Contents

Introduction

1.1 How it's different from Word

1.1.1 WYSIWYG - What You See Is What You Get

You may be familiar with the challenge that is using Word to write equations even as simple as those in Figure 1.1. The formatted text on the page is exactly everything you have, which often means spending time trying to fix how it looks and what goes where, a process we call typesetting. So **what is the alternative?**

The Laplace transform of a function $f(t)$, defined for all real numbers $t \geq 0$, is the function $F(s)$, which is a unilateral transform defined by:

$$F(s) = \int_0^\infty f(t)e^{-st}dt$$

Figure 1.1: Word, an example of What You See Is What You Get

1.1.2 Typesetting

Let me introduce you to LaTeX, pronounced either *lah-tec* or *lay-tec*.

– **stuff here**.

The main motivators for why you would want to use it are:

1. Beautifully written documents. No more trying to deal with the various fonts, margins, spacing, etc.;
2. Easy bibliography management, citations and cross-references;
3. Easy equations and graphs. Even extremely complicated mathematical concepts can be easily written.

Let's go back to our example, and see how it would be done with LaTeX:

Listing 1.1: Example document written with LaTeX

```
1 \documentclass[12pt,a4paper]{article}
2 \usepackage{amsmath}
3 \begin{document}
4 The Laplace transform of a function $f(t)$, defined for all real numbers $t \geq 0$ is the function $F(s)
      $, which is a unilateral transform defined by:
5 \begin{equation*}
6     F(s) = \int_0^\infty f(t)e^{-st} dt
7 \end{equation*}
8 \end{document}
```

If you are familiar with programming, this may vaguely look like a *markup language*, and it is! If you are not, don't worry. We will introduce the most distinctive features, and go into more detail as necessary.

1.1.3 Markup language

There are generally three types of markups:

1. **Environments** introduce some kind of formatting, such as lists, math mode or more complicated things. A backslash, `\`, is used to indicate a markup, curly brackets `{}` indicate an *argument*, and square brackets `[]` (optionally) provide options.

   With very few exceptions, environments have the following format:

```
1     \begin{environment-name}[options]
2         ...
3     \end{environment-name}
```

   One of the exceptions is in Listing 1.1. Can you spot it? It is `$...$`, the *in-line math mode* environment. We will be exploring it in more detail later.

2. **Instructions** define some feature of the document. This ranges from breaking a page to defining the headings you have available. In our example you can see:

   `\documentclass[...]{article}`

3. **Variables** can either be predefined or user defined, and these range from greek letters to the integral sign to a whole expression. Some examples seen are `\geq` (**g**reater or **eq**ual) and `\int` ($\int$). Intuitively, `\alpha` results in α, and similarly for other greek letters.

   Note: There are some characters with predefined meaning, such as `{`, `}` and `&`. To use the literal curly brackets, ampersand, etc we would need to *escape* them. Conveniently, this is done with a backslash (\), so feel free to think of it as a variable: `\{`, `\}` and `\&`.

1.1.4 Packages

You may have noticed that `\usepackage{amsmath}` was not mentioned as an instruction the previous section. That's because packages are worth mentioning on their own.

Packages add features to our document, similar to *import* in most programming languages. `amsmath` gives us a wide array of maths tools, but there are packages for drawing, graphing, colouring, better management of bibliography, easier organisation of your files... and the list goes on.

1.1.5 Compilation

We can use any text editing software to write and save our documents in `.tex` files. In order to produce a `.pdf`, it needs to be *compiled*. As a beginner you may spend a lot of time scratching your head, wondering why it's not compiling.

The good news is that recommended IDEs (**I**ntegrated **D**evelopment **E**nvironments, basically text editors filled with features) handle the compilation for you and have features to help you spot mistakes.

Installation

2.1 Installing LaTeX

2.2 Installing an IDE

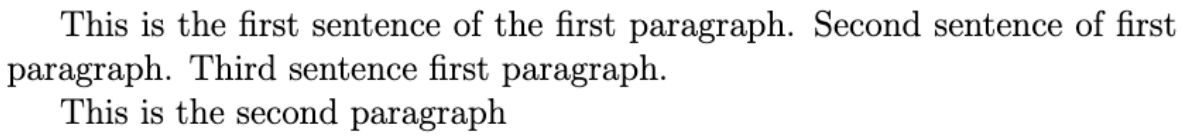
Getting started

3.1 My first document

A LaTeX file has a `.tex` extension, and begins with what we call a *preamble* — declaring the *class* of document, followed by `\begin{document}...\end{document}`.

```
1 \documentclass[12pt]{article}
2 \begin{document}
3 This is the first sentence of the first paragraph. Second sentence of first paragraph.
4 Third sentence first paragraph.
5
6 This is the second paragraph
7 \end{document}
```

Producing the following output:



Generally, we declare a document class with `\documentclass[option1, ...]{class}`, with the most commonly used classes being `article` and `report`. Every class has a different set of default behaviours, such as `report` providing a title page, but we can give give it *options*. The option we set was to change the font size from the default `10pt` to `12pt`.

Another option commonly used in academia is `twocolumn` to produce a two column document. You can find more options and information on default behaviour on this link.

Later on we will come back to the *preamble* for other important commands. Generally we create templates, so it isn't necessary to remember every small detail, and very quick to get started on a new document.

3.1.1 Paragraph

You will notice that the first paragraph consists of both lines 3 and 4. This is because a paragraph is only created by having a full empty line (like line 5). One advantage to separating sentences by a new line is that you can more readily move, copy and delete them in your editor.

Another important feature is that indentation was made automatically. LaTeXis smart enough to indent for you and almost always get it right. If you really want to force a paragraph without indentation, use \\ at the end of the previous one, like so:

```
1    paragraph one\\
2    paragraph two not indented.
```

Note Including an empty line after \\ will result in a very common warning: `Underfull hbox`. More information on this later in the common errors and warnings section.

3.1.2 Sectioning

The basic way we separate documents is into `section`, `subsection` and `paragraph`.

Listing 3.1: Caption

```
1  \documentclass{article}
2  \begin{document}
3  \tableofcontents
4  \section{First header}
5  text text.
6  \subsection{Counted subheader}
7  \paragraph{Leading text}
8  normal text that follows.
9  \subsection*{Uncounted subheader}
10 \section{Second header}
11 \end{document}
```

Results in:

Every tag that includes some form of counting can have an asterisk (*) to remove the counting. In this case, you can see the difference between `\section{}` and `\section*{}`, and most importantly, the table of contents, generated with `\tableofcontents`, excluded the uncounted subheader.

The `report` class also offers `\chapter(*){}` and `\part(*){}`, relevant mostly to very large documents.

Note: You will notice that the table of content and the actual content are in the same page. If you want a page break at any point, just use `\pagebreak`!

Contents

1 First header

text text.

1.1 Counted subheader

Leading text normal text that follows.

Uncounted subheader

2 Second header

3.1.3 Bold, italic, underline, etc

```
\textit{Italics}, \underline{underline}, \textbf{bold}.
\textit{Emphasis switches from ``italics'' to \emph{normal}} and \emph{vice-versa} based on context.
\texttt{And we can even get monospace!}
```

Results in:

Italics, underline, **bold**. *Emphasis switches from "italics" to* normal and *vice-versa* based on context. `And we can even get monospace!`

VSCode has a shortcut for these, so you don't need to remember the exact keyword. `Ctrl+L` to initiate a LaTeX command, then `Ctrl+` the first letter of the command — `Ctrl+i`, `Ctrl+b`, `Ctrl+u`, `Ctrl+e` or `Ctrl+t`, respectively. So for bold, you would press is `Ctrl+L+Ctrl+B`. For Mac, just replace `Ctrl` for `Cmd`.

Quotations are done with `` ``text here'' ``. When you type `` ` ``, the editor will automatically insert `'`.

Note: Generally the suggestion is to use `\emph{}` over `\textit{}`. Think of it as a "generic highlighter" that you can modify with default behaviour to *italicise*, but you could make it change colours or font size or anything else.

3.2 Bibliography management

The tool that allows us to easily manage bibliography is called BiBTeX. In particular, we are using a *package* called `natbib` that gives us some extra features. Using it has two distinct moments: Adding an entry to our bibliography, and citing.

3.2.1 Creating a bibiliography

A bibliography file has a `.bib` extension, and each entry has a very specific format. Let's start with creating the file `bibliography.bib`, so our working directory looks like this:

```
Example
├── bibliography.bib
└── example.tex
```

Each entry has a source, whether `article`, `book`, `misc` or many more and has the following format:

```
@article{ GerberLeahR2005, %Unique identifier
  author    = {Gerber, Leah R and Beger, Maria and McCarthy, Michael A and Possingham, Hugh P},
  title     = {A theory for optimal monitoring of marine reserves},
  issn      = {1461-023X},
  journal   = {Ecology letters},
  pages     = {829--837},
  volume    = {8},
  publisher = {Blackwell Science Ltd},
  number    = {8},
  year      = {2005},
  edition   = {Editor, Ransom Myers Manuscript received 15 March 2005 First decision made 21 April 2005
      Manuscript accepted 6 May 2005},
},
```

It's worth highlighting that the basic format is essentially `@article{ID,...}`, with each entry being separated by commas. BiBTeX will handle "et al" and other conventions as long as you stick to the following format:

```
author = {LastName1, FirstName1 and LastName2, FirstName2 and...}
```

The good side is that you rarely, if ever, need to type it out yourself. When you find an article through UCL's library, JAMA, Science Direct and many other resources, there will be an option to **export citation to BiBTeX**. Simply copy the contents to your bibliography file and you're ready to cite!

3.2.2 Manually adding a bibliography entry

While in a `.bib` file, you can easily create the skeleton for a bibliography entry. Simply type `@` and press `Ctrl+space`, it will give you suggestions. Generally this is only used for citing random websites, so you will want to pick the `@misc` option.

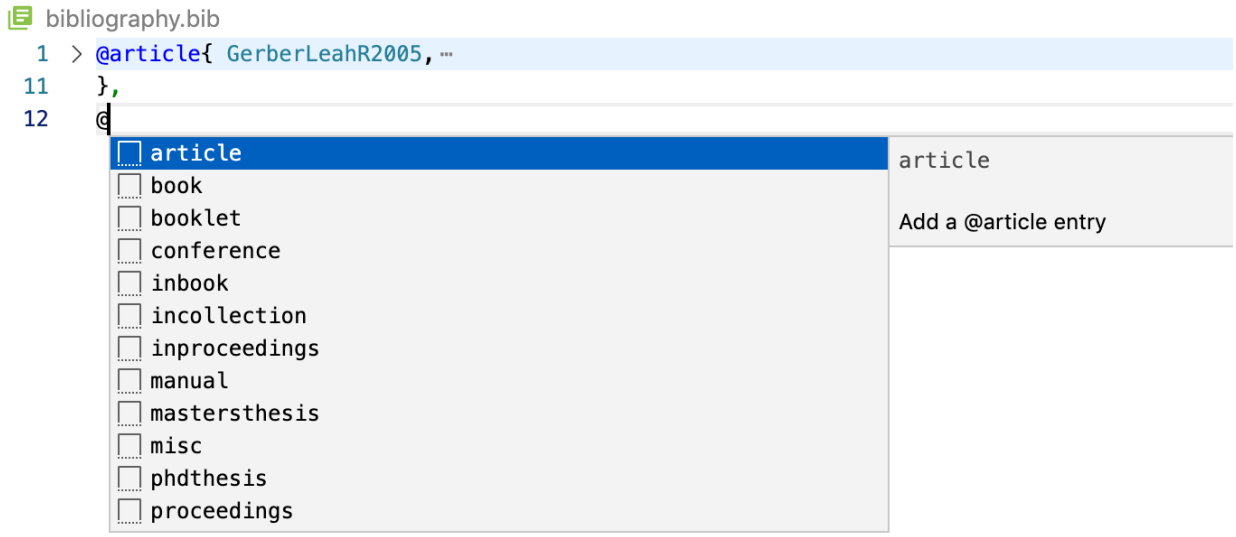


Figure 3.1: Autocompleting a bibliography entry in a `.bib` file.

3.2.3 Citations

Now we just need to let our document know where to find our bibliography file and we can use `\cite{}` (or its variants) to include citations. Every entry that that is cited, automatically gets added to a Reference section at the end of your document.

Listing 3.2: example.tex

```
1 \documentclass[]{article}
2 \usepackage[square,numbers]{natbib}
3 \bibliographystyle{unsrtnat}
4 \begin{document}
5 Citations are made so easy with \LaTeX, can you see \cite{GerberLeahR2005}?
6 \bibliography{bibliography}
7 \end{document}
```

Giving us the following output:

The options in `\usepackage[square,numbers]{natbib}` are what determine that in-text citations is **[1]**. Alternatively if you prefer **(Gerber et al, 2005)**, use `\usepackage[round]{natbib}`, and `\citep{}` (see

Citations are made so easy with LaTeX, can you see [1]?

References

[1] Leah R Gerber, Maria Beger, Michael A McCarthy, and Hugh P Possingham. A theory for optimal monitoring of marine reserves. *Ecology letters*, 8(8): 829–837, 2005. ISSN 1461-023X.

code below). It is also possible to do narrative style citations, such as "In their work, Gerber et al (2005) describe…". This is achieved with `\cite{}` with this same setting.

```
1 \usepackage[round]{natbib}
2 ...
3 \citep{GerberLeahR2005}
```

The `natbib` package gives us the `bibliographystyle{unsrtnat}` option, which determines the style of the references. There are other styles, as well as more information on `natbib` on this link.

A really important feature of VSCode is Intellisense, these automatic sugggestions of authors (and more), which it finds from our `.bib` file. If you are not getting suggestions, try activating Intellisense by pressing `ctrl+space`. It also works with normal commands and tags starting with \.

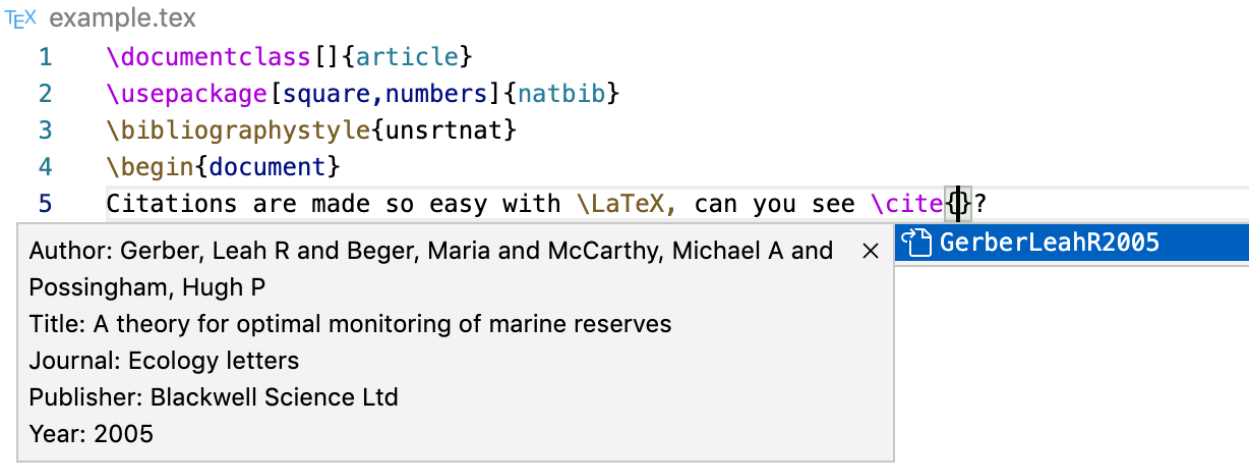


Figure 3.2: Autocomplete from our bibliography file

Automatically generating numbers for figures, tables and correcting any citations is a key feature of LaTeX. This means we can easily refer to a figure, move it around and it will correctly choose its number. Before we get into adding all that it is a great idea to take a detour and discuss organisation.

3.3 Environments

3.3.1 Staying Organised

So far our examples have been extremely short, but imagine having dozenschapters with dozens of packages and whatever configuration is necessary for them. One thing we can do is split up our document into the *preamble*, the file `main.tex`, where we have all the setup, and the *content*. The content can be split up whichever way you see fit — with the more complicated it is, the more you can split it up.

Let's expand our working directory, wherein we are typesetting three sections of our article. This is `Example2` .

Note: Don't worry too much about minor details. Templates cover most of the usual needs, important is just understanding the motivation, so you can use/edit solutions you find on the internet.

```
Example2
├── bibliography.bib
├── main.pdf
├── main.tex
├── code
│   └── code.m
├── figures
│   └── gradient-circle.png
└── files
    ├── introduction.tex
    ├── animal-rights.tex
    └── animal-lefts.tex
```

Let's first take a look at our preamble file, `main.tex`.

Listing 3.3: `main.tex`

```
1 \documentclass{article}
2 \usepackage{geometry}
3 \geometry{top=1.0in, bottom=1.0in, left=1.0in, right=1.0in}
4 \usepackage{setspace} \doublespacing
5 \usepackage{import}
6 \usepackage{tikz}
7
8 \usepackage{listings}
```

```
\lstset{
        numbers=left, frame=single, breaklines=true, %Keep text inside a frame, and number each line.
        basicstyle = \scriptsize\ttfamily, %smaller size, monospaced
}

\begin{document}
\section{Introduction}
    \subimport{files/}{introduction.tex}
\section{Animal Rights}
    \subimport{files/}{animal-rights.tex}
\section{Animal Lefts}
    \subimport{files/}{animal-lefts.tex}
\end{document}
```

`\usepackage{geometry}` and its command `\geometry{}` allows us to set each margin individually. APA styling suggests 1 inch all around, but you may need to adjust the left margin if you are binding your thesis.

The package `setspace` and its command `\doublespacing` provides automatic double spacing.

To "inject" the contents of another file into our preamble, the `import` package gives us `\subimport{}{}`. The first bracket requires a relative path starting from the root of your working directory, and the second bracket is the name of the file.

VSCode will help you navigate the folders and find the files. As you star typing `\subimport` it will offer the command as a suggestion. Select it by pressing `tab`, navigate to choose the folder called `files/`, then press `tab` to select. Press `tab` again to move to the second set of brackets. If it doesn't display options, press `ctrl+space`, and you can pick the file.

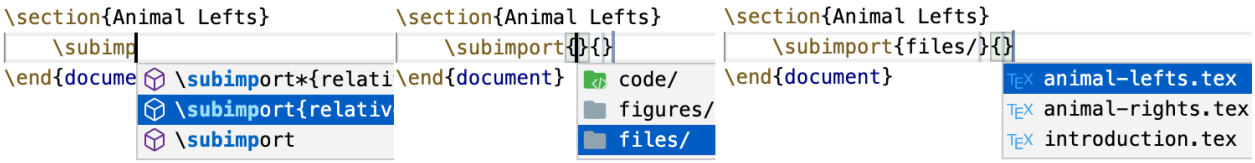


Figure 3.3: Intellisense-assisted picking files.

Finally, the `tikz` package is what we use for mathematical drawing, graphing, importing pictures, and much more. It will be used plenty in the coming section.

The biggest advantage of this separation is that each content file has absolutely no configuration at all. This is called *Separation of concerns*, and it allows us to just focus on the content, and all of the setting up comes from a template with minor tweaks. Our `introduction.tex` file, then, looks like this:

Listing 3.4: `introduction.tex`

```
\begin{figure}[h]
```

```
    \centering
    \includegraphics{figures/gradient-circle.png}
    \caption{This is our first picture}
    \label{fig:gradient-circle}
\end{figure}
```

With the output:

1 Introduction

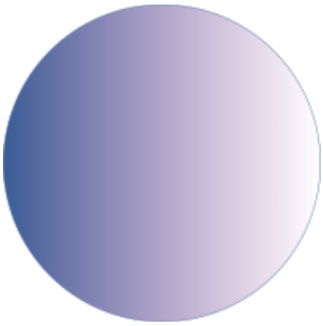


Figure 1: This is our first picture

3.3.2 Figure and caption

`\begin{figure}...\end{figure}` creates a figure *environment*. LaTeXtries to find the best position for every environment, but you can force their position by passing the option `[h]`. Usually contents are left-adjusted, so we can push the environment to the centre by using `\centering`. `\caption{}` automatically numbers sequentially.

`tikz` provides the `\includegraphics{}` command that imports our picture. Often we need to scale pictures, which can be achieved with the options `width=0.5\textwidth` or `scale=0.8`. `\textwidth` is a LaTeX variable which automatically calculates the usable size of your document. So in order to scale the picture to 0.5 of the textwidth, we would use:

```
\includegraphics[width=0.5\textwidth]{figures/gradient-cicle.png}
```

3.3.3 Label and cross-reference

The `\label{}` command is paired with `\ref{}` for automatic cross-referencing across any file of our document. Take a look at our `animal-rights.tex` file. We are able to reference both the table in this file and the figure in another file, so it's important to be very explicit with our labels, so you can actually find them!

The suggestion is to use `fig:file-name` for figures, `eq:name` for equations, and so on. This way you can "filter" while you navigate through the suggested names.

Listing 3.5: `animal-rights.tex`

```
This information can be found in Table \ref{tb:risk}, or in the circular shape of Figure \ref{fig:
    gradient-circle}
\begin{table}[h]
    \centering
    \begin{tabular}{r|lc}
        (1,1) & (1,2) & Third Column,first Row \\
        \hline
        Column1 & Column2 & Column3 \\
        Column1 & Column2 & Column3 \\
    \end{tabular}
    \caption{This is our first table}
    \label{tb:risk}
\end{table}
```

which looks like this:

2 Animal Rights

This information can be found in Table 1, or in the circular shape of Figure 1

(1,1)	(1,2)	Third Column,first Row
Column1	Column2	Column3
Column1	Column2	Column3

Table 1: This is our first table

3.3.4 Table

`\begin{table}...\end{table}` creates a table *environment*, which is different from creating a table itself. `table` has similar properties to `figure`, allowing you to set a caption, label and position.

`tabular`, on the other hand, creates a table. This is always followed with curly brackets deciding the **number of columns**, the **adjustment** of the text and whether there are **vertical dividers**. `{r|lc}` means a right-adjusted column with a vertical divider, a left-adjusted column, and a centre-adjusted column.

Each column is separated by `&` and each row is separated by `\\`. `\hline` is used to produce a horizontal line that separates titles from content.

Note: As you may have noticed, some characters have special meaning, like &, {, } and \. To display the literal symbol, it needs to be *escaped* by a preceeding \, like this: `\&`, `\{`, etc. The backslash is an exception, because `\\` is also a special character, so you have to use `\textbackslash`.

3.3.5 Lists

There are two types of lists: numbered and unnumbered, and these are `enumerate` and `itemize` environments, respectively. Take a look at the code in `animal-lefts.tex`.

Listing 3.6: animal-lefts.tex

```
\begin{enumerate}
    \item First numbered item
    \item Second numbered item. Let's nest another list
    \begin{itemize}
        \item First itemised
        \item Second itemised
    \end{itemize}
\end{enumerate}
```

Resulting in:

3 Animal Lefts

1. First numbered item
2. Second numbered item. Let's nest another list
    - First itemised
    - Second itemised

A new entry is only created with `\item`, so you can have as much code between entries as you want, including other environments and nestings of `enumerate`, like this:

```
\begin{enumerate}
    \item First question
        \begin{enumerate}
            \item Sub question
                \begin{enumerate}
                    \item Item on subquestion
                \end{enumerate}
    \end{enumerate}
    \item Second question
\end{enumerate}
```

Resulting in:

1. First question
    (a) Sub question
        i. Item on subquestion
2. Second question

3.3.6 Code

The package `listings` allows presenting code in the exact way it is seen in this document. The appearance of the frame can be changed greatly, and `listings` is very well documented here. Generally, stick to the options presented in the preamble and in the templates, and it will cover most of your needs, namely:

```
\lstset{
        numbers=left, frame=single, breaklines=true, %Keep text inside a frame, and number each line.
        basicstyle = \scriptsize\ttfamily, %smaller size, monospaced
}
```

If you are looking for highlighting in the same way you'd find in your editor, look for the package `minted`. It requires a fair bit of setup, so it will not be covered here.

We have the option of writing out the code in the file inside the `lstlisting` environment or import from file with `\lstinputlisting{}`. Try it for yourself! Remember that an environment is always `\begin{environment-name}`...`\end{environment-name}`.

```
    \lstinputlisting[language=Matlab, caption={caption-here}, label={label-here}]{code/code.m}
```

Note: The `verbatim` environment, `\verb||` and `\texttt{}` produce monospaced fonts as well, and there are moments when each one is appropriate. Generally `\verb||` (which can also be done with `\verb!!`) escapes every character inside the `|...|` and is what one would use for short in-text code snippets which might interfere .

3.4 Maths

3.4.1 Equations, sums and alignment

3.4.2 Use of variables

3.4.3 Vectors and Matrices

3.4.4 Calculus

3.5 Graphs with Tikz

3.6 Circuits

3.7 Control systems

How do I...?

4.1 Search engine

4.2 Stack Exchange

4.3 CTAN package information

Working faster

5.1 Becoming familiar with the IDE

5.2 Using snippets