# 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

```
\documentclass[12pt,a4paper]{article}
\usepackage{amsmath}
\begin{document}
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:
\begin{equation*}
    F(s) = \int_0^\infty f(t)e^{-st} dt
\end{equation*}
\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:

   ```
   \begin{environment-name}[options]
       ...
   \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 $\alpha$, 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

Thre are two main options: TeX Live and MiKTeX, both put together all the underlying components that are necessary to create documents.

### 2.1.1 MiKTeX

MiKTeX can be downladed for Windows, Mac OS and Linux: link.

### 2.1.2 TeX Live

Link to all the instructions. The web page is pure text, so the navigation can be a little challenging. Below is some information which links to the appropriate area.

#### Mac

To install MacTeX, follow the instructions link

MacTeX will install TeX Live, plus a few other utilities. Either of the suggested editors will still need to be installed.

#### Linux

Your package manager will have a TeX Live distribution. Link to the Arch Linux wiki.

Fedora and other RPM based distributions can download it with

```
1    # dnf install texlive-scheme-full
```

Debian/Ubuntu:

```
1    # apt-get install texlive
```

## 2.2 Installing an IDE

### 2.2.1 Visual Studio Code

Download

VSCode is a general purpose editor, and if you plan on doing any programming at all, it may be worth becoming more familiar with it. Microsoft has an excellent introduction to it in their website.

In order to give VSCode the ability to work with and compile LaTeX, we must install the LaTeX Workshop extension. Start by opening the extensions on the side bar (shortcut: `ctrl+shift+x`). Search for "latex" and the first option should be LaTeX Workshop published by James Yu.

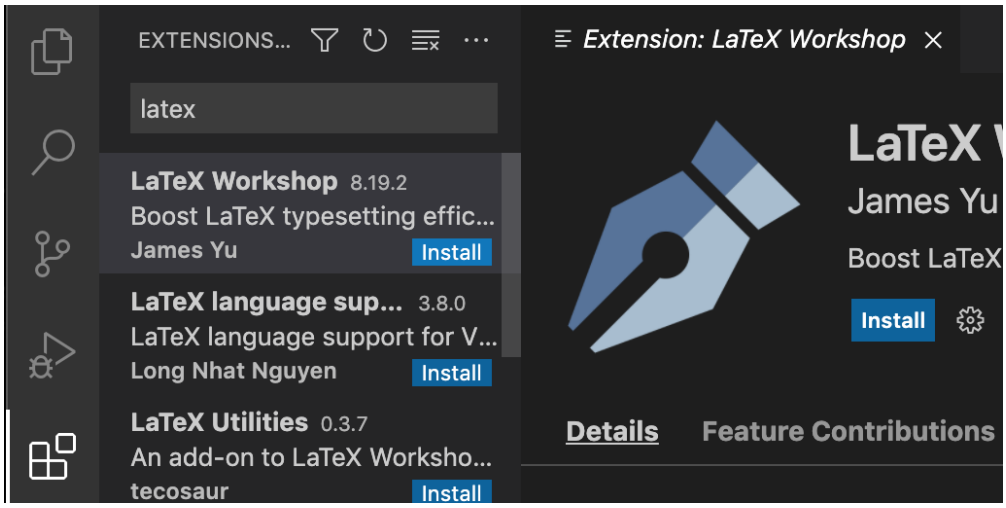



Figure 2.1: Installing LaTeX Workshop

#### Compilation

Unless changed in the settings, your `.tex` will be compiled on saving the file. If compile on save does not trigger, you can manually do it with a few different options:

1. Open the command pallet (`ctrl+shift+P`), type "latex build" and select the option "LaTeX Workshop: Build LaTeX Project"

2. On the top right, while in a `.tex` file, click the green arrow (`ctrl+alt+B`).

#### Other

The LaTeX Workshop extension should also give you access to another tab on the left-hand side. This includes navigating the document's structure and seeing helpful math symbols.

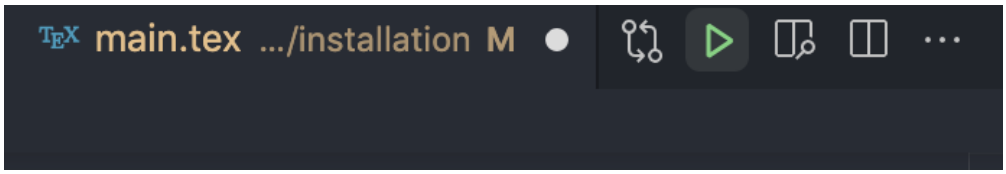



Figure 2.2: Manually building LaTeX project

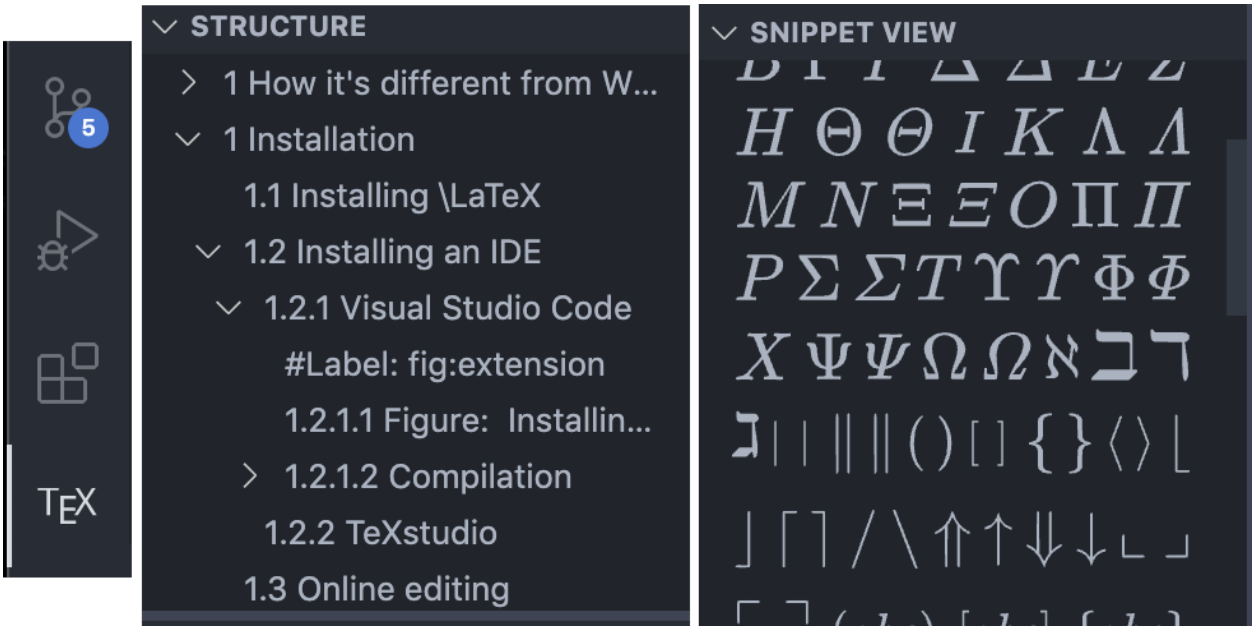



Figure 2.3: Accessing useful tools provided by LaTeX Workshop

### 2.2.2 TeXstudio

Download

TeXstudio is an IDE dedicated to LaTeX, so there is minimal setup. Simply download and you're ready to go!

## 2.3 Online editing

If you need a tool similar to Google Drive/One Drive for online collaboration, the most popular option is Overleaf. UCL students and staff have access to professional accounts on Overleaf, and the link to activate it can be found in the software database: here.

# 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:

This is the first sentence of the first paragraph. Second sentence of first paragraph. Third sentence first paragraph.

This is the second paragraph

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

```
1 \textit{Italics}, \underline{underline}, \textbf{bold}.
2 \textit{Emphasis switches from ``italics'' to \emph{normal}} and \emph{vice-versa} based on context.
3 \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 for the author:

```
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!

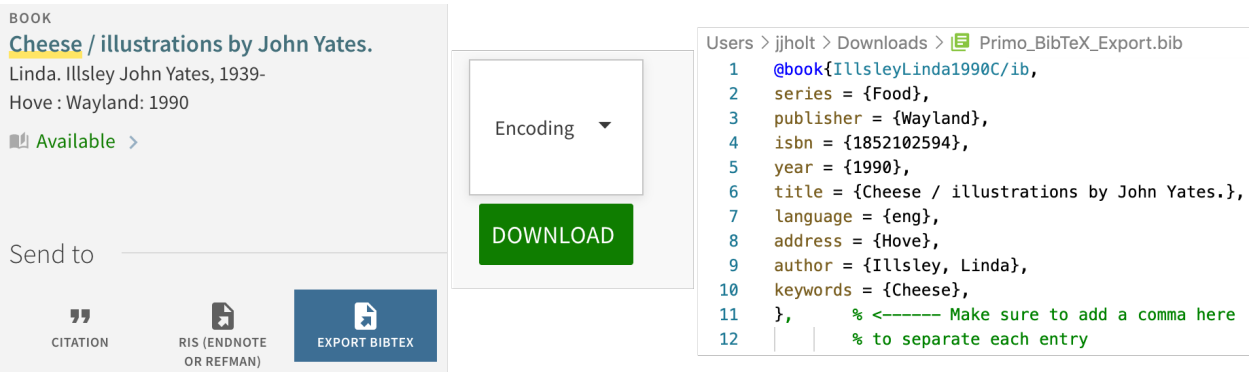



Figure 3.1: Getting a BiBTeX file from UCL's Library. From here, just paste to your `.bib` file.

### 3.2.2 Manually adding a bibliography entry

While in a `.bib` file, you can easily create a scaffold for a bibliography entry. Simply type `@` and press `Ctrl+space` to see the suggestions. Generally this is only used for citing random websites, for which we use the `@misc` option. The scaffold will include all the basic requirements to generate a good bibiliography entry.

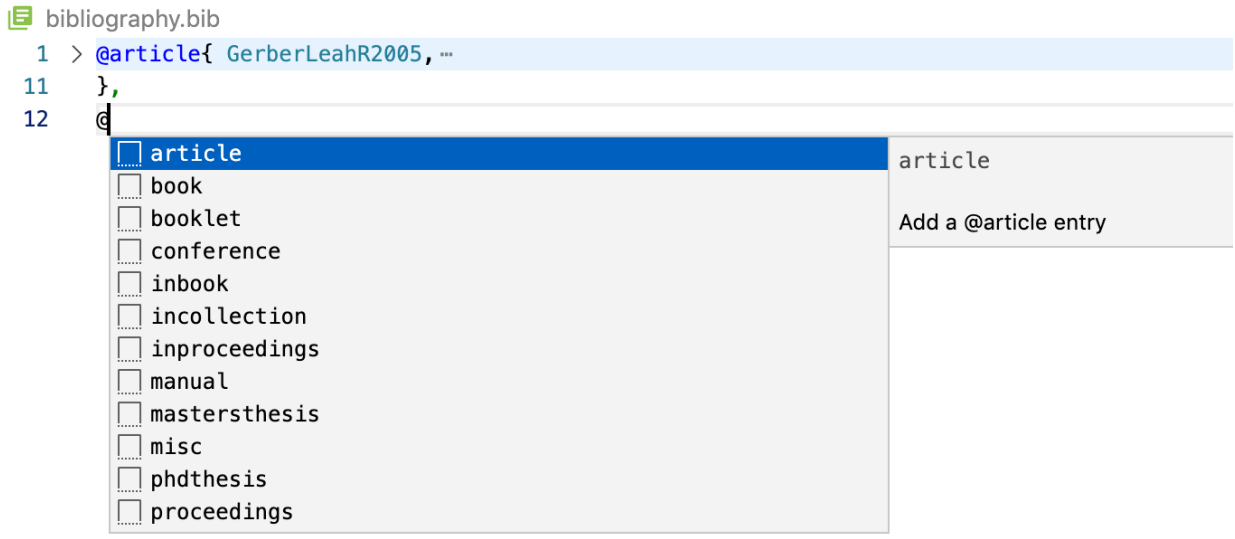



Figure 3.2: Autocompleting a bibliography entry in a `.bib` file.

### 3.2.3 Citations

Now we just need to let our document know where to find our bibliography file, which we had called `bibliography.bib` and we can use `\cite{}` (or its variants) to do in-text citations. Every cited entry automatically gets added to a Reference section at the end of the document. If you want to include any

non-cited entry, use `\nocite{id}`; and to include all entries: `\nocite{*}`.

Listing 3.2: example.tex

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

Giving us the following output:

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.

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 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.

```
\usepackage[round]{natbib}
...
\citep{GerberLeahR2005}
...
In their work, \cite{GerberLeahR2005} describe...
```

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

**Note:** When citing you may want to include a tilde (`~`) between the last word and `\cite{}`. This guarantees that the citation and the word are on the same line. Like so: `Tomorrow is a day~\cite{GerberLeahR2005}`.

A really important feature of VSCode is Intellisense, these automatic suggestions the editor finds depending on context. In the case of bibliography, it looks in the `.bib` file we indicate in the preamble, as you

can see in Figure 3.3. If you are not getting suggestions, try activating Intellisense by pressing `ctrl+space`. It also works with various other tags. Try typing \, then pressing `ctrl+space` to see the enourmous list.

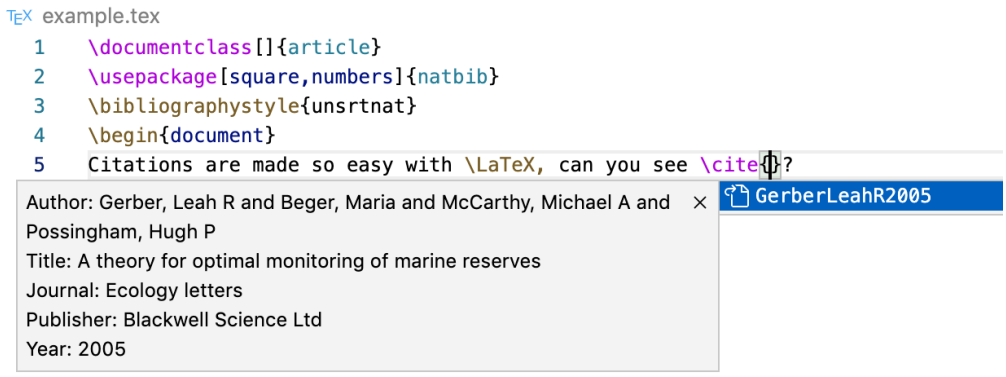



Figure 3.3: Autocomplete from our bibliography file.

Automatically generating the number for figures and tables, and easy cross-referencing are a key feature of LaTeX. This means we can refer to a bibliography entry or a figure by its unique id, 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 dozens of chapters 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 would want to split it up.

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 focus on the content completely independently from the setup and settings. In fact, generally we have templates and just do minor tweaks with every document.

Let's expand our working directory like seen below. These files can be found in `Example2` .

```
Example2
├── bibliography.bib
├── cheese-prices.dat
├── 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`

```
\documentclass{article}
\usepackage{geometry}
\geometry{top=1.0in, bottom=1.0in, left=1.0in, right=1.0in}
\usepackage{setspace} \doublespacing
\usepackage{import}
\usepackage{tikz}

\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 for binding a thesis, for example.

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

The package `listings` allows presenting code in the exact way it is seen in this document. We will discuss it more in the coming sections.

The `tikz` package is what we use for mathematical drawing, graphing, importing pictures, and much more. We will discuss it more in the coming sections.

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. You can include a `\subimport{}{}` in a file that has itself been `subimport`ed, and this is the key to organisation.

Intellisense will help you navigate the folders and find the files. As you type `\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 next *tab stop* (the second set of brackets). If it doesn't display options, press `ctrl+space`, and you can pick the file.

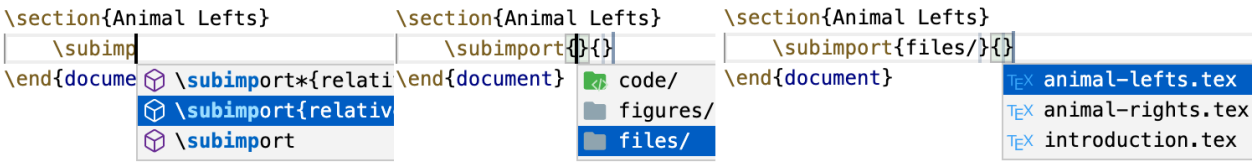



Figure 3.4: Intellisense-assisted picking files.

### 3.3.2 Figure and caption

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 as seen in Figure 3.5

`\begin{figure}...\end{figure}` creates a figure *environment*. LaTeX tries to find the best position for every environment, but you can force their position by passing the option `[h]`. Usually contents are left-

# 1 Introduction

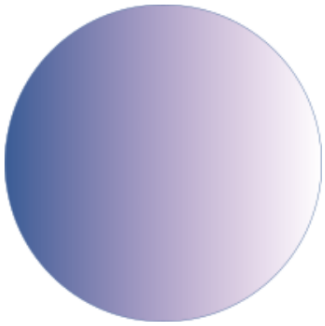


Figure 1: This is our first picture

Figure 3.5: Output from `introduction.tex`

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:

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

**Challenge:** Occasionally we need to put two pictures side-by-side. How would you achieve that?

The simplest way is to include both within the same `figure` environment and make sure their widths are less than half of the allowable text width: `width=0.5\textwidth`. If you need captions on each subfigure, then you are looking for the package `subcaption`, and you can find more information here.

```
1 \begin{figure}[h]
2 \centering
3     \includegraphics[width=0.49\textwidth]{figures/pic1.png}
4     \includegraphics[width=0.49\textwidth]{figures/pic2.png}
5 \caption{Two pictures in one figure environment}
6 \label{fig:pic1-pic2}
7 \end{figure}
```

### 3.3.3 Label and cross-reference

The `\label{}` command is paired with `\ref{}` for automatic cross-referencing across any file of our document. From the `animal-rights.tex` file we are able to reference both the table in this file and the figure in `introduction.tex`. This makes it 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. With larger documents, you may even want to include chapter references to you can "filter" the names: `fig:ch6:dog-with-tail`.

Listing 3.5: `animal-rights.tex`

```
1  This information can be found in Table \ref{tb:risk}, or in the circular shape of Figure \ref{fig:
       gradient-circle}
2  \begin{table}[h]
3      \centering
4      \begin{tabular}{r|lc}
5          (1,1) & (1,2) & Third Column,first Row \\
6          \hline
7          Column1 & Column2 & Column3 \\
8          Column1 & Column2 & Column3 \\
9      \end{tabular}
10     \caption{This is our first table}
11     \label{tb:risk}
12 \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.

**Challenge:** How would you create the following table? You can find a solution in the `Table` folder in the examples.

| Index | Cheese | Price (USD/kg) |
|---|---|---|
| 1 | Feta | 10.50 |
| 2 | Camembert | 12.20 |
| 3 | Edam | 8.95 |

Table 3.1: Cheese prices in local supermarket.

**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`.

### Table from data

Using the package `pgfplotstable` and its accessories described below, we can automatically generate tables. To be precise, it generates a `tabular` environment, so we would still want to put inside a `table` environment in order to add captions, labels, etc.

```
1 \usepackage{pgfplotstable, booktabs, array}
```

The data used here is `cheese-prices.dat` in the `Example2` folder.

Listing 3.6: `cheese-price.dat`

```
1 Index   Cheese  "Price (USD/kg)"
2 1       Feta    10.50
3 2       Camembert       12.20
4 3       Edam    8.95
```

Then the data can be imported with `\pgfplotstabletypeset{}`. By default, it expects only numbers — because we have text in our table, we indicate the `string type` option, and similarly `text indicator="` so text with spaces is kept together, `"Price (USD/kg)"`. Finally, the horizontal line after the head rows is created with the code in line 4.

```
\begin{table}[h]\centering
    \pgfplotstabletypeset[
        text indicator=", string type,
        every head row/.style={after row=\hline},
    ]{Example2/cheese-prices.dat}
    \caption{Replicated from a table!}
\end{table}
```

| Index | Cheese | Price (USD/kg) |
|---|---|---|
| 1 | Feta | 10.50 |
| 2 | Camembert | 12.20 |
| 3 | Edam | 8.95 |

Table 3.2: Replicated from a table!

Generally tables are best kept simple, but there is a lot of customising that can be done. The documentation for `pgfplotstable` can be found here.

### 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 3.7: 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

As mentioned earlier, the package `listings` creates an environment to present 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 bit of setup, but you can find more information here.

We have the option of manually writing the code in-file with the `lstlisting` environment. Try writing in-file for yourself! Tip: An environment is always `\begin{environment-name}`...`\end{environment-name}`.

It is also possible to import from another file with `\lstinputlisting{}`, and you can see an example

of it below. Don't worry about remembering it exactly — you will always be able to check documentation, the internet, or as we will describe at the end of this, use a *snippet*. Snippets are the key to typesetting documents incredibly fast, and will be covered extensively in the end.

```
\lstinputlisting[language=Matlab, caption={My Example code}, label={code:matrix}]{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 in-text code which might interfere with LaTeX itself.

## 3.4 Maths

There are two ways to create a maths environment: math mode and display mode. LaTeX is usually in text mode, but anything between `$...$` or `\(...)\` becomes math mode. A math environment allows subscripts, superscripts, greek letters and for most things one would associate with mathematics.

```
In text mode. Now maths: \(x^1+\alpha_0 = y^{2x}\)
```

In text mode. Now maths: $x^1 + \alpha_0 = y^{2x}$.

`{}` is how we indicate that everything inside the brackets should be treated together. This should become clear from the following code:

```
\( x^12 + x^{12} + x_0^2 + {x_0}^2 \)
```

and its result: $x^12 + x^{12} + x_0^2 + {x_0}^2$

**Display math** can be done in several ways. The suggestion is using the `amsmath` package, and either the `align` or `equation` environments. `align` numbers every single line, and `align*` does not. Similarly for `equation`.

To declare a new line, use \\, and `&` defines the point in the expression (if any) for horizontal alignment.

```
\documentclass{article}
\usepackage{amsmath}
\begin{document}
    \begin{align*}
        f(x) &= (x+1)(x-1) \cdot \left( \frac{\alpha}{\ln 5} \right)\\
        &= x^2 - 1 \left( \frac{3}{\ln 5} \right)
    \end{align*}
\end{document}
```

$$f(x) = (x+1)(x-1) \cdot \left(\frac{\alpha}{5}\right) + \log_2 7$$
$$= (x^2-1)(\frac{\alpha}{5}) + \log_2 7$$

The usual product symbol $\cdot$ is created with `\cdot`. $\times$, relevant for both cross product and scientific notation, is produced with `\times`.

The pair `\left( \right)` produce parenthesis that fit any vertical size. You will see the effect of them missing in the second expression. Similarly, they can be used with `\left\{ \right\}` to fix the size of `{}` and `\left[ \right]` for `[]`.

$$\left(\frac{\alpha}{5}\right) \qquad (\frac{\alpha}{5})$$

```
\begin{equation*}
    \left( \frac{\alpha}{5} \right) \qquad (\frac{\alpha}{5} )
\end{equation*}
```

Because math environments change the way letters are displayed, `\ln`, `\sin`, `\cos`, `\log`, `\exp` and more can be used to produce the font as we would expect.

`\frac{numerator}{denominator}` produces the proper fractions $\frac{a}{b}$. Occasionally you may find $a/b$ more suitable.

### 3.4.1 Making typing easier

A few things become very clear: There is a whole lot of syntax to remember; it is very easy to forget something and your `pdf` won't compile; and big expressions become very complicated to keep track. Let's highlight how we avoid these problems.

Inserting brackets with the appropriate sizing is made easy with snippets `@(`, `@[` and `@{`. Type whichever one you want and Intellisense should popup. Press `tab` and it will even place your cursor in the right spot. If you expect to put a lot inside those brackets, press `enter`, and use vertical space + identations to keep everything tidy.

Inserting greek letters in VSCode can be done by typing `@` followed by the first letter of the english name. Some special letters have a "variable" variant, like $\varphi$ (variable phi) compared to $\phi$ (phi). So $\gamma$ is produced with `@g`, `@f` $= \phi$ and `@vf` $= \varphi$.

Finally, to create the unnumbered `align*` environment, type `bsal`, then `tab` — **b**egin, **s** for unnumbered, **al**ign. Similar snippets exist for other environments, such as align — `bal`; `equation` — `beq`, `equation*` — `bseq`. A full list can be seen here.

Don't worry if this doesn't quite make sense. There are some exercises meant to practice exactly this.

### 3.4.2 Organisation

Let's look at a simple example

$$f(x) = \left( \frac{3+x}{x^{2x}} + \left[ \sqrt{x} + \sin(2x) \right] \right)^2$$
$$= \dots$$

The following code snippets describe two approaches to splitting up the expression. You have a lot of freedom to choose what works best for you, though the suggestion is to use something akin to the second example.

```
f(x) &
    = { \left( \frac{3+x}{x^{2x}} + \left[ \sqrt{x} + \sin(2x) \right] \right) }^2
\\
&= \dots
```

Compared to:

```
f(x) & = {
    \left(
        \frac{3+x}{x^{2x}}
        + \left[ \sqrt{x} + \sin(2x) \right]
    \right)
}^2
\\
&= \dots
```

### 3.4.3 Variables

It is also worth pointing out how to define LaTeX variables. The motivation is similar to how you would say "let $a = e^x + \sqrt{3}$" and use $a$ in the expressions.

**Defining a new variable** uses `\newcommand{\command}{expression}`. With our example, that means `\newcommand{\a}{e^x+\sqrt{3}}`. Now we can produce the same output by typing `\a`.

**Redefining a variable** uses `\renewcommand{}{}`. The point of separating the two is so that you don't accidently define/redefine a variable within the same *scope*. Variables in LaTeX are bound to their immediate environment. If this doesn't mean anything, just follow these two two rules:

1. If you need it for only this (set of) equation(s) in particular, define the variable inside the `align` environment.

2. If you need it across the whole document, set it in your preamble before the `document` environment.

For anything else in-between, reusing a variable name may require a little trial-and-error.

```
\begin{align} \label{eq:test}
\newcommand{\x}{\alpha+5}
    \x
\end{align}
```

Would produce the following result, and we can refer to it by number by using `\ref{eq:test}`.

$$\alpha + 5 \tag{3.1}$$

### 3.4.4 Vectors, matrices and cases

Vector notation is achieved using `\vec{F}`, producing $\vec{F}$. If you prefer bold instead of the arrow $\vec{F} \implies \mathbf{F}$, redefine the `\vec` variable in the preamble to math bold, like so: `\renewcommand{\vec}{\mathbf}`.

The `matrix` environment produces matrices without brackets, essentially the same as the `tabular` environment. For parenthesis we would use `pmatrix`; and for square brackets, `bmatrix`. The syntax is similar to tables/tabular we covered earlier — `&` separate columns and `\\` separate rows. Notice you will need to create a matrix environment inside of a math environment, like below

$$\vec{F} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}, \qquad A = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

```
\begin{align*}
    \vec{F} = \begin{bmatrix}
        1 & 2 & 3 \\
        3 & 2 & 1
    \end{bmatrix}
    , \qquad %Long space
    A = \begin{pmatrix}
        1 \\ 2
    \end{pmatrix}
\end{align*}
```

Another important environment which has similar syntax is `cases`, which looks like this:

$$f(x) = \begin{cases} 1, x \geq 5 \\ -1, x < 4 \\ 0, 4 \leq x < 5 \end{cases}$$

```
    \begin{align*}
        f(x) = \begin{cases}
            1, x \geq 5 \\
            -1, x < 4 \\
            0, 4 \leq x < 5
        \end{cases}
    \end{align*}
```

You will notice `\leq` and `\geq` for $\leq$ and $\geq$, respectively. They can be quickly produced with VSCode with the snippets `@<` and `@>`.

The last thing we will cover for maths symbols is calculus. A full list of symbols native available can be found here. Many of the most commonly used ones have snippets available in our editor, like `@8` $\implies$ `\infty` ($\infty$); `@2` $\implies$ `\sqrt{}` ($\sqrt{}$); `@-` $\implies$ `\bigcap` ($\bigcap$), and much more.

Finally, there are many options to how one can align and number equations, `amsmath` has excellent documentation explaining the various cases. See page 5 here.

### 3.4.5 Calculus

Let's typeset derivatives from first principle and integrals from riemann sums.

Try typesetting these yourself, including the organisation! You can find a solution in the `Calculus` example folder.

#### Derivatives

Given an expression $f(x) = x^2 + 5$, the derivative, $f'(x)$, from first principle is given by:

$$\begin{aligned} f'(x) &= \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x} \\ &= \lim_{\delta x \to 0} \frac{(x+\delta x)^2 + 5 - (x^2+5)}{\delta x} \\ &= \lim_{\delta x \to 0} \frac{x^2 + 2x\delta x + \delta x^2 + 5 - (x^2+5)}{\delta x} \\ &= \lim_{\delta x \to 0} \frac{\cancel{\delta x}(2x+\delta x)}{\cancel{\delta x}} \\ f'(x) &= 2x \end{aligned} \tag{3.2, 3.3}$$

—

The way these have been typeset in this document was to use an `align` environment (a numbered environment) and the `\nonumber` command before each new line I did not want numbered. All equations are aligned at the equals sign, which we achieve by setting the alignment anchor `&` before the `=`.

```
1    f'(x) & = ... \\
2    & = ... \\
```

As expected, $\delta x$ can be written with `\delta x`. Notice the space between both is important, seeing as `\deltax` is not a command itself.

The limit symbol comes from `\lim_{exp}`. The underscore `_` refers to a subscript, and you will see similar notation for integrals, sums and even besides brackets.

Finally, the `cancel` package provides us with the `\cancel{exp}` command, which puts a strike through `exp`. It also provides `\cancelto{value}{exp}`, which in a math environment generates $\cancel{x+5}^{0}$.

```
1  \cancelto{0}{x+5}
```

Once again, try to replicate it for yourself and see a solution in the `Calculus` example.

Our options for derivatives are straightforward:

$$\frac{d\ f(x)}{dx} \quad \frac{d^2}{dx^2}f(x) \quad \frac{\partial^3}{\partial^3 x}f(x,y)$$
$$\nabla^2 \vec{f} \qquad \nabla \cdot \vec{\omega} \qquad \nabla \times \vec{F}$$

```
\frac{d\ f(x)}{dx} \quad \frac{d^2 }{d x^2} f(x)
\quad \frac{\partial^3 }{\partial^3 x} f(x,y) \\
\nabla^2 \vec{f} \qquad \nabla \cdot \vec{\omega}
\qquad \nabla \times \vec{F}
```

It's worth mentioning that math environments ignore spaces. It automatically chooses an appropriate spacing after `+` or `=`, for instance, but sometimes you need to force a space. Small spaces are done with '`\ `' (note the empty space that follows the backslash), alternatively larger spaces are `\quad` and `\qquad`.

**Integrals**

Let $f : [a, b] \to \mathbb{R}$ be a function defined in the closed interval $[a, b]$ and with partitions

$$P = \{[x_0, x_1], [x_1, x_2], \dots [x_{n-1}, x_n]\}$$

such that

$$a = x_0 < x_1 < x_2 \dots x_n = b$$

A Riemann sum $S$ is defined as:

$$S = \sum_{i=1}^{n} f(x_i^*)\Delta x_i \tag{3.4}$$

Now if $f$ is integrable within the interval and $\Delta x_i$ approaches zero, we have an integral:

$$\int_a^b f(x)\,dx = \lim_{\Delta x_i \to 0} S = \lim_{\Delta x_i \to 0} \sum_{i=1}^{n} f(x_i^*)\Delta x_i \tag{3.5}$$

And finally, if $F(x)$ is the integral of $f(x)$, then

$$\int_a^b f(x)\,dx = F(x)\Big|_a^b = \Big[F(x)\Big]_a^b$$

—

**Note:** The `align` environment has a peculiar spacing between text and the math display area, which is larger than `equation`, the environment used for these examples.

The uppercase sigma sum symbol $\sum$ is done with `\sum`. Intuitively, if you need the pi product symbol $\prod$, that is `\prod`. As expected, `_` marks the text to go below and `^` the text that goes above. Therefore this is what was used:

```
\sum_{i=1}^n
```

Integrals use the `\int` tag, with similar sub/superscripts behaviour to sums. To guarantee a small space between the expression and $dx$, the suggestion is to use `\, dx`. The other variants of the integral symbol can be seen below.

$$\int f(x)\,dx \quad \iint dA \quad \iiint \oint$$

```
\int f(x) \,dx \quad \iint \,dA \quad \iiint \oint
```

The bar that indicates "evaluated in the interval $a$ to $b$", $\Big|_a^b$, is done by using `\Big|_a^b`. Note that both `\big` and `\Big` exist, each one referring to a slightly different length. A similar syntax can be used for other brackets.

```
F(x) \Big|_a^b = \Big[ F(x) \Big]_a^b = \Big\{ F(x) \Big|}_a^b
```

$$F(x)\Big|_a^b = \Big[F(x)\Big]_a^b = \Big\{F(x)\Big\}_a^b$$

### 3.4.6 Theorems, lemmas, corollaries, etc

While generally not used in an engineering context, you may find the creation of theorems, etc useful:

There is a very famous theorem,

**Theorem 1** *Angles of a triangle add up to* 180°.

which can be followed by a corollary

**Corollary 1.1** *The first corollary that follows*

More explanations followed by

**Corollary 1.2** *The second corollary*

**Theorem 2** *A second theorem*

To create each environment, the syntax is

```
\newtheorem{environment-name}{display}[numbered-after]
```

`environment-name` is a label. `display` refers to the actual text that will be shown — Theorem and Corollary, with the capitalisations. `numbered-after` is from where it should derive the numbering — `section`, another theorem, etc.

Then we use the environment like any other:

```
\begin{environment-name} ... \end{environment-name}
```

This can be placed at any point in your document, though most likely in your preamble.

To create the example above, we did the following:

```
\newtheorem{theorem}{Theorem}
\newtheorem{corollary}{Corollary}[theorem]
There is a very famous theorem,
\begin{theorem}
    Angles of a triangle add up to \ang{180}.
\end{theorem}
which can be followed by a corollary
\begin{corollary}
    The first corollary that follows
\end{corollary}
More explanations followed by
\begin{corollary}
    The second corollary
\end{corollary}
\begin{theorem}
    A second theorem
\end{theorem}
```

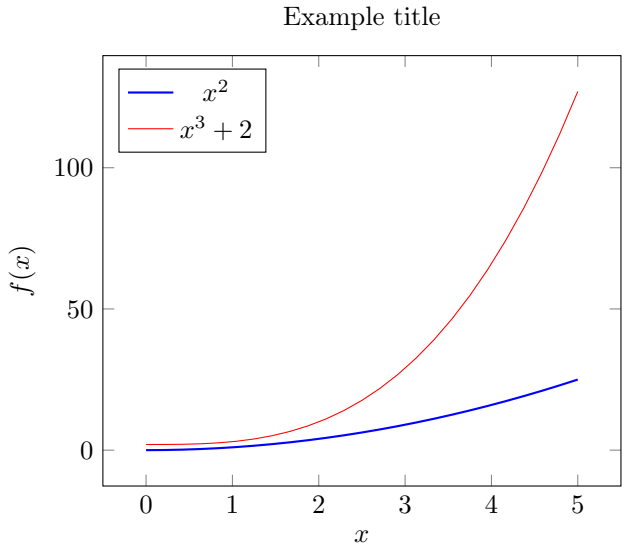
## 3.5 Graphs with Tikz and pgfplots

Generally graphing is done using the packages `tikz` and `pgfplots` inside of the `tikzpicture` environment. Generally figures require captions and you may need to refer to them, therefore we wrap `tikzpicture` with the usual `figure` environment.

The objective here is to read through the code below before seeing the results, and see if you know what to expect. The `pgfplots` package is extremely well documented with hundreds of examples, and very well structured tutorials. Take the examples highlighted here as graphs we do often, but it is highly recommended to skim through the documentation here.

`axis` is the most common axis environment, but we can also use `semilogxaxis`, `semilogyaxis` and `loglogaxis` for our logarithmic axis needs, `polaraxis`, and more. There are hundreds of options with names that tend to be self-evident, and the best approach is to search for examples and modify them to your needs.

```
    \begin{axis}[
        %options here separate by commas,
    ]
        %plots here separated by colons;
    \end{axis}
```



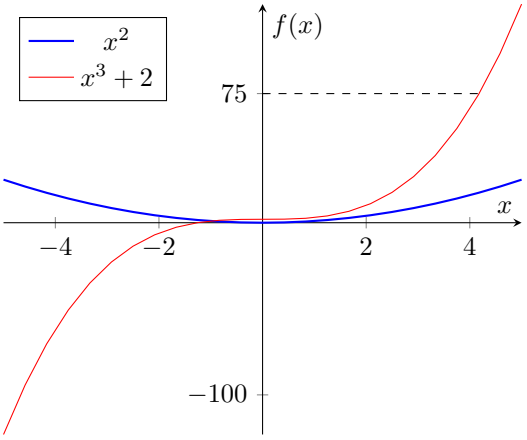

```
\begin{figure}[h]\centering
    \begin{tikzpicture}
        \begin{axis} [
            xlabel=\(x\) , ylabel=\(f(x)\),
                title=Example title,
            domain=0:5, legend pos = north west
                ,
            axis lines = left,
        ]
            \addplot[blue] {x^2};
            \addlegendentry{\(x^2\)}
            \addplot[red] {x^3+2};
            \addlegendentry{\(x^3+2\)}
        \end{axis}
    \end{tikzpicture}
    \caption{Example caption}
    \end{figure}
```

Figure 3.6: Example caption

Plotting a function is simply a matter of using `\addplot[options] {function};`. The options include things like colour, thickness, the domain for that specific plot and much more. Generally you will want to have the domain defined in the `axis` environment, not for each plot, but you will see plenty of examples where you do both.

We created legends by using `\addlegendentry{}`, but we also have the option of using `\legend{}` and describing them all in one place, as will be seen in the next example.

We have a lot of freedom with the frequency of the label ticks, whether you want them replaced with text, minimum and maximum displayed values. Changing the options of our plot to the following, and adding a dashed-line plot gives us the graph seen below.




```
\begin{tikzpicture}
    \begin{axis} [
        xlabel=\(x\), ylabel=\(f(x)\),
        legend pos = north west,
        domain=-5:5, axis lines = center,
        ytick={-100,0,75},
    ]
        \addplot[blue, thick] {x^2};
        \addlegendentry{\(x^2\)}
        \addplot[red] {x^3+2};
        \addlegendentry{\(x^3+2\)}
        \addplot[dashed, domain=0:4.18] {75};
    \end{axis}
\end{tikzpicture}
```

`\addplot[dashed, domain=0:4.18] {75};` is particularly interesting because it goes against the general recommendation. Here we are specifically creating an arbitrary dashed line at `y=75` in the domain `0:4.18`.

**Note:** Pay attention to the use of commas, colons and correctly creating a math environment. Not using them correctly is a very common reason why your document won't compile. `\addplot{x^2};` has the expression written normally and ends with a colon (`;`). On the other hand, `\addlegendentry{x^2}` is attempting to display `x^2`, but without math mode LaTeX doesn't know what to do — hence the `\(x^2\)`.

### 3.5.1 Bar plot

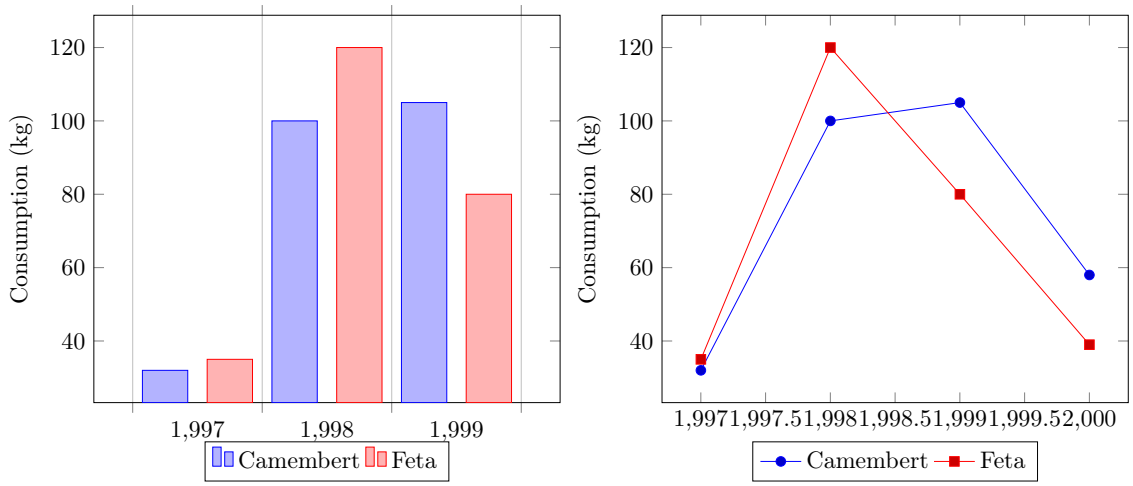



Figure 3.7: A bar plot and the same data without `ybar`.

```
\begin{tikzpicture}[scale=0.9]
    \begin{axis}[
        ybar interval=0.7,
        ylabel=Consumption (kg),
        legend style={
            at={(0.5,-0.1)}, anchor=north,legend columns=-1
        },
    ]
        \addplot coordinates {
        (1997,32) (1998,100) (1999,105) (2000,58)
        };
        \addplot coordinates {
            (1997,35) (1998,120) (1999,80) (2000,39)
        };
        \legend{Camembert, Feta}
    \end{axis}
\end{tikzpicture}
```

The option `ybar interval = 0.7` creates a bar plot in the y-direction with a bar thickness (in the x-direction) of 0.7. On the right of it you can see the exact same plot without the `ybar` option. You may notice the data points in the year 2000 which are omitted — this is also caused by `interval`. Play with it

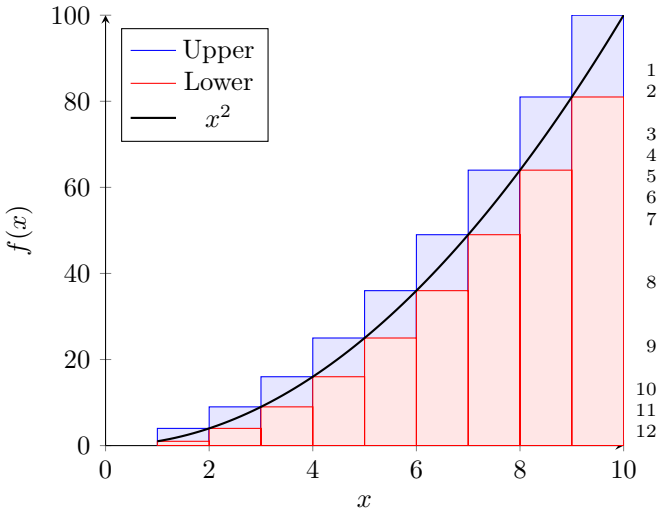
yourself, and take a look at the documentation examples for more customisation.

`\legend style = {...},` moves the legend from inside the plot box to the outside. Notice that in the previous example we simply used `legend pos`, but there is a lot that can be changed through styles.

This time around we created plots from coordinates, which are simply given as a list without commas, as highlighted below The main use-case tends to be simple bar plots, because usually we would import data from a file, as in our next example.

```
1    \addplot coordinates {
2        (x1,y1) (x2,y2) (x3,y3)...
3    };
```

**Challenge:** Can you combine the two to create a graphical representation of a Riemann sum? Below is one possible solution based on this thread in Stackexchange, an extremely important tool to finding solutions and learning!



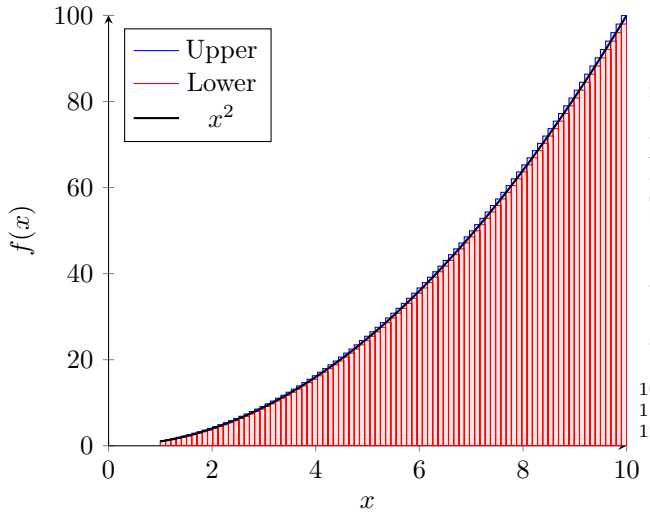

```
1  \begin{tikzpicture}
2      \begin{axis}[xlabel= \(x\), ylabel= \(f(x)
           \),
3              xmin=0, ymin=0,
4              axis lines = left, domain=0:10,
5              legend pos = north west,
6          ]
7          \addplot [draw=blue, fill=blue!10, ybar
               interval, samples=10, domain
               =10:1]{x^2}\closedcycle;
8          \addplot [draw=red, fill=red!10, ybar
               interval, samples=10, domain=1:10]
               {x^2}\closedcycle;
9          \addplot[smooth, thick,domain=1:10,
               samples=40]{x^2};
10         \legend{Upper, Lower, \( x^2 \) }
11     \end{axis}
12 \end{tikzpicture}
```

One important thing to note: the `draw` option controls the colour of the stroke/outline, while `fill` colours from the curve to the axis. Now if you want to colour the area between two curves, you may want to look at section 5.7 of the documentation or work through an example, like this one.

By increasing the number of `samples` (subdivisions), we can even see that it approaches the integral.

### 3.5.2 Plot from data

Data files are usually either `.csv` (with columns separated by commas), `.dat` or `.txt` (columns separated by tabs). Generally if copying directly from LibreOffice/Excel, columns will be tab separated; whilst Octave/Matlab data by default is comma separated.




```
\begin{tikzpicture}
    \begin{axis}[xlabel= \(x\), ylabel= \(f(x)
         \),
            xmin=0, ymin=0,
            axis lines = left, domain=0:10,
            legend pos = north west,
        ]
        \addplot [draw=blue, fill=blue!10, ybar
             interval, samples=90, domain
            =10:1]{x^2}\closedcycle;
        \addplot [draw=red, fill=red!10, ybar
            interval, samples=90, domain=1:10]
             {x^2}\closedcycle;
        \addplot[smooth, thick,domain=1:10,
            samples=40]{x^2};
        \legend{Upper, Lower, \( x^2 \) }
    \end{axis}
\end{tikzpicture}
```

Three columns of random data were generated using LibreOffice and can be found in the examples folder called `data.dat`. They can also be seen below.

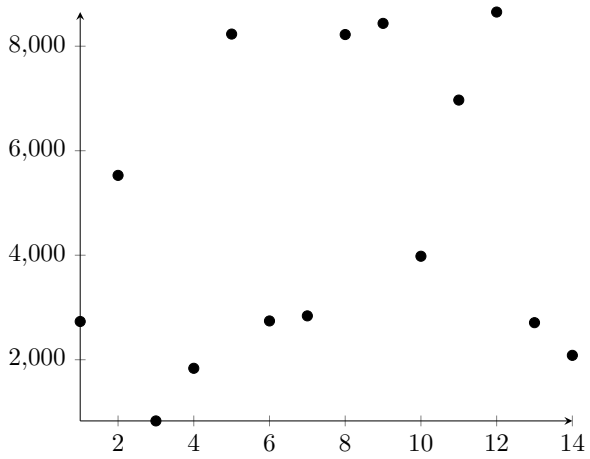
Listing 3.8: data.dat

```
Var1    Var2    Var3
1       2731.60851756311        13.2894415066652
2       5527.96064928581        23.0248555701667
3       828.88421543021 26.0238681538563
4       1835.9477498787 27.3949401553229
5       8232.23568836794        27.9277183310726
6       2742.10952781195        35.574656094906
7       2839.26593759326        32.6529698844285
8       8223.29229933961        35.585410610198
9       8436.78393620134        43.2400122448311
10      3980.50189677834        43.1362128057345
11      6969.0856706613 46.5513604389675
12      8653.93008312339        48.190253854583
13      2709.0352967109 57.5333844238011
14      2083.69661702771        59.2944882816431
```

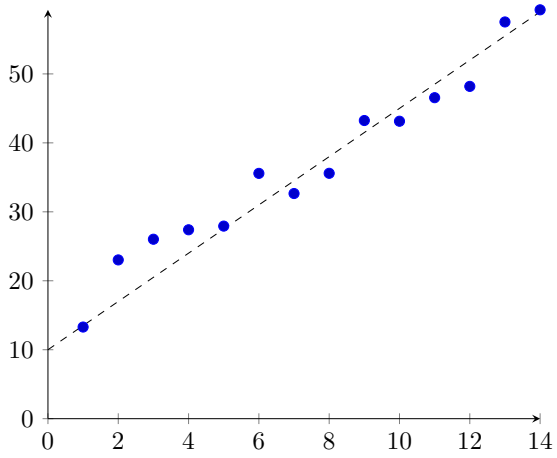
**Challenge:** Can you import this data in to a table using `pgfplotstable`? **Hint:** you may want to use these options: `precision = 3` and `fixed zerofill`.

By default, it picks the first two columns to represent x and y. In the next example, we deliberately choose the variables we want in each plot; allow automatic colouring `\addplot+`; and combine a function with the data in the same axis.

It's worth noting that `pgfplots` tries to graph with the least amount of empty space, but we can always specify the minimum value of x and y in the axis with `xmin` and `ymin`.




```
\begin{tikzpicture}
\begin{axis}[axis lines = left]
    \addplot [only marks] table {Examples/data.
        dat};
\end{axis}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\begin{axis}[ axis lines = left, ymin=0, ]
    \addplot+ [only marks] table[x=Var1, y=Var3
        ] {Examples/data.dat};
    \addplot [dashed, domain=0:14] {10+3.5*x};
\end{axis}
\end{tikzpicture}
```

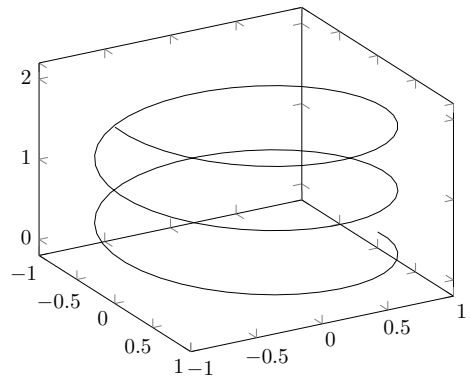
**Note:** It is possible to get `pgfplots` to generate a linear regression from your data, but realistically we process the data elsewhere, and use this only for display. If you would like to do it anyway, look at section 3.3 of the `pgfplots`' documentation here.

### 3.5.3 3d plots

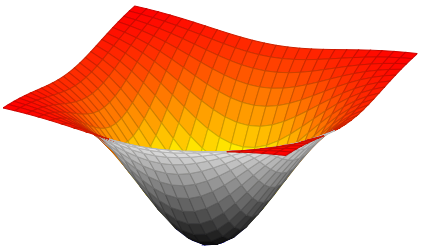
It is possible to do 3d plots with `pgfplots`, but beyond anything simple, you will find that using matlab and exporting the graph is far more computationally efficient[1]. Nevertheless, we will present a couple of examples here and go more in-depth with importing graphs in the following section.

Parametised plots are the simplest to get started, but pay attention to the use of `x` in all directions. Note that these examples are taken from `pgfplots`' documentation.

[1] Alternatively, you may want to look into `tikz-3dplot`.




```
\begin{tikzpicture}
    \begin{axis}[view={60}{30}]
        \addplot3 [domain=0:5*pi,samples=100,
            samples y=1,]
        (
            {sin(deg(x))}, {cos(deg(x))}, {2*x
                /(5*pi)}
        );
    \end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
    \begin{axis}[hide axis,mesh/interior
        colormap name=hot,colormap/blackwhite
        ,]
        \addplot3 [domain=-1.5:1.5,surf] {-exp
            (-x^2-y^2)};
    \end{axis}
\end{tikzpicture}
```

## 3.6 Importing graphs from Matlab

### 3.6.1 Importing bitmap graph

The simplest method to import get your Octave/Matlab graphs onto a document is to export them to a bitmap picture, and import like we did before.

Our code looks something like the following[2], and the full example is in `Examples/Importing-graphs`.

Listing 3.9: `Importing-graphs/bitmap/example.m`

```
x=linspace(0,2*pi);
y=sin(x);
plot(x,y);
xlabel('x'); ylabel('f(x)');
legend('f(x)=sin(x)', 'Location', 'best');
print('figures/example.eps','-depsc')
```

Then, to include the graph, we simply use `\includegraphics`.

```
\begin{figure}[h]
    \centering
    \includegraphics[width=\textwidth]{figures/example.eps}
\end{figure}
```

[2]You may notice the figure is exported to `.eps`, not `.png`, `.tiff` or something otherwise more common. For that you may want to look into `export_fig`.

Below you can see the output compared to something natively drawn with pgfplots. It is a deliberately simple graph that can be easily replicated natively, but it highlights that font sizes and other small issues can become problematic for legibility.

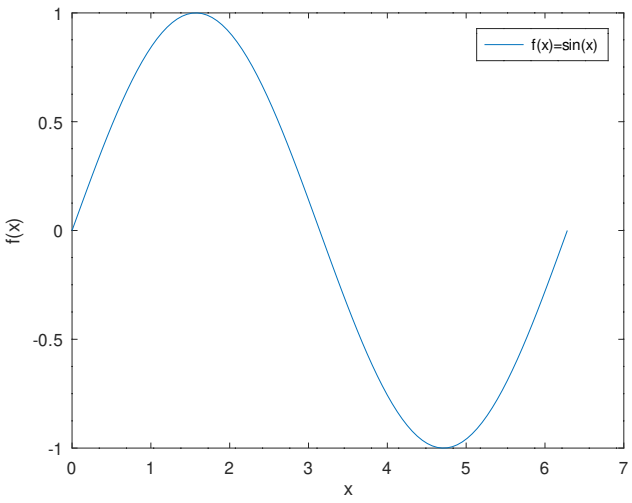



Figure 3.8: Matlab

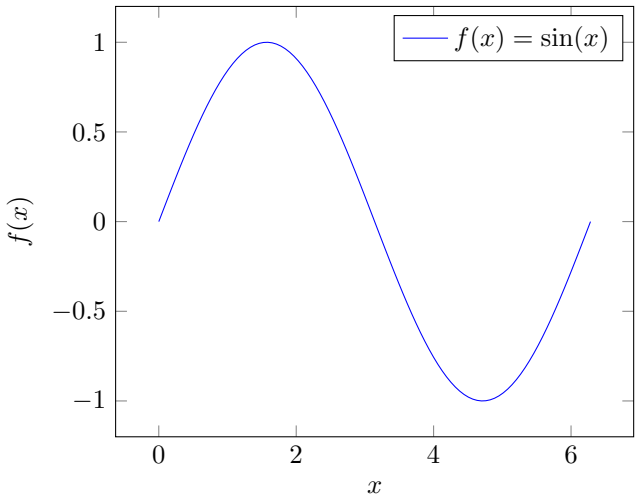



Figure 3.9: Natively rendered

So what can we do instead? We can generate the graphs in Octave/Matlab and export the data for `tikz` and `pgfplots` to natively render it. We have two options to do so: `matlab2tikz` and exporting to a `.svg` file.

**Challenge:** Replicate the sinusoidal graph with `pgfplots`, and you might see a straight line, indicating that it uses degrees, not radians. Try to search for the solution! Hopefully you will run into this link.

### 3.6.2 Matlab2tikz

This is an outstanding script written by Nico Schlömer which can be downloaded from Mathworks' file exchange, or directly from the github. Please refer to the installation on the github page, if you are unfamiliar with installing external matlab functions. In short, use Set Path, `addpath` or see Figure 3.6.2 to include the downloaded `src/` to your path.

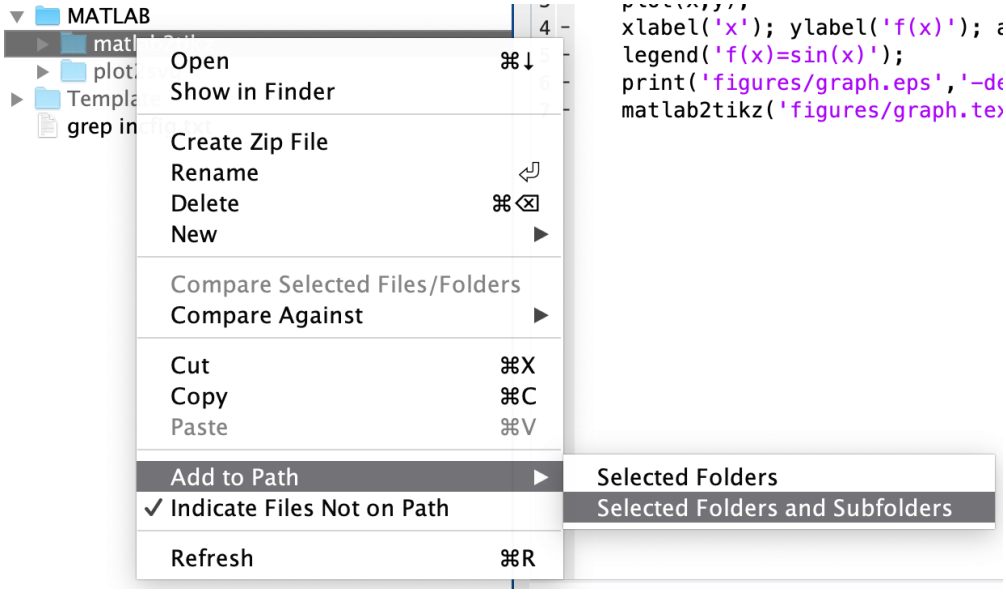



Figure 3.10: Adding external function to Matlab

Usage is very similar to exporting a bitmap image, the only difference is that we get a `.tex` to import, which can be further edited if you really want to:

Listing 3.10: `Importing-graphs/matlab2tikz/mat2tikz.m`

```
x=linspace(0,2*pi);
y=sin(x);
plot(x,y);
xlabel('x'); ylabel('f(x)');
legend('f(x)=sin(x)', 'Location', 'best');
matlab2tikz('figures/graph.tex', 'height', '\fheight', 'width', '\fwidth');
```

Importing requires a few packages and settings, as can be seen below. The information is extracted from `matlab2tikz`'s github, and reading their FAQ may be very helpful

The code is not separated into preamble and content, but it should be clear what goes where. Importantly, the whole example can be found in `Examples/Importing-graphs`.

Listing 3.11: `Importing-graphs/matlab2tikz/totikz.tex`

```
\documentclass{article}
\usepackage{pgfplots}
\pgfplotsset{compat=newest}
\usetikzlibrary{plotmarks}
\usetikzlibrary{arrows.meta}
\usepgfplotslibrary{patchplots}
\usepackage{amsmath}
\pgfplotsset{plot coordinates/math parser=false}
\begin{document}
```

```
\begin{figure}[h] \centering
    \newlength\fheight{} \newlength\fwidth{}
    \setlength\fheight{8cm}
    \setlength\fwidth{12cm}
    \input{figures/graph.tex}
\end{figure}
\end{document}
```

Resulting in:

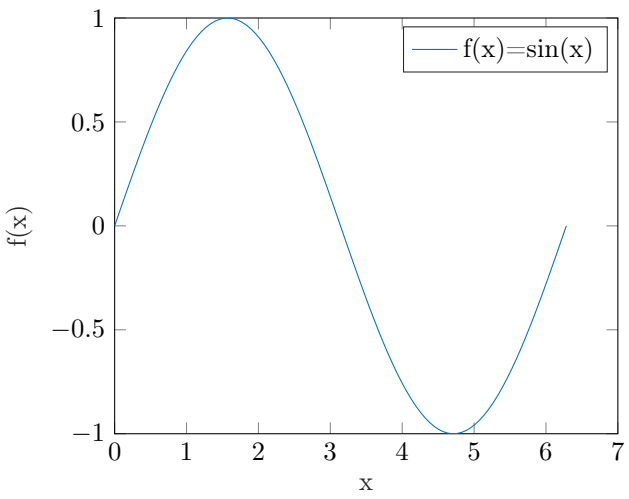



Figure 3.11: Matlab generated and rendered natively.

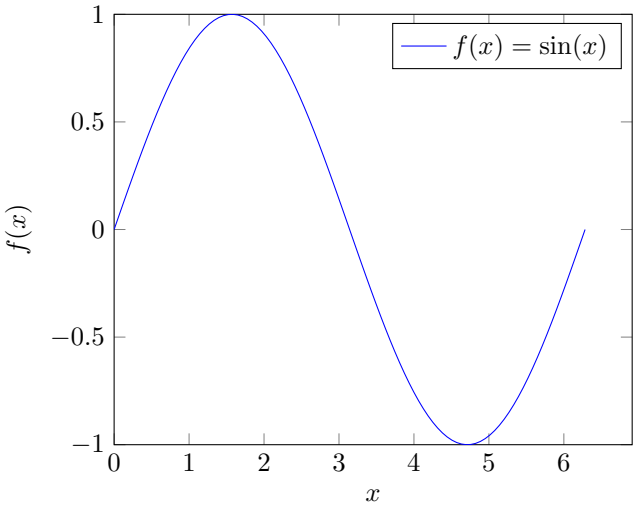



Figure 3.12: Generated with `pgfplots`.

**Note:** Do you know why the fonts look still different? In the graph generated by matlab, we did not tell it to render using math mode, i.e. f(x) vs $f(x)$.

### 3.6.3 Importing vector graph

Occasionally the graphs produced in Octave/Matlab are complicated enough that they don't render very well with `matlab2tikz`. In those cases, the plot is exported to a scalable vector format (`.svg`), then Inkscape produces a special `.pdf` which gets imported into our document. What is special about this `.pdf` is that the fonts are rendered natively in LaTeX.

Briefly about Inkscape: it's a free and open source vector editor, serving a similar purpose to Adobe Illustrator, CorelDRAW, etc. You might find that it serves all your vector needs, and it's definitely worth exploring. For our purposes, you can get away with simply using it to Save As `.pdf` and small tweaks to the graph.

The tool we will use is plot2svg developed by Juerg Schwizer. Installation is the same as before —

download plot2svg, then add the `src/` to your path, and you're ready.

### Generating the graph

The example in full is available in `Examples/Importing-graphs/vector`. With the code below, we produce both a `.svg` and a `.tex` to be imported.

Listing 3.12: `Importing-graphs/vector/surface.m`

```
[x,y] = meshgrid(-pi/4:0.05:pi/4);
z=tan(x.*y);
L = 4*del2(z,0.05);
figure,surf(x,y,L);
xlabel('x'); ylabel('y'); zlabel('z');
title('Laplacian␣of␣\(z=\tan(xy)\)');
plot2svg('figures/laplacian.svg');
matlab2tikz('figures/laplacian.tex', 'height', '\fheight', 'width', '\fwidth');
```

It's worth highlighting that you can get matlab to interpret LaTeX syntax with the options shown above. This is great if you're either exporting the bitmap image or the `.tex` file. With `.pdf_tex`, you cannot have matlab interpret the latex, and therefore need to play around with getting math mode using `\( \)`.

### Converting to `.pdf_tex`

Now we can open the `.svg` in inkscape, Save As `.pdf` and choose to "omit text in PDF and create LaTeX file". Generally you might also need to tick "Use exported object's size".

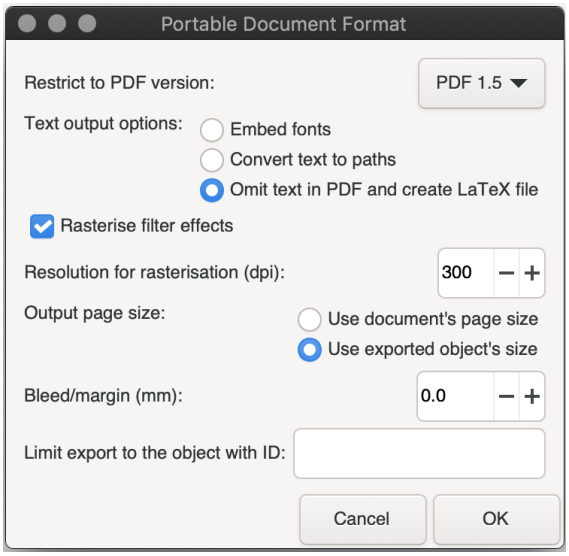



Figure 3.13: Saving options for `.tex_pdf`

Bear in mind that from this step and forward, it is the exact process for importing any vector figure you draw with inkscape.

### Importing to LaTeX

Gilles Castel created a really concise solution to importing, and we will take advantage of it:

```
\usepackage{import}
\usepackage{pdfpages}
\usepackage{transparent}
\usepackage{xcolor}
\newcommand{\incfig}[2][1]{%
    \def\svgwidth{#1\columnwidth}
    \import{./figures/}{#2.pdf_tex}
}
\begin{document}
    \begin{figure}[h]\centering
        \incfig{laplacian}
        \caption{From \texttt{plot2svg}.}
    \end{figure}
\end{document}
```

Resulting in the following graph.

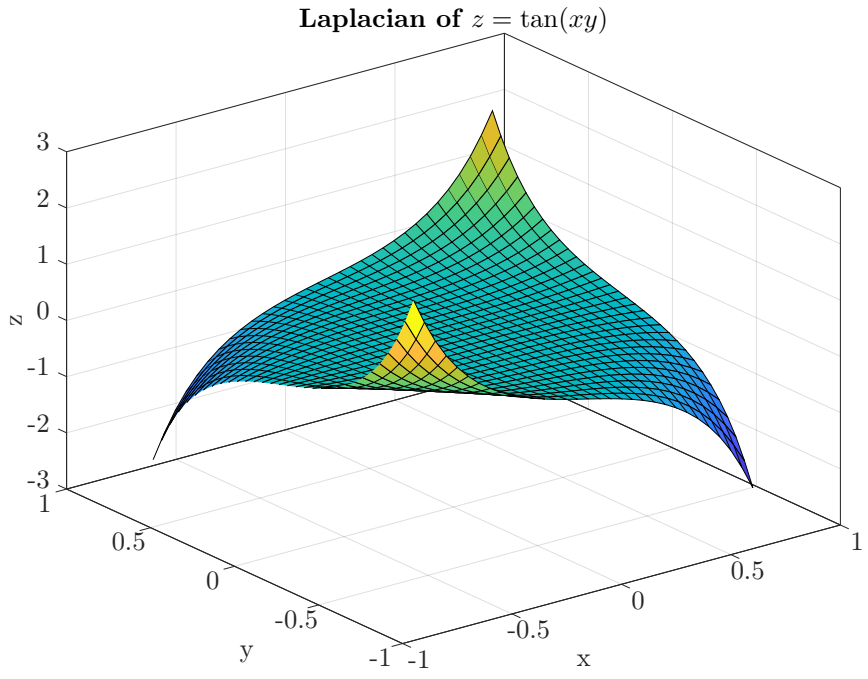



Figure 3.14: From `plot2svg`.

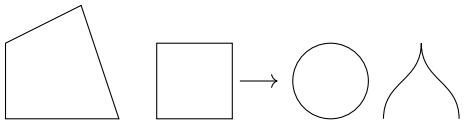
**Challenge:** Import the one generated by `matlab2tikz` into your document, and see how they compare. You can see the results in `Examples/Importing-graphs/vector`. In this case there is no difference between the two, so it may come down to preference or situations where your graph is not displayed how it should.

## 3.7 Drawing with Tikz

For this section we will create a very simple control system and generally introduce you to drawing with `tikz`. Some things are better drawn programmatically, while others are more easily done with vector drawing (for example, using inkscape). Understanding how `tikz` works allows you to make drawing considerably easier, and even if you prefer to draw blocks elsewhere, the skills will be useful in drawing circuits.

### 3.7.1 Simple shapes

Let's start by creating some simple shapes.



```
\begin{tikzpicture}
    \draw (0,0) -- (1.5,0) -- (1,1.5) -- (0,1)
        -- cycle;
    \draw (2,0) rectangle (3,1);
    \draw (4.3,0.5) circle (0.5cm);
    \draw [->] (3.1,0.5) -- (3.6,0.5);
    \draw (5,0) .. controls (5,0.5) and
        (5.5,0.5) .. (5.5,1)
        .. controls (5.5,0.5) and (6,0.5)
            ..(6,0);
\end{tikzpicture}
```

Most drawings begins with `\draw (x0,y0);` which decides the very first point of your drawing. From there you can assign coordinates to link lines, or use keywords that create circles, ellipses, arcs, rectangles, etc, each with their own special syntax. The syntax for `rectangle` uses the position of two opposite vertices: `(3,0) rectangle (4,1)`. On the other hand, `circle` takes the coordinates for the centre and a radius in either `cm`, `in`, `pt` or `em`. If you are not quite sure what these units mean, just stick to `cm`.

The creation of arrows is just as straightforward, with using the option `[->]` or `[<-]` to indicate the orientation of the arrow. Curved paths have a far more complicated syntax, using `controls`:

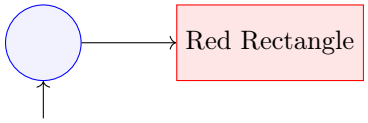
```
(starting coordinate) .. controls (first control point) and (second control point) .. (end point)
```

Now imagine when you have to change the position of one of the drawings. What a nightmare! For this reason, we have access to nodes — portions of a picture that are given a position, shape and other options. A `node` can be created at any coordinate in your drawing. One approach will be demonstrated now and another in the next section. The basic syntax is the following:

```
    \node [options] (name) at (x,y) {text in shape};
```


```
\begin{tikzpicture}
    \node [circle, draw=blue, fill=blue!5, minimum size=1cm,]
        (blueCircle) at (0,0) {};
    \node[rectangle, draw=red, fill=red!10, minimum height=1cm
        , minimum width = 2cm] () at (3,0) {Red Rectangle};
\end{tikzpicture}
```

And not only can we use the names of the nodes place of coordinates, but we have access to both absolute and relative coordinates.




```
\begin{tikzpicture}
    \node [circle, draw=blue, fill=blue!5, minimum size=1cm,]
        (blueCircle) at (0,0) {};
    \node[rectangle, draw=red, fill=red!10, minimum height=1cm
        , minimum width = 2cm] (redRectangle) at (3,0) {Red
        Rectangle};
    \draw[->] (blueCircle) -- (redRectangle);
    \draw[->] (blueCircle) ++ (0,-1) -- (blueCircle);
\end{tikzpicture}
```

### 3.7.2 Control system

For this upcoming section, please take a look at `Examples/Control-systems` for the examples in full.

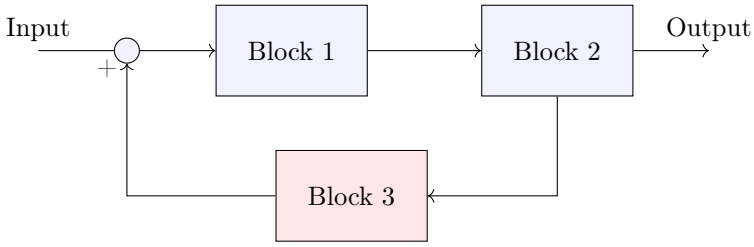



Figure 3.15: Simple control system

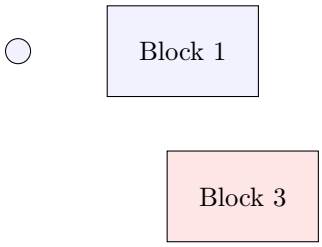
You are probably able to replicate this exact block, but having to repeat all of those instructions for several blocks that are identical would be tedious. Gladly, `tikzstyle` as well as some tikz libraries give us some special features to both eliminate repetition and be very precise with our arrow and text positioning.

```
\usepackage{tikz}
\usetikzlibrary{positioning,shapes,arrows}

%Control blocks shortcuts
\tikzstyle{block} = [draw, minimum width = 2cm, minimum height = 1.2cm, fill=blue!5]
\tikzstyle{sum} = [draw, fill=blue!5, circle, node distance=1cm]
```

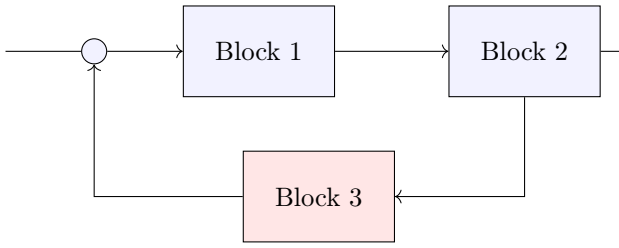
**Note:** You can have a `tikzstyle` either inside one tikzpicture environment or for your entire document. These are used commonly enough that the suggestion is to have them as a part of your entire document, as is the case in the templates.

Now each block can be created by using the option `block`, instead of describing its minimum height, width and colour. But we still have the option to overwrite the "default" colour if we want to. Pay attention to absolute positioning on line 3, compared to relative positioning, as seen in lines 4, 5 and 6.



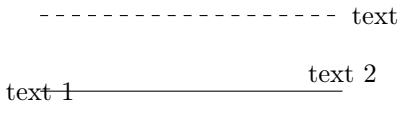

```
1 \begin{tikzpicture}
2     %Blocks
3     \node [block] (block1) at (0,0) {Block 1};
4     \node [block, right = 1.5 cm of block1,] (block2) {
          Block 2};
5     \node [block, below left = 1 cm of block2, fill=red
          !10] (block3) {Block 3} ;
6     \node [sum, left = of block1] (sum1) {};
7 \end{tikzpicture}
```

Now we can add the arrows completely separately from the shapes. And you will notice `|-` and `-|`, which create a 90° bend without the need to manually set the points.




```
1  \begin{tikzpicture}
2  %Blocks
3  \node [block] (block1) at (0,0) {Block 1};
4  \node [block, right = 1.5 cm of block1,] (
       block2) {Block 2};
5  \node [block, below left = 1 cm of block2, fill
       =red!10] (block3) {Block 3} ;
6  \node [sum, left = of block1] (sum1) {};
7  %Arrows
8  \draw[->] (block1) -- (block2);
9  \draw[->] (block2.east) -- ++(1.0,0);
10 \draw[->] (block2.south) |- (block3.east) ;
11 \draw[->] (block3.west) -| (sum1) ;
12 \draw[->] (sum1.west) ++ (-1,0) -- (sum1.west)
13     (sum1.east) -- (block1.west)
14 ;
15 \end{tikzpicture}
```

Adding text in drawings requires a node, but gladly there is another way to create them. Following a set of coordinates, we can simply start a node, like so:

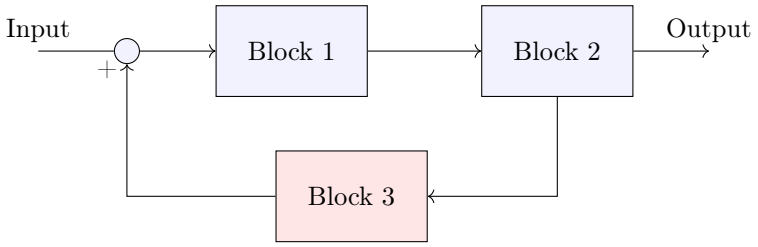



```
1 \begin{tikzpicture}
2     \draw[dashed] (0,1) -- (4,1) node[right](){
          text};
3     \draw (0,0) node() {text 1} -- (4,0) node[
          above]() {text 2};
4 \end{tikzpicture}
```

Finally we can combine all of this to create our control system.

```
1 \begin{tikzpicture}
2     %Blocks
3     \node [block] (block1) at (0,0) {Block 1};
4     \node [block, right = 1.5 cm of block1,] (block2) {Block 2};
5     \node [block, below left = 1 cm of block2, fill=red!10] (block3) {Block 3} ;
6     \node [sum, left = of block1] (sum1) {};
7     %Arrows
8     \draw[->] (block1) -- (block2);
9     \draw[->] (block2.east) -- ++(1.0,0) node [right,above](){Output};
```

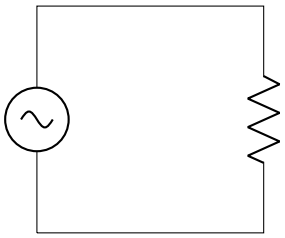
```
    \draw[->] (block2.south) |- (block3.east) ;
    \draw[->] (block3.west) -| (sum1) node[below left](){+};
    \draw[->] (sum1.west) ++ (-1,0) node[above](){Input}-- (sum1.west)
        (sum1.east) -- (block1.west)
    ;
\end{tikzpicture}
```




## 3.8 Circuits

We draw circuits using the `circuitikz` package. All of the knowledge from `Tikz` will come in handy, with the benefit that electrical components have already been created and given names. The documentation has a component list, and you can find the name for the exact component you need.

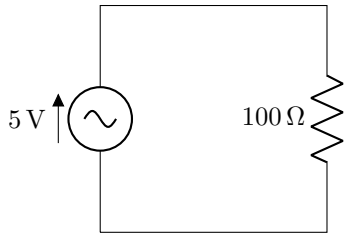
Let's create our first circuit:



```
\usepackage{circuitikz}
\begin{document}
    \begin{circuitikz}
        \draw (0,0) to[sV] (0,3) -- (3,3) to[R] (3,0) -- (0,0)
            ;
    \end{circuitikz}
\end{document}
```

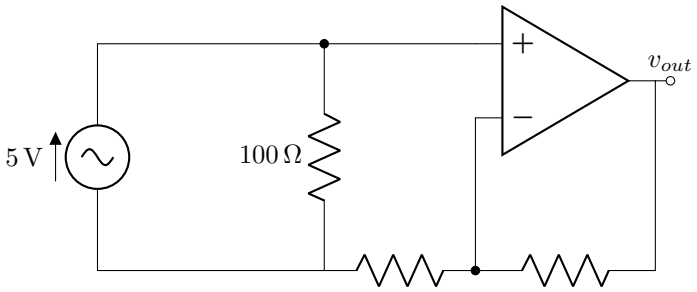
To add values with units, the suggestion is to use the package `siunitx`. In fact, this is recommended across any document. Basic usage is `\SI{value}{unit}`, and we have the option to either describe the units in text or simply using a symbol: `\ohm` or `ml`.

You have a couple of options with how to produce the label, `[element-name=label]`, or `[element-name, a=label]`. It automatically chooses a good position, but it can be tweaked with `a=` for one side, and `l=` on the other.




```
\usepackage{circutikz}
\usepackage{siunitx}
\begin{document}
    \begin{circuitikz}
        \draw (0,0) to[sV=\SI{5}{\volt}] (0,3) -- (3,3) to[R=\
            SI{100}{\ohm}] (3,0) -- (0,0);
    \end{circuitikz}
\end{document}
```

Next let's add an amplifier. By default the inverting input is on the top, hence the option `noinv input up`. Remember, `++ (x,y)` creates relative coordinates, so `(1,1) -- ++ (2,0)` means a line from (1,1) to (3,1)




```
\begin{circuitikz}
    \draw (0,0) to[sV=\SI{5}{V}] (0,3) -- (3,3) to[R, a=\SI{100}{\ohm}] (3,0) -- (0,0);
    \draw (3,3) to[short,*-] ++ (2,0) node [op amp, noinv input up, anchor=+](opamp){};
    \draw (3,0) to[R,-*] (opamp.- |- 3,0) coordinate (intersection) to[R] (opamp.out |- intersection) --
        (opamp.out) to[short,-o] ++ (0.2,0) node[above]{ \( v_{out} \) };
    \draw (opamp.-) -- (intersection);
\end{circuitikz}
```

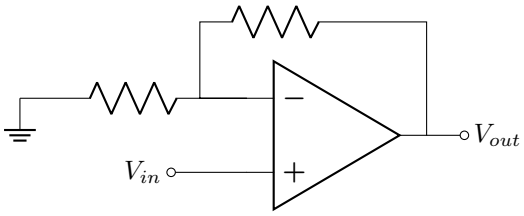
The usage of `|-` and `-|` may take some getting used to. `(opamp.out |- intersection)` means "the coordinates for the intersection between `opamp.out` and `intersection`". However, `(0,0) |- (1,1)` creates the lines `(0,0) -- (0,1) -- (1,1)`. On the other hand, `(0,0) |- (1,1)` creates the line `(0,0) -- (1,0) -- (1,1)`.

In the same way that you can create nodes at any point to include text or an image (in this case, the amplifier), you can also give the coordinates a name. Bear in you mind you can also create the amplifier with `\node`, as per the example below.

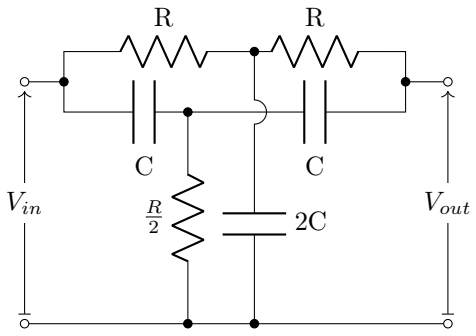
It may have been clear from previous examples, but you don't need to create separate `\draw`. The motivation to do so is tidiness and legibility.

Below you can see an example with only absolute coordinates.

It's quite easy to get lost with complicated circuits, so the suggestion is to think about it in terms of a continuous line drawing. You will notice this circuit draws the horizontal lines across the the resistors, then across the capacitors, then the vertical lines.




```
\begin{circuitikz}
\node [op amp] (opamp) at (0,0) {};
\draw
    (opamp.out) |- ++ (-1,1.5) to[R] ++ (-2,0)
        |- (opamp.-) to [R] ++ (-3,0) node[
        ground]{}
    (opamp.+) to [short,-o] ++ (-1,0) node[left
        ]{$V_{in}$}
;
\draw (opamp.out) to[short,-o] ++(0.5,0) node[
    right]{$V_{out}$};
\end{circuitikz}
```




```
\begin{circuitikz}
    \draw (0,0.5) to[short, o-*] (0.65,0.5) |- (1.3,1) to
        [R=R] (3.3,1) to[R=R] (6.3,1) -- (6.3,0.5) to[
        short, *-o] (7,0.5);
    \draw (0.65,0.5) -- (0.65,0) to[C,a=C] (3.3,0) to[C,a
        =C] (6.3,0) -- (6.3,0.5);
    \draw (0,-3.5) to[short,o-o] (7,-3.5); %ground
    \draw (2.7,0) to[resistor, a=$\frac{R}{2}$, *-*]
        (2.7,-3.5);
    \draw (3.8,-3.5) node[circ]{} to[C, a=2C] (3.8,-0.2)
        -- ++ (0,0) arc (-90:90:0.2) -- (3.8,1) node[circ
        ]{};
    \draw [|->] (0,-3.35) -- (0,0.35) node[midway,fill=
        white]{$V_{in}$};
    \draw [|->] (7,-3.35) -- (7,0.35) node[midway,fill=
        white]{$V_{out}$};
\end{circuitikz}
```

## 3.9 Packages worth exploring

### 3.9.1 SI Units

The package `siunitx` was briefly mentioned when we discussed circuits. Generally this is the go-to package for proper typesetting of units both in text and math modes.

$\mu g\,s^{-1}$ and 1.25 kJ
Some are made easier: 5 kg to 10 kg, µg

```
\si{\micro\gram\per\second} and \SI{1.25}{\kilo
    \joule}

Some are made easier: \SIrange{5}{10}{\kg}, \si
    {\ug}
```

### 3.9.2 Chemistry

#### Formulae

The package mhchem is the go-to for chemical formulae with very simple syntax:

$$\mathrm{H_2CO_3} \rightleftharpoons 2\,\mathrm{H^+(aq)} + \mathrm{CO_3^{2-}} + (1/2)\,\mathrm{H^\bullet}$$
$$\mathrm{H_2} + {}^{227}_{90}\mathrm{Th^+} \xrightarrow{\Delta} \mathrm{NO}_x$$
A triple bond is $\mathrm{H-C{\equiv}CH}$

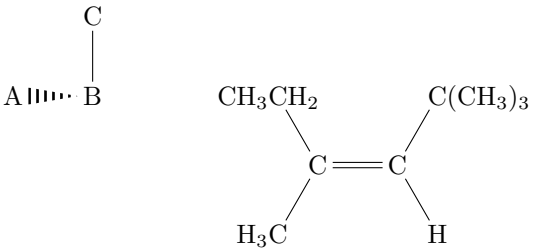
```
\usepackage{mhchem}
\begin{document}
\ce{H2CO3 <=> 2H+(aq) + CO3^{2-} + (1/2) H^{.}}

\ce{H_2 + ^{227}_{90}Th+ ->[\Delta] NO_x}

A triple bond is \ce{H-C#CH}
\end{document}
```

#### Geometry

For drawing geometry, the package used is chemfig. It is extremely powerful, and the documentation is worth exploring if you need to typeset geometry. A few examples will be shown below.




```
\usepackage{chemfig}
\begin{document}
    \chemfig{A>:B-[2]C}
    \qquad \qquad
    \chemfig{CH_3CH_2-[:-60,,3]C(-[:-120]H_3C)=
        C(-[:-60]H)-[:60]C|{(CH_3)_3}}
\end{document}
```

# Finding resources

Generally, finding more learning resources can be broken down into having a particular problem you are trying to solve, and learning what is possible with a package.

## 4.1 Specific problems

It is very likely someone else had the same exact problem you had, so use your preferred search engine! Invariably you will run into Stack Exchange, and adapting these solutions to your exact needs is a really important skill.

Some times you will find particularly well detailed solutions, like this one explaining how to produce beautiful tables. Most likely, however, you will find a lot of options and commands you may not understand, like this. In that case, the package documentation is key.

## 4.2 Package information

If you know what you are looking for, package documentation can be found on CTAN. And if you are using VSCode, clicking on the package's name will give you a link to the documentation:

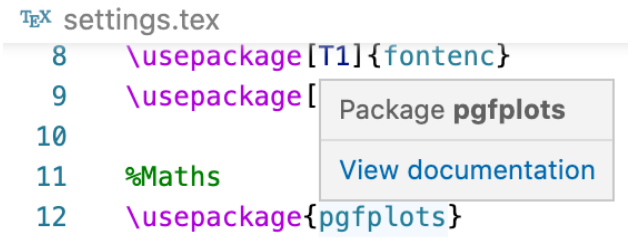



Figure 4.1: Viewing package documentation from inside VSCode

You may find documentation that includes examples, such as `pgfplots`, or general explanations about best practices, like `booktabs`. For the latter, it is likely someone has detailed explanations of a use-case in Stack Exchange.

# Working faster

## 5.1 Becoming familiar with the IDE

## 5.2 Using snippets