especially useful when typesetting equations. Some only work in math mode and some work in both text and math mode. They are as follows:

- \! inserts a negative thin space in math mode.
- \, inserts a (positive) thin space in text mode or math mode.
- \> and \: insert a medium space in math mode.
- \; inserts a thick space in math mode.
- \enspace inserts a space of .5em in text mode or math mode.
- \quad inserts a space of 1em in text mode or math mode.
- \qquad inserts a space of 2em in text mode or math mode.
- \hspace{<length>} inserts a space of <length> in text mode or math mode.
- \hfill fills the available horizontal space.

The following example demonstrates how small amounts of horizontal space can influence the appearance of an equation:

### Code:

```latex
\begin{align*}
% No horizontal spacing.
% Horizontal spacing adjustment.
\end{align*}
```

### Result:

\[
g(a, b) = \int_0^a \int_0^b G(x, y) \,dy \,dx
\]

\[
g(a, b) = \int_0^a \!\!\int_0^b G(x, y) \,dy \,dx
\]

Note: if you think that I am typesetting my differentials wrong then you are not alone. This seems to be a regional preference and there is a lot of debate about it in the \LaTeX community.

### 1.3 Multi-Line Equations

The `align` environment allows for convenient typesetting of equations that require more than one line. However, the `align` environment will give each of those lines an equation number. Using the `align*` environment instead will remove those numbers, but sometimes it is nice to give the entire grouping one common equation number.
This cannot be accomplished (to my knowledge) with the align environment. Instead, use the split environment within the equation environment. An example is shown below:

```
\begin{align}
f(x) &= (x -2)(x +1) \label{eq:1a} \\
&= x^2 -x -2 \label{eq:1b}
\end{align}
```

7 Using the split environment within equation.
\begin{equation}
\begin{split}
f(x) &= (x -2)(x +1) \\
&= x^2 -x -2
\end{split}
\end{equation}

Note that the syntax is exactly the same in the split environment as it is in the align environment (that is, & and \ work the same) except that each line gets its own label in the align environment while the split environment only needs one label for the entire equation which is ... kind of the point.

## 1.4 Units

Typesetting units can be a tricky business, especially for new \LaTeX users. However, the siunitx package [36] really helps. The \si command will typeset units on their own while the \SI command will typeset a number along with the appropriate units. The example below demonstrates some of the functionality of the package:

```
\begin{itemize}
\item \SI{\kilo\gram\meter\per\second}
\item \SI[per-mode=symbol]{\kilo\gram\meter\per\second}
\item \SI{200}{\centi\meter}
\item \SI{2.4e-9}{\gram}
\item \SI{49588394394}{\mole}
\item \SI{30}{\%}
\item \ang{30}
\item \num{3.02e23}
\end{itemize}
```
Result:
- kg m s\(^{-1}\)
- kg m/s
- 200 cm
- \(2.4 \times 10^{-9}\) g
- 49 588 394.394 mol
- $30
- 30 \%
- 30\(^\circ\)
- \(3.02 \times 10^{23}\)

See the \texttt{siunitx} documentation to alter the style of the output. Keep in mind that the \texttt{siunitx} package only knows S.I. units by default. However, it is easy to add imperial units with the \texttt{\DeclareSIUnit} package as shown below.

\textbf{Code:}
% These lines need to be in the preamble.
\usepackage{siunitx}
\DeclareSIUnit\inch{in}
\DeclareSIUnit\foot{ft}

% This goes in the body of the document.
A yard stick contains \SI{36}{\inch}, or \SI{3}{\foot}.

\textbf{Result:}
A yard stick contains 36 in, or 3 ft.

Note that the \texttt{SIunits} package [35] is depreciated and should not be used.

1.5 Custom Commands

It is incredibly useful to be able to define custom commands. There are (unfortunately) several ways to do this; most of the time I find that the \texttt{\newcommand} command does the job. For example, the following line creates a command called \texttt{\myCommand} which simply prints the text “Hello, world!”

\textbf{Code:}
% Put this in the preamble to create the command \myCommand.
\newcommand{\myCommand}{Hello, world!}

Once this line is in the preamble, it can be implemented as follows:

\textbf{Code:}
My custom command prints \myCommand
Custom commands become particularly useful when they are created to take arguments. The general form of the `\newcommand` command looks like `\newcommand{<name>}[<number of arguments>]{<output>}`. If there are three arguments, then to use them when defining the output you simply type `#1`, `#2`, and `#3`. An example is shown below:

**Code:**

```latex
% Put this in the preamble to create the command \myVec .
\newcommand{\myVec}[3]{#1 \, \vec{i} + #2 \, \vec{j} + #3 \, \vec{k}}

% Put this in the body of the document to implement the command .
Vectors in $\mathbb{R}^3$ have the form $\myVec{x}{y}{z}$.
```

**Result:**

Vectors in $\mathbb{R}^3$ have the form $x \vec{i} + y \vec{j} + z \vec{k}$.

Note that the `\myVec` command was written assuming that it would be used in math mode. If it were invoked outside of math mode then there would be an error (the `\vec` command does not work outside of math mode). It can be valuable to write a command which forces the compiler to use math mode even if it is invoked outside of math mode. If you want to do this, look up the `\ensuremath` command.

There may be a time when you want to overwrite a command that already exists. For example, maybe you don’t like the way that \LaTeX renders the `\vec` command by default and you want to overwrite it with your own vector notation. If you tried `\newcommand{\vec}[1]{<output>}` then the compiler would throw an error because you’re trying to define a command that already exists. In this case, simply replace \newcommand with `\renewcommand` and the compiler will understand what you’re doing. Of course, use `\renewcommand` with caution; there are many commands that should never be overwritten.

The other commands (that I’ve used before) to define things are `\def`, `\let`, `\providecommand`, and `\DeclareRobustCommand`. The `\def` and `\let` commands are similar (though subtly different) and they are more primitive than `\newcommand` (the syntax for these two commands is slightly different than that of `\newcommand`). The `\providecommand` command works just like `\newcommand` except that the `\providecommand{\aCommand}{<output>}` defines the command `\aCommand` according to the definition unless the command already exists, in which case the old definition is preserved. The `\DeclareRobustCommand` command is a more stable version of `\newcommand`. Some commands are fragile and may fail in certain environments. If you find that a custom command is causing an error and you can’t see a reason why that should be, try replacing `\newcommand` with `\DeclareRobustCommand`. It is generally considered bad practice to use `\DeclareRobustCommand` in all cases “just to be safe.” In my entire lifetime I’ve only ever come up with one custom command that required `\DeclareRobustCommand`. It is shown below: (Note that this is a poorly written command that I do not use any more.)

```
% This command throws and error sometimes when defined with \newcommand .
\DeclareRobustCommand{\autorefeq}[1]{\hyperref[#1]{Equation \ref{#1}}}
```

It is also possible to create custom environments in \LaTeX as well as custom commands. This is done with the (you guessed it) `\newenvironment` command. Custom environments are a little trickier and, personally, I use them a lot less. As a reminder, the difference between a command and an environment is shown below:

```
% Commands are called as follows :  
\someCommand{...}
```
% Environments are called as follows:
\begin{someEnvironment}
...
\end{someEnvironment}

Advanced users (whatever that means) should look up the xkeyval package [2]. This is a lovely package for creating commands with more functionality (like key/value options) which has excellent documentation.

Custom commands need to come with a warning. They are useful, but they should be implemented with some reservations. Overusing them can cause confusion and frustration. The internet (or at least the parts of the internet that I frequent) is full of arguments about when it is appropriate to use a custom command. It is also my understanding that some journals will not accept article submissions that use custom commands.

1.6 Modular Documents

Most \LaTeX{} projects are going to consist of a single .tex document and the files that result from compiling it. Sometimes, however, it is convenient for the content of the project to be spread over multiple files. In this case there will still be one “master” .tex file which will be compiled and several other .tex files which contain content that will be retrieved by the master file. This can be done using the \input and \include commands.

The \input command looks for a single argument (the name of a file without extension) and it then looks for a .tex file by that name. It then simply places the content from that .tex file into the file that called the \input command. For example, suppose that you have a folder called “Project” which contains five files: master.tex, introduction.tex, chapter1.tex, chapter2.tex, and chapter3.tex. The contents of master.tex might look something like the following:

\begin{verbatim}
% Below is everything in master.tex:
\documentclass{article}
\begin{document}
% The \input command is used to pull content from other files into this file. Make sure that each of the corresponding .tex files (for example, introduction.tex) are in the same directory as master.tex.
\input{introduction}
\input{chapter1}
\input{chapter2}
\input{chapter3}
\end{document}
\end{verbatim}

Note that \input{introduction} does not contain the file extension. The \input command assumes that its arguments are .tex files. When master.tex is compiled the result will be exactly the same as if the \input{...} command were replaced by the contents of the file that it calls. Simply put, \input is analogous to “copy-and-paste.” Note that, in this case, no file other than master.tex should contain \documentclass, \begin{document}, or \end{document} commands.

The second way to accomplish a similar goal is with the \include command. There are important differences between \input and \include. The difference in the output is that \include will put \clearpage commands (essentially) before and after the content in the file. Also, \input commands can be nested (that is, you can use an \input command inside of a file that will also be called by an \input command) whereas \include commands cannot be nested. However, \input commands can be nested inside of \include commands.

The primary feature of \include is that each individual file is given its own .aux file which, coupled with the \includeonly command, means that you can have a document which contains references to labels that are not in the content of the document, but the document can still understand those references. This is useful if, for example, you want to write lecture notes but you don’t want to present all of the notes to your students at once. You can have references in chapter 1 which refer to labels in chapter 3, but when only chapter 1 is compiled those references will still work. (Of course, this confuses the hyperref package.)
2 Quizzes and Exams

One way to typeset exams is through the `exam` document class [26]. This document class provides custom formatting, imports several packages, and contains custom commands that allow its users to create nice exams. I do not like to use this class. I prefer to build exams from scratch (usually using the `article` class). This section contains some of the techniques that I use to do that.

2.1 Margins, Headers, and Footers

By default, \LaTeX uses extremely wide margins which are not ideal for exams. The variables that are used for the spacing around the edge can be modified by hand but this is not advisable. There are a lot of them, they interact with each other in complicated ways, and it is difficult to manipulate them to get the margins that you want. A popular way to achieve smaller, uniform margins is to use the `fullpage` package [25] as below:

```latex
% This is depreciated. Do not do this.
\usepackage{fullpage}
```

However, the `fullpage` package is depreciated and should not be used. Instead, use the `geometry` package [16] with the `margin` option as follows:

```latex
% Put this in the preamble to get margins of the desired size.
\usepackage[margin=1in]{geometry}
```

The `geometry` package has several other useful features for advanced users.

The headers and footers also interact with the margins. A convenient and popular package for these things is the `fancyhdr` package [27] which is the package that was used to make the headers and footers in this document. Use the following commands to load the `fancyhdr` package:

```latex
% These can appear in the preamble or in body of the document.
% We first tell the document to use headers and footers.
\pagestyle{fancy}
% These three lines control the contents of the header.
\head{\small \LaTeX in the Classroom}
\chead{}
\rhead{}
% These three lines control the contents of the footer.
\lfoot{\small \textit{Tuesday, January 28, 2014}}
\cfoot{}
\rfoot{\small Page \thepage\ of \pageref{LastPage}}
```

Once the package has been loaded you can actually control the header and footer. The code below was used to create the headers and footers for this document:
The \texttt{\textbackslash\head} command controls the text on the left side of the header, the \texttt{\chead} command controls the text in the middle of the header, and the \texttt{\rhead} command controls the text on the right side of the header. The footer commands are similar. The command \texttt{\pageref{LastPage}} requires the use of the \texttt{lastpage} package [15].

It is often convenient to change or remove the header at different points in the document. For example, this document has no header and footer on the first page. To change the contents of the header, simply change the \texttt{\xhead} and \texttt{\xfoot} commands at any point; those changes will start on the page where those commands occur. You can also remove the headers and footers with another call to \texttt{\pagestyle}. The line below removes all headers and footers from that point forward:

\begin{quote}
\begin{verbatim}
1 \% Headers and footers will be removed unless another call to \pagestyle is made.
2 \pagestyle{empty}
\end{verbatim}
\end{quote}

The \texttt{\pagestyle} command can be used at any point in the document to change or remove the headers and footers.

If you are using a header and footer, you need to tell the \texttt{geometry} package so that it can adjust the margins accordingly. This is done with the \texttt{includeheadfoot} option as follows:

\begin{quote}
\begin{verbatim}
1 \% Make sure to use the includeheadfoot option if you use headers and footers.
2 \usepackage[includeheadfoot,margin=1in]{geometry}
\end{verbatim}
\end{quote}

The margins may still interfere with the header sometimes (though it is unusual). If the header is overlapping the first line(s) on each page, try adjusting the \texttt{\headsep} parameter. In the line below, adjust 10pt until you get the desired spacing:

\begin{quote}
\begin{verbatim}
1 \% This line in the preamble changes the distance between the header and the text.
2 \setlength{\headsep}{10pt}
\end{verbatim}
\end{quote}

\subsection{Question Numbering}

There are fancy ways to number questions, but the easiest is with the \texttt{enumerate} environment. A simple example is shown below.

\begin{quote}
\begin{verbatim}
\begin{enumerate}
  \item (2pt) Define a solvable group.
  \item (3pt) Prove that all finite $p$-groups are solvable.
\end{enumerate}
\end{verbatim}
\end{quote}

\begin{verbatim}
\begin{enumerate}
\item (2pt) Define a solvable group.
\item (3pt) Prove that all finite $p$-groups are solvable.
\end{enumerate}
\end{verbatim}

Every use of an \texttt{enumerate} environment restarts the counter. However, this can be avoided with the \texttt{enumitem} package [18]. After loading the package, simply replace \texttt{\begin{enumerate}} with \texttt{\begin{enumerate}[resume]} to make the numbering pick up where it last left off. An example is shown below:

\begin{quote}
\begin{verbatim}
\begin{enumerate}[resume]
\item Make sure this line is put in the preamble.
\end{verbatim}
\end{quote}
This list will start from 1, like normal.
\begin{enumerate}
\item (2 pt) Define a solvable group.
\item (3 pt) Prove that all finite $p$-groups are solvable.
\end{enumerate}

Fill in the blank:
\begin{enumerate}[resume]
\item (1 pt) There are exactly \rule{-1pt}{2em}\rule{0.5pt}{0.5pt} groups of order six.
\item (1 pt) If there is only one group of order $n$ then $n$ is divisible by at most \rule{-1pt}{2em}\rule{0.5pt}{0.5pt} primes.
\end{enumerate}

Result:
1. (2pt) Define a solvable group.
2. (3pt) Prove that all finite $p$-groups are solvable.

Fill in the blank:
3. (1pt) There are exactly ____ groups of order six.
4. (1pt) If there is only one group of order $n$ then $n$ is divisible by at most ____ primes.

It is also useful to be able to change the labeling of the items. That is, instead of using numbers it might be preferable to use letters or roman numerals. This is also done with the \texttt{enumitem} package.

\begin{code}
% Make sure that the \texttt{enumitem} package is loaded in the preamble.
% Labels will look like (III).
\begin{enumerate}[label=(\Roman*)]
\item (2 pt) Define a solvable group.
\item (3 pt) Prove that all finite $p$-groups are solvable.
\end{enumerate}

Fill in the blank:
\begin{enumerate}[resume]
\item (1 pt) There are exactly \rule{-1pt}{2em}\rule{0.5pt}{0.5pt} groups of order six.
\item (1 pt) If there is only one group of order $n$ then $n$ is divisible by at most \rule{-1pt}{2em}\rule{0.5pt}{0.5pt} primes.
\end{enumerate}

Result:
(I) (2pt) Define a solvable group.
(II) (3pt) Prove that all finite $p$-groups are solvable.
Fill in the blank:

a: (1pt) There are exactly ___ groups of order six.

b: (1pt) If there is only one group of order n then n is divisible by at most ___ primes.

The most useful choices for counters are \texttt{\arabic}, \texttt{\Arabic}, \texttt{\alph}, \texttt{\Alph}, \texttt{\roman}, and \texttt{\Roman}. The meanings should be self-explanatory. The \texttt{enumitem} package has many useful features that provide additional functionality to the \texttt{enumerate} environment.

This is not the only way to change the display of the counters. In fact, it is neither the simplest nor the best. The \texttt{enumerate} package \cite{4} provides an easier way of accomplishing the same but it does not interact well with the \texttt{enumitem} package. (The \texttt{enumerate} package is something of a watered-down version of \texttt{enumitem}.) The most complete (and most advanced) way to change the \texttt{enumerate} environment is to redefine its variables directly. This can be done locally or globally and allows full control. However, I find that the \texttt{enumitem} packages works in most cases.

Note that \texttt{enumerate} environments can be nested so that problems can have different parts. An example is shown below:

\begin{Code}
\begin{enumerate}
\item Let \( f(x)=x^2 -1 \).\end{enumerate}
\begin{enumerate}
\item (2pt) Find all \( x \) such that \( f(x)=-1 \).
\item (2pt) Find all \( x \) such that \( f(x)=1 \).
\end{enumerate}
\item (1pt) If \( f(x)=x^3 -6x^2+2x-1 \) find \( f(3) \).
\end{enumerate}
\end{Code}

\begin{Result}
1. Let \( f(x) = x^2 - 1 \).
   \begin{enumerate}
   \item (2pt) Find all \( x \) such that \( f(x) = -1 \).
   \item (2pt) Find all \( x \) such that \( f(x) = 1 \).
   \end{enumerate}
2. (1pt) If \( f(x) = x^3 - 6x^2 + 2x - 1 \) find \( f(3) \).
\end{Result}

\subsection*{2.3 Spacing}

It is likely that you will want to add space between questions for students to work. This is done with vertical spacing commands.

\texttt{\clearpage}: This command (with no arguments) ends the current page and pushes all of the content after it onto the next page. There is also a \texttt{\newpage} command; the difference is in how they handle floats. Most of the time \texttt{\clearpage} and \texttt{\newpage} do the same thing. I usually use \texttt{\clearpage} unless I have a problem.

\texttt{\vspace}: This command adds a specified amount of vertical spacing up until a page break. That is, \texttt{\vspace(2in)} leaves 2 inches of vertical spacing between the command before and the command after unless there is not 2 inches of space left on the page, in which case it acts like \texttt{\clearpage}. The \texttt{\vspace*} command does the same except it runs across page breaks.

\texttt{\vfill}: This command (with no arguments) takes all of the vertical spacing that would normally appear at the end of the page and puts it at the specified location. If two \texttt{\vfill} commands are used on the same page then the space is distributed evenly between those two commands. Note that vertical spacing is allocated to \texttt{\vspace} commands before it is distributed to \texttt{\vfill} commands.

An example is going to be illustrative. Consider the code below:
The two \clearpage commands ensure that these three questions all land on the same page and nothing else will be on that page. The \vspace{2in} command places 2 inches of vertical spacing after the first question. The two \vfill commands then make sure that there is the same amount of space after the second question and the third question. Note that it is perfectly acceptable to put the \clearpage command within the enumerate environment.

At this point, I do not know of a way to place a certain percentage of the remaining vertical spacing in a particular place. That is, if there are two questions on a page you might want to give one of them 40% of the vertical space and the other one 60% of the vertical space. I do not know a way to accomplish that effect. I would use a single \vspace command and a single \vfill command and play with the argument of the \vspace until it looked approximately correct.

### 2.4 Answer Keys

After students take a quiz I always want to give them a key. At first I would copy Quiz.tex, rename it Quiz-key.tex, and then change all of the vertical spacing commands to answers. This is not a good system. It is terribly inconvenient to have so much identical content in two different files. Every time one of them is changed the other needs to be changed and it can be hard to remember to do that. I have since developed solutions to this problem.

One option is to put the following custom command into your preamble:

```latex
% This is the command that allows switching between quiz and key.
% Use #1 for a quiz and #2 for a key.
\newcommand{\switchme}[2]{#1}
```

Then, when typesetting your quiz, put all vertical spacing commands in the first argument of \switchme and all answers in the second argument. All \clearpage commands should be written as \switchme{\clearpage}{}. The implementation is shown below:

```latex
\switchme{\clearpage}{}
\begin{enumerate}
\item What is $8 + 14$?\switchme{\vspace{2in}}{Answer: $8+14=22$}
\item If $f(x) = 2x - 7$, find $f(5)$.\switchme{\vfill}{Answer: $f(5) = 3$}
\item Simplify $\sqrt{50}$ as much as possible.\switchme{\vfill}{Answer: $5\sqrt{2}$}
\end{enumerate}
```

The utility of the \switchme command is that the way it is defined above it only prints the first argument (the vertical spacing) while ignoring the second. However, by changing the #1 in the definition to #2 the command only prints the second argument
(the answer) and ignores the first argument. When the document is compiled with a value of #1 in the definition of `\switchme` you get a quiz but by simply changing the value of #1 to #2 and recompiling you get a key.

3 Handouts

Typesetting handouts - like syllabi or homework assignments - is often different than typesetting quizzes or exams.

3.1 Answer Keys (Again)

Section 2.4 explored one way to make a single LaTeX file that works as both a quiz and an answer key. In a handout, there may be times when you do not care about adding vertical spacing. For example, suppose you’re writing a homework assignment for your students. In this case, there is another way to make a single document which works as both an assignment and a key.

The idea here is similar. We will define a command called `\answer` which does different things depending on its definition. First, the body of the homework assignment should look something like this:

```
\begin{enumerate}
  \item What is $\sqrt{4}$?
  \answer{$2$}
  \item How many feet are in a mile?
  \answer{5280}
  \item What is the capital of Maine?
  \answer{Augusta}
\end{enumerate}
```

Then place the following two commands in the preamble:

```
% Leave this line uncommented to get an assignment.
\newcommand{\answer}[1]{}

% Leave this line uncommented to get a key.
\newcommand{\answer}[1]{\par \fbox{Answer: } #1}
```

Leaving the first uncommented tells the compiler to ignore everything inside of an `\answer` command. Leaving the second uncommented tells the compiler to render the answer with some formatting (of your choosing).

A convenient usage for this trick is when there is one common set of questions that are used across many different documents. For example, my lecture notes contain homework exercises at the end of each section. These exercises show up in four different places: in the notes, in my “instructor version” of the notes, in handouts that I give to the students (containing only the exercises), and in the answer keys that I post after homework is due. As mentioned before, it is terribly inconvenient to have the same code in four different places as any small change needs to be implemented four times.

The solution that I have adapted is to keep the questions in their own separate file. For example, there is a file called `HWquestions3_4.tex` whose contents are below:

```
% The complete contents of HWquestions3_4.tex (as you will notice, my class is very easy).
\item What is $\sqrt{4}$?
\answer{$2$}
\item How many feet are in a mile?
\answer{5280}
\item What is the capital of Maine?
\answer{Augusta}
```
Note that this content is not wrapped in an `enumerate` environment. This is so that I can have control over the `enumerate` environment in each individual document. Then each of the four documents which contain this content have code similar to the following:

```latex
\begin{enumerate}
% This line imports the questions.
\input{HWquestions3_4}
\end{enumerate}
```

Remember that `\input` is appropriate here because it is unlikely that I want the `\clearpage` commands that `\include` puts before and after things. This implementation allows me to put different definitions of the `\answer` commands in each document separately depending on whether I want the answers included and how I want them to be formatted.

### 3.2 Clickables

Of course, \LaTeX\ can handle references without any additional setup. Use of the `\label{key}` and `\ref{key}` (and some of its cousins) are built in by default. However, with more and more documents being accessed on a screen (rather than being printed), it is convenient if those references can be clickable.

This functionality is provided by the `hyperref` package \cite{hyperref}. Simply loading the `hyperref` package in the preamble will make all links clickable, but there is a lot of other customization available. This is a very large and versatile package that I leave you to explore, but below is the (important part of the) implementation of the `hyperref` package in preamble of this document:

```latex
% First the package is imported.
\usepackage{hyperref}
% The \hypersetup command customizes the package.
\hypersetup{colorlinks=true, linktocpage=true, pdfhighlight=/O, urlcolor=myBrown, linkcolor=myBlue, citecolor=myGreen}
```

The code above won’t compile in another document unless the colors `myBrown`, `myBlue`, and `myGreen` are defined. There are, of course, default colors but the `xcolor` package \cite{xcolor} allows users to define their own custom colors. This is another package with a lot of options, but for now I leave you with the way the colors were defined in this document:

```latex
% First import the package.
\usepackage{xcolor}
% Then define the colors.
\definecolor{myGreen}{rgb}{0,.6,0}
\definecolor{myBrown}{rgb}{.6,0,0}
\definecolor{myBlue}{rgb}{.255,.41,.884}
```

### 3.3 Equation Numbering and Referencing

By default, displayed equations (those that are not in-line) which are formatted using the `equation` environment are numbered while those formatted with the `equation*` environment and with `\[\ldots\]` will not be numbered. (There is also a `\nonumber` command that you can use in the `equation` environment and the like.)

This behavior is inconvenient. Sometimes mathematicians prefer to label only those equations that will be referenced and no others. The `mathtools` package \cite{mathtools} allows you to use the `equation` environment (or `align` and the like) in every case and then only number those equations which have references. This is accomplished with the following two lines in your preamble:
Now you can typeset as normal with only two changes. First, use the equation environment for all equations. Second, do not use the \ref command for equations any more. Instead, use the \eqref command. An example is shown below:

\begin{verbatim}
\texttt{Code:}
\begin{verbatim}
\begin{equation}
\left<\alpha, \beta\right> = \frac{1}{|G|} \sum_{g \in G} \alpha(g) \beta(g).
\end{equation}
\end{verbatim}
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\texttt{Result:}
The inner product of class functions $\alpha$ and $\beta$ is defined as
\begin{equation}
\left<\alpha, \beta\right> = \frac{1}{|G|} \sum_{g \in G} \alpha(g) \beta(g).
\end{equation}
\end{verbatim}

It can be shown that if $\chi$ and $\lambda$ are irreducible characters then
\begin{equation}
\left<\chi, \lambda\right> = \begin{cases} 1 & \text{if } \chi = \lambda \\
0 & \text{if } \chi \neq \lambda
\end{cases}.
\end{equation}
\end{verbatim}

The relationship in (4) is called \textit{row orthogonality}.

This allows you to label every single equation, reference them as you please, and let the numbering take care of itself.

Another useful command for referencing things is the \autoref command (which comes with the hyperref package \cite{hyperref}). This command tells \LaTeX{} to format the word “Equation” with the hyperlink in addition to the equation number. This is made clear in the following example:

\begin{verbatim}
\texttt{Code:}
\begin{verbatim}
% Only the equation number is formatted.
The relationship in \eqref{eq:orth} is called \textit{row orthogonality}.
%
% The word "Equation" is typeset along with the equation number.
The relationship in \autoref{eq:orth} is called \textit{row orthogonality}.
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\texttt{Result:}
The relationship in (4) is called \textit{row orthogonality}.
The relationship in Equation 4 is called \textit{row orthogonality}.
\end{verbatim}

The \autoref command works wonderfully if the mathtools package is not used. However, one needs to be careful when using the \autoref command on equations along with the mathtools package. In my experience, things seem to work fine if
the \texttt{mathtools} package is loaded (and the \texttt{\mathtoolsset} command is called) before the \texttt{hyperref} package.

The \texttt{\autoref} command also works with other references, as well. In fact, the utility of the \texttt{\autoref} command comes from the fact that the command itself will figure out the kind of thing that you’re referencing so that you don’t have to worry about it. References to sections with the \texttt{\autoref} command come out looking like “Section 3.3.” More complicated usage will require some tweaking, however. For example, if one creates a theorem environment (using the \texttt{amsthm} package \cite{31}) then references to theorems with the \texttt{\autoref} command will not work properly unless one provides a command called \texttt{\thetheorem\autorefname}. See the \texttt{hyperref} documentation for more on this.

There are two other packages that are worth mentioning here. The \texttt{autonum} package \cite{14} provides a functionality similar to that of the \texttt{mathtools} package and the \texttt{cleveref} package \cite{9} provides commands that work like \texttt{\autoref}. When I was learning how to make these things work I found that \texttt{mathtools} and \texttt{\autoref} worked best for me; you might decide differently.

3.4 Common Preamble

It is often the case that several documents are meant to have identical formatting. Perhaps all quizzes should look the same or all homework assignments should look the same. This can be done with custom packages or class files, but I’ve started using a common preamble to accomplish this.

For example, suppose that we’re trying to create twenty homework assignments with the same formatting. It is likely that all of the formatting and package implementation for these documents will happen in the preamble and the preamble will be nearly identical for each document. I find it convenient, then, to put the preamble in its own file called (for example) \texttt{hw\_preamble.tex}. An example of such a file might look like this:

```latex
\begin{verbatim}
\% This is the entire contents of hw_preamble.tex:
\%
\% Add common packages.
\usepackage{amsmath}
\%
\% geometry creates uniform margins.
\usepackage[includeheadfoot,margin=0.5in]{geometry}
\%
\% Add any necessary custom commands.
\newcommand{\answer}[1]{% the default behavior of \answer
\%
\% It is likely that headers and footers will have common elements.
\usepackage{fancyhdr}
\pagestyle{fancy}
\renewcommand{\headrulewidth}{0.5pt}
\renewcommand{\footrulewidth}{0.5pt}
\lhead{\small Homework Assignment}
\chead{ }
\rhead{ }
\lfoot{\small{\textbf{Last Updated: \today}}}
\cfoot{ }
\rfoot{\small{\textbf{Page \thepage\ of \pageref{LastPage}}}}
\end{verbatim}
```

Once the common preamble is in one file, each of the twenty homework assignments will look the same except for the content. For example, the homework for Section 10.4 might be in a file called \texttt{hw10\_4.tex} which looks like:

```latex
\begin{verbatim}
\documentclass{article}
\input{hw_preamble}
\rhead{\small Section 10.4} % We put the section number in the header
\%\renewcommand{\answer}[1]{%par\fbox{Answer: } #1} % uncomment this line to make a key.
\begin{document}
\%  
\end{verbatim}
```
An \input command imports the contents of hw_preamble.tex into the preamble and the \rhead command makes a small tweak in header to reflect that the assignment is for Section 10.4. Here we have also modified the \answer trick from Section 3.1 to fit the situation. In hw_preamble.tex there is a definition of \answer which ignores any input. In order to make a key we simply re-define the \answer command in the actual document (remember to use \renewcommand here since \answer has already been defined) to format and print the answer.

4 Images and the Like

Images are terribly important for teachers. You want to be able to use \LaTeX{} to ask the homework and quiz questions that you want to ask instead of formulating the questions that you can ask around the \LaTeX{} that you know. Providing a diagram with a question is essential from time to time.

There are two ways to use images in a document. You can create images using an outside source and then include them into your document or you can use various packages to have \LaTeX{} create an image for you. We will cover both methods.

4.1 Importing Images

The graphicx package \cite{5} allows you to include images that were made from an external source. Once you’ve imported the package in your preamble, the \includegraphics command allows you to include images. Suppose we have a file called cat.jpg. It can be displayed as follows:

```
\begin{center}
\includegraphics{cat}
\end{center}
```

The \includegraphics command places the image right where you tell it without any changes. There are some optional arguments that help control the size of the image, rotate the image, or clip the image. Examples are shown below:
If you compile your document with `pdflatex` then you can use any of these file types: `.jpg`, `.png`, `.pdf`, or `.eps`. It is advisable to use images with vector graphics (`.pdf` or `.eps`) rather than raster graphics (`.jpg` or `.png`).

### 4.2 Tikz

There are several ways to create various images within \LaTeX. For example, there are ways to use both Gnuplot [33] and Python [13] within a `.tex` document. Here we are going to describe how to use the `tikz` package to create images. You will notice that there is not a bibliography entry for `tikz`; this is because it is a part of a larger package called `pgf` [30].

The `pgf` package is an extraordinary and powerful package with a wide range of functionality. The documentation [29] alone is over 700 pages. Here we will give an introduction to the things of which the `tikz` package is capable. A good reference for beginners is *A very minimal introduction to TikZ* by Crémer [8] which is provided in the CTAN documentation for the `pgf` package [30].

The first step is to load the `tikz` package. For more complicated uses of `tikz` there are other steps required in the preamble, but for now simply load the `tikz` package as normal:

```latex
\usepackage{tikz}
```

For now, all drawings in `tikz` will happen in the `tikzpicture` environment. There are other options, but personally I use the `tikzpicture` environment almost exclusively - even for advanced uses. This environment is not centered by default; be sure to use the `center` environment if it should be centered.

```latex
\begin{tikzpicture}
\draw (0,0) -- (2,1);
\end{tikzpicture}
```

A `tikzpicture` environment is based on *nodes*. Everything happens within a Cartesian plane centered about some origin. The choice of origin is merely a reference. For example, if the picture is centered then the whole image will be centered; it will not center around the origin. Nodes are nothing more than points in this plane. When defining them you can use units if you want. For example, the node `(1in,6em)` will give a node which is 1in horizontally from the origin and 6em vertically from the origin. However, I find it more useful to leave out units and use the default scaling. As a reference, the squares in the grid below have size of length one “unit” according to `tikz`.

```latex
% Make sure the tikz package is loaded in the preamble.
\begin{tikzpicture}
% This is where the magic happens.
\end{tikzpicture}
```

We will start with some of the basic commands which make shapes. Here are some of these commands:

- \texttt{\draw (0,0) -- (2,1);}
This command draws a line between the nodes (0,0) and (2,1).

- `\draw (0,0) rectangle (2,1);`

  This command draws a rectangle with corners at (0,0) and (2,1).

- `\draw (0,0) circle(1.5);`

  This command draws a circle of radius 1.5 centered at (0,0).

- `\draw (0,0) grid (2,1);`

  This command draws a grid of squares with sides length 1 in the rectangle with corners at (0,0) and (2,1).

- `\draw (1,0) arc [radius=2, start angle=90, end angle=225];`

  This command draws an arc of a circle starting at (1,0) with the specified attributes.

- `\draw (0,0) to [out=90,in=195] (3,2);`

  This command draws a curved line which goes from (0,0) to (3,2). It leaves (0,0) at an angle of 90° and enters (3,2) at an angle of 195°. I imagine this is an interpolating polynomial subject to tangency conditions but I do not know for sure.

Some examples are shown below. Note that most of your document cares (at least somewhat) about whitespace. However, inside of the `tikzpicture` environment, whitespace - and line breaks in particular - are ignored. Lines are ended with the `;` character.

```
\begin{tikzpicture}
\draw (0,0) grid (13,4);
\draw (1,1) -- (2,3);
\draw (2.5,1.5) rectangle (3.5,3.5);
\draw (5.5,2) circle (1.5);
\draw (9.5,1.5) arc [radius=1, start angle=0, end angle=135];
\draw (10.5,1) to [out=90,in=270] (12,3);
\end{tikzpicture}
```

The `scale` optional argument can be used to shrink or stretch the entire picture. The code below is identical to the previous example except for the addition of `[scale=0.3]`. I believe that this is a linear scaling; that is, in this case height and width are both 30% of their original size. Note that when text is used (as we will demonstrate later), the text size does not change under scaling.

```
\begin{tikzpicture}[scale=0.3]
\draw (0,0) grid (13,4);
```

The same command used to draw a line can be used to draw connected line segments. If those connected line segments form a closed curve then you can use the `fill` argument to fill the curve with a color of your choosing, as shown below:

```
\begin{tikzpicture}
\draw (0,0) -- (2,0) -- (1,2) -- (0,1) -- (-1,2) -- (-1,0);
\draw [fill=gray] (3,0) -- (5,0) -- (6,2) -- (3,0);
\end{tikzpicture}
```

The example below shows how to add thickness to lines, how to use different line styles, and how to add arrowheads to either side of the line. Note that the optional argument of the form `<key1>-<key2>` describes the arrowheads. Make sure that `key1` describes the arrowhead at the beginning of the line (not at the left) and `key2` describes the arrowhead at the end of the line.

```
\begin{center}
\begin{tikzpicture}
\draw [dashdotted,<->] (0,4) -- (10,4);
\draw [dashdotdotted, help lines, latex - latex] (0,3.5) -- (10,3.5);
\draw [dashed, ultra thick, to-to] (0,3) -- (10,3);
\draw [densely dashed, very thick, ->-] (0,2.5) -- (10,2.5);
\draw [loosely dashed, thick, <->] (0,2) -- (10,2);
\draw [dotted, semithick, |<->|] (0,1.5) -- (10,1.5);
\draw [densely dotted, thin, |<-|] (0,1) -- (10,1);
\draw [loosely dotted, very thin, -<] (0,0.5) -- (10,0.5);
\draw [ultra thin, <->] (0,0) -- (10,0);
\end{tikzpicture}
\end{center}
```
The colors of lines can also be changed. A `\draw[red]` command will draw a red line.

The `node` command allows for labeling things and printing text, as shown below:

```
\begin{center}
\begin{tikzpicture}
\draw[thick,<->] (6.3,0) -- (0,0) -- (0,2.3);
\draw[help lines] (0,0) grid (6,2);
\node at (1,1) {text};
\node[above] at (3,1) {N};
\node[right] at (3,1) {E};
\node[below] at (3,1) {S};
\node[left] at (3,1) {W};
\node[above right] at (5,1) {NE};
\node[below right] at (5,1) {SE};
\node[below left] at (5,1) {SW};
\node[above left] at (5,1) {NW};
\end{tikzpicture}
\end{center}
```

Result:

All nodes to this point have been defined in Cartesian coordinates using the notation \((<x_{\text{pos}}>,<y_{\text{pos}}>)\). An alternate notation, which is very convenient, is to define them in polar coordinates using \((<\text{angle}>,<\text{radius}>)\). Anything in the \texttt{tikz} package assumes that angles are in degrees. Nodes can be stored if they are to be reused. In the example below, nodes \texttt{a}, \texttt{b}, and \texttt{c} are stored and used later.

```
\begin{center}
\begin{tikzpicture}
\node (a) at (0,0) {a};
\node (b) at (3,0) {b};
\node (c) at (1,2) {c};
\draw (a)--(b)--(c)--(a);
\end{tikzpicture}
\end{center}
```
Note the behavior of the labels on the nodes. This code places circular nodes with text labels and lines between it are told not to overlap the text. In fact, they point from the center of one node to the center of the other node and stop just short of the node itself. This is ideal behavior when typesetting a commutative diagram, but unfortunate behavior when making a diagram of a triangle. The solution lies in the \texttt{coordinate} command as shown below:

\begin{center}
\begin{tikzpicture}
\coordinate [label=left:{a}] (a) at (0,0);
\coordinate [label=right:{b}] (b) at (3,0);
\coordinate [label=above:{c}] (c) at (1,2);
\draw (a) --(b) --(c) --(a);
\end{tikzpicture}
\end{center}


\begin{center}
\begin{tikzpicture}
\def \theang {20}
\draw
\draw[fill=LightYellow]
\end{tikzpicture}
\end{center}

Note that calculation can be done within the \texttt{tikzpicture} environment. Arithmetic is allowed and there are certain functions that can be used (there will be more on this later). This ability to perform calculations along with custom commands can allow for some very clever tricks. In the example below a variable called \texttt{\theang} is defined which controls the angle of inclination. This variable is defined to be 20 below but if it is changed to something else then the entire diagram will shift to reflect that change.

\begin{center}
\begin{tikzpicture}[scale=.8]
\def \theang {20}
\draw
\draw[fill=LightYellow]
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
\def \theang {20}
\draw
\draw[fill=LightYellow]
\end{tikzpicture}
\end{center}
These commands alone can make a lot of useful diagrams. Some examples are given in Figure 1. Note that colors are defined in these examples using the \texttt{svgnames} option in the \texttt{xcolor} package \cite{19}.

![Result:](image)

Figure 1: Some diagrams from MATH 112 that were made using the \texttt{tikz} package.
4.3 Graphing Functions

One of the biggest needs that math teachers have when it comes to using images is graphing functions. The obvious way to place the graph of a function in a document is to use the \includegraphics command that was introduced in Section 4.1. I do not ever do this myself, but this method should not be discarded. If you have a way of graphing functions (Mathematica [34] and Matlab [21] are popular choices) then you can easily place those graphics into your document.

I prefer to use the \texttt{tikz} [30] package to create graphs of functions. This preference stemmed from a desire to keep documents self-contained. I find it inconvenient to keep track of external files. It is harder to make small changes to external files and it is easier to send a single file to other people than it is to send whole groups of files. I continue to use \texttt{tikz} because I find it to be a convenient and fully-featured solution to several difficult problems.

One should keep in mind that \texttt{tikz} is not a calculator. It has some basic calculation features but it is not computational software. As such, there are occasional limitations to what sorts of things that \texttt{tikz} can do. It can handle functions of a moderate complexity and it can always handle a Taylor approximation. However, for complex functions it is better to use the features of the \texttt{pgf} package (actually the \texttt{pgfplots} package as we will see later) instead of the \texttt{tikz} package. The downside is that functions drawn with the \texttt{tikz} package can be decorated using the package's other features, while functions drawn with the \texttt{pgf} package will be displayed more like what would be displayed on a graphing calculator.

The \texttt{tikz} package contains a multitude of advanced features for graphing functions. We will only cover one of them here: the \texttt{plot} command. The syntax of the \texttt{plot} command is shown below; axes have been added for convenience.

\begin{verbatim}
\begin{tikzpicture}
  \draw[<->] (-1,0) -- (7,0) node[right]{\tiny {$x$}};
  \draw[<->] (0,-1) -- (0,4) node[above]{\tiny {$y$}};
  \draw plot[domain=0:6] ({\x},{0.13*(\x-1)*(\x-3)*(\x-5)+2});
\end{tikzpicture}
\end{verbatim}

Note that the \texttt{draw} command actually plots a parametric function. The parameter is \texttt{x} and the \texttt{domain} argument specifies the interval on which the parametric function is plotted. That is, the contents of \texttt{\{<first coord>,<second coord>\}} are what get plotted as a function of \texttt{x}. For extremely detailed graphs (maybe something like \texttt{sin(1/x)}) one can include the \texttt{samples} argument in the \texttt{plot} command to increase the number of discrete points on which the parametric function is evaluated. For example, a call to \texttt{draw plot[domain=0:1,samples=500]{\ldots,\ldots}} will plot 500 points on the interval \texttt{[0,1]} and connect them with a line.

The \texttt{draw} command can be augmented as before to create more attractive and visually striking graphs.

\begin{verbatim}
\begin{tikzpicture}
  \draw[help lines] (-0.7,-0.7) grid (6.7,3.7);
\end{tikzpicture}
\end{verbatim}
Remember that \texttt{tikz} understands certain functions. The ability to do calculation becomes particularly useful when graphing functions. See the documentation for a complete list of available functions; several examples are shown below:

\texttt{Code:}

\begin{verbatim}
\begin{tikzpicture}
l\draw[help lines] ( -5.1 , -4.1) grid (5.1 ,4.1);
l\draw[->] ( -5.3 ,0) -- (5.3 ,0) node [right]{\tiny \textit{x}};
l\draw[->] (0 , -4.3) -- (0 ,4.3) node [above]{\tiny \textit{y}};
l\begin{scope}
l\clip ( -5.1 , -4.1) rectangle (5.1 ,4.1);
l\draw[ultra thick, DarkGreen] plot[domain=0:5.2, samples=100] (\x,{sqrt(\x)});
l\draw[ultra thick, DarkBlue] plot[domain=-5.2:5.2, samples=100] (\x,{pow(\x,2)});
l\draw[ultra thick, Red] plot[domain=-5.2:5.2, samples=100] (\x,{exp(\x)});
l\draw[ultra thick, Orange] plot[domain=0.005:5.2, samples=100] (\x,{log10(\x)});
l\draw[ultra thick, Brown] plot[domain=0.01:5.2, samples=100] (\x,{log2(\x)});
l\draw[ultra thick, Gold] plot[domain=-5.2:5.2, samples=100] (\x,{abs(\x)});
l\draw[ultra thick, Lime] plot[domain=-5.2:5.2, samples=100] (\x,{sin(\x \ r)}) ;
l\draw[ultra thick, Coral] plot[domain=-5.2:5.2, samples=100] (\x,{cos(\x \ r)}) ;
l\end{scope}
\end{tikzpicture}
\end{verbatim}

\texttt{Result:}
Note the use of the `\clip` command. This tells `tikz` not to print anything outside of the defined rectangle. This is necessary as \( y = e^x \) gets very large on \([-5, 5]\) and we don’t want the point \((5, e^5)\) printed. The `scope` environment ensures that the only things affected by the `\clip` command are those within it.

To this point, every one of the examples have used the parametric function in the `plot` command to plot a single valued function. That is, the `plot` command has been used in the form \((x, {f(x)})\). Of course, this can be used to plot any parametric function. Plotting \((\{\cos(x)\}, \{\sin(x)\})\) will result in (some part of) a circle. One use for this that I have found is the example shown below:

```latex
\begin{tikzpicture}[scale=2]
\def\angle{1500} % angle you're drawing
\def\factor{10} % total available "loops"
\def\start{2} % "loop" at which the arc starts
\draw[<->] (-1.3,0) -- (1.3,0);
\draw[<->] (0,-1.3) -- (0,1.3);
\draw[DarkGreen, thick] (0,0) circle (1);
\draw[Red, thick] (0,0) -- (\angle:1);
\draw[-,Red] plot[domain=360*\start:360*\start + \angle, samples=200]\( (x,x / (360*\factor )) \);
\draw ({\angle / 2}:{ .85 }) node[Red]{\text{tiny}{\ang{\angle}}};
\end{tikzpicture}
```

**Result:**

![Graph showing the parametric function example](image-url)
Of course, the examples are limitless. Figure 2 gives two final examples of functions graphed with \texttt{tikz}.

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\centering
\begin{tikzpicture}
\addplot[domain=-1:1, samples=100, smooth] {x};
\addplot[red, dashed, domain=-1:1, samples=100, smooth] {x};
\addplot[blue, domain=-1:1, samples=100, smooth] {1/x};
\end{tikzpicture}
\caption{A function and its inverse.}
\end{subfigure}
\hspace{2cm}
\begin{subfigure}{0.45\textwidth}
\centering
\begin{tikzpicture}
\addplot[domain=-1:1, samples=100, smooth] {x};
\addplot[red, dashed, domain=-1:1, samples=100, smooth] {1/x};
\addplot[blue, domain=-1:1, samples=100, smooth] {x + 1};
\end{tikzpicture}
\caption{Examining how to add two functions.}
\end{subfigure}
\caption{Some examples of graphing functions with \texttt{tikz}.}
\end{figure}

### 4.4 Advanced Tikz Examples

The \texttt{tikz} package offers a whole world of possibilities (don’t you dare close your eyes). Here we explore some of those possibilities. The code is not provided. You can look up the package yourself or check the source code for this document.

**The \texttt{pgfplots} package**

It was mentioned that the \texttt{tikz} package is a subset of the \texttt{pgf} package and that it is not always the best way to graph a function. Here we show some examples of how to use the \texttt{pgfplots} package \cite{pgfplots} to graph functions. Truth be told, the \texttt{pgfplots} package has a more intuitive syntax for these graphs. However, I prefer to use the \texttt{tikz} package whenever possible so that I can utilize its other conveniences. To use the \texttt{pgfplots} package, put the following lines in the preamble:

```
\usepackage{tikz}
\usepackage{pgfplots}
```

Figure 3 shows two examples of the visual style of graphs produced with the \texttt{pgfplots} package.
One thing that the \texttt{pgfplots} package can do that the \texttt{tikz} package cannot do is graph functions in three dimensions. Figure 4 shows an example of such a thing.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example3d.png}
\caption{The graph of \( z = f(x,y) \) where \( f(x,y) = x^2 - y^2 \) on a domain of \([0, 1] \times [-1, 1] \).}
\end{figure}

\textbf{The \texttt{spy} Library}

The \texttt{tikz} package comes with several libraries. One of them is the \texttt{spy} library. This library allows you to zoom in on a particular part of a \texttt{tikz} image. It is not something that I use regularly but it is very handy when I need it. To use the \texttt{spy} library be sure to add the following lines to your preamble:

\begin{verbatim}
1 \usepackage{tikz}
2 \usetikzlibrary{spy}
\end{verbatim}

Figure 5 shows two examples of the types of things the \texttt{spy} library can do.

\textbf{The \texttt{fit} Library}

Another of the \texttt{tikz} libraries is the \texttt{fit} library. Its purpose is to define options for specifying a node to contain a set of coordinates. It is a useful way to highlight things in an equation. I will not go into the details of such things, except to provide the example in Figure 6. This example was written by Kottwitz [20] and found online. Just like with the \texttt{spy} library, the \texttt{fit} library is used by putting the following in your preamble:

\begin{verbatim}
1 \usepackage{tikz}
2 \usetikzlibrary{fit}
\end{verbatim}
4.5 Figures

Images created with the \texttt{tikzpicture} environment or imported with the \texttt{\includegraphics} command can be placed in a document on their own. However, in larger documents it can be helpful to place them in a \texttt{figure} environment. This allows them to be labeled, it provides a convenient command for captioning, and it turns them into a “float” so that \LaTeX{} can try to place them where they fit the best. Any figure which has a label in this document was created within a \texttt{figure} environment. For example, the code shown below was used to produce Figure 7.

```latex
\begin{figure}[ht]
\centering
\includegraphics{cat}
\caption{A very pretty kitty.\label{fig:fluffy}}
\end{figure}
```

Of course, a \texttt{tikzpicture} environment can be used in place of the \texttt{\includegraphics} command. The optional argument \texttt{[ht]} tells the compiler that the figure should be placed exactly where it is in the code (that’s the \texttt{h}) and if that can’t be done then it should be placed at the top of a page (that’s the \texttt{t}).
One may notice that several of the figures in this document have subfigures. For example, Figure 2 contains Figure 2a and Figure 2b. This is accomplished with the subfig package [7].

5 Final Thoughts

This document is concluded with a list of several other miscellaneous packages that teachers may find useful. No example code is provided; I leave you to research them on your own.

Crossing things out: When providing solutions for students it can sometimes be nice to “cancel” things out. Conveniently, this functionality is provided by the cancel package [3]. This is how we typeset the following equation:

\[
\frac{x^2 + 5x + 6}{x^2 - 2x - 8} = \frac{(x+2)(x+3)}{(x+2)(x-4)} = \frac{x + 3}{x - 4}
\]

Tables: \LaTeX{} can format tables by default using the tabular environment. A nice extension of this environment is provided by the tabularx package [6].

Code: There are many packages available for typesetting code. The key is that you want to be able to type \LaTeX{} without the command being compiled. This can be accomplished in-line with the \verb{} command. To typeset \LaTeX{} you simply type \verb{\LaTeX{}}. By default, \LaTeX{} provides a verbatim environment for typesetting several lines of code in this manner. Personally, I prefer the added functionality of the listings package [23] which is what was used to typeset the code examples in this document.

Make sure to know the difference between when the \verb{} command and verbatim environments (or the like) are appropriate and when you simply want fixed-width font. An easy way to get the font is with the \texttt{} command. Note that \verb{monospace} and \texttt{monospace} both render “monospace.” There are advantages and disadvantages to both commands.

Multiple columns: From time to time it is useful to use multiple columns in a document. This can be accomplished with the multicol package [22]. It is an excellent and fully-featured package. If you need to increase or decrease the size of margins temporarily while changing the number of columns, I find that the best solution is to use the adjmulticol package [32].

The boxes used in this document: The boxes used by this document to surround the snippets of code are created with the colorbox package [28]. If you find that the colorbox package does not do what you need there is similar functionality provided by the mdframed package [10]. The choice between these two excellent packages is a matter of personal preference.

Presentations: It is my personal opinion that mathematicians should not use anything other than \LaTeX{} to typeset projector-style presentations (or most anything else, really). The popular solutions provided by Google and Microsoft are user friendly at first, but they are inconvenient for typesetting mathematics. And, frankly, other mathematicians can tell.
Use the \texttt{beamer} document class \cite{37} to create presentations in \LaTeX{}. For those who are already familiar with typesetting documents using \LaTeX{} it is a very smooth transition to learning how to typeset in \texttt{beamer}.

Remember that package load order matters. If you have a document which loads many packages then you need to be careful about the order of the \texttt{\usepackage} commands. The general rule is that broad packages should go first and narrow packages should go last. The \texttt{babel} package should be added early so that the rest of your packages know the language that you’re using and the \texttt{amsmath} package should also be added early because many packages alter functionality that is first implemented there. It also makes good sense to add the \texttt{geometry} package early.

Sometimes there are unavoidable package conflicts. If two packages both contain the same two command definitions and you want one from each, then there is no load order that will make that happen. However, keep in mind that there is always a way to get the commands that you need. The packages that you are using exist somewhere on your computer. In the worst case scenario you can always open them, find the commands, and place their definitions in the preamble of your own document which eliminates the need to import the packages.
References


