

Contents

1	Introduction	3
1.1	How it's different from Word	3
1.1.1	WYSIWYG - What You See Is What You Get	3
1.1.2	Typesetting	4
1.1.3	Markup language	4
1.1.4	Packages	5
1.1.5	Compilation	5
2	Installation	6
2.1	Installing L ^A T _E X	6
2.2	Installing an IDE	6
3	Getting started	7
3.1	My first document	7
3.1.1	Paragraph	8
3.1.2	Sectioning	8
3.1.3	Bold, italic, underline, etc	9
3.2	Bibliography management	10
3.2.1	Creating a bibliography	10
3.2.2	Manually adding a bibliography entry	11
3.2.3	Citations	11
3.3	Environments	13
3.3.1	Staying Organised	13
3.3.2	Figure and caption	15
3.3.3	Label and cross-reference	16

3.3.4	Table	17
3.3.5	Lists	19
3.3.6	Code	20
3.4	Maths	21
3.4.1	Making it easier	22
3.4.2	Organisation	22
3.4.3	Variables	23
3.4.4	Vectors, matrices and cases	23
3.4.5	Calculus	24
3.5	Graphs with Tikz	27
3.5.1	Bar plot	29
3.5.2	Plot from data	30
3.5.3	3d plots	31
3.6	Circuits	32
3.7	Control systems	32
4	How do I...?	33
4.1	Search engine	33
4.2	Stack Exchange	33
4.3	CTAN package information	33
5	Working faster	34
5.1	Becoming familiar with the IDE	34
5.2	Using snippets	34

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 L^AT_EX, 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 L^AT_EX:

Listing 1.1: Example document written with L^AT_EX

```
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` (greater or **e**qual) and `\int` (*f*). 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 L^AT_EX

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:

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. \LaTeX is 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	1
1.1 Counted subheader	1
2 Second header	1

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 bibliography

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:

```
1 @article{ GerberLeahR2005, %Unique identifier
2   author   = {Gerber, Leah R and Beger, Maria and McCarthy, Michael A and Possingham, Hugh P},
3   title    = {A theory for optimal monitoring of marine reserves},
4   issn     = {1461-023X},
5   journal  = {Ecology letters},
6   pages    = {829--837},
7   volume   = {8},
8   publisher = {Blackwell Science Ltd},
9   number   = {8},
10  year      = {2005},
11  edition   = {Editor, Ransom Myers Manuscript received 15 March 2005 First decision made 21 April 2005
               Manuscript accepted 6 May 2005},
12 },
```

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:

```
1 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!

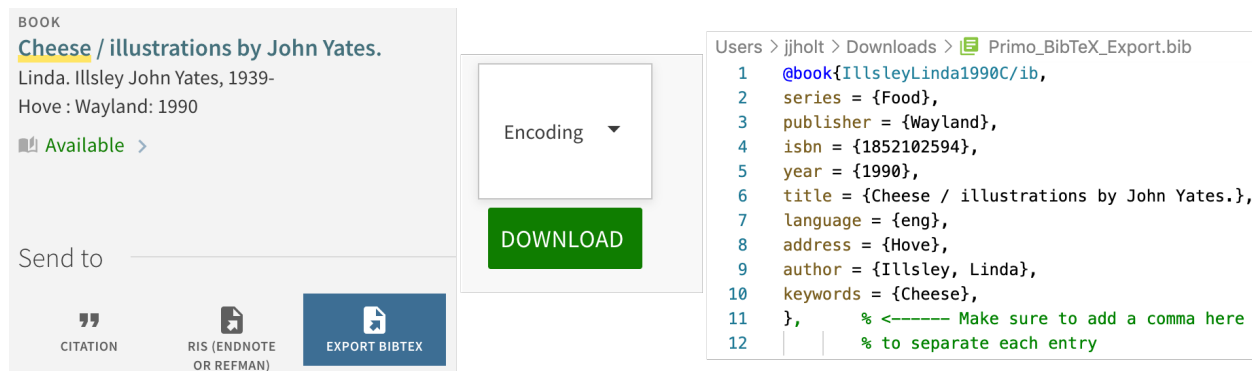


Figure 3.1: Getting a BiBTeX file from UCL's Library. From here, just paste to your .bib file.

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 bibliography entry.

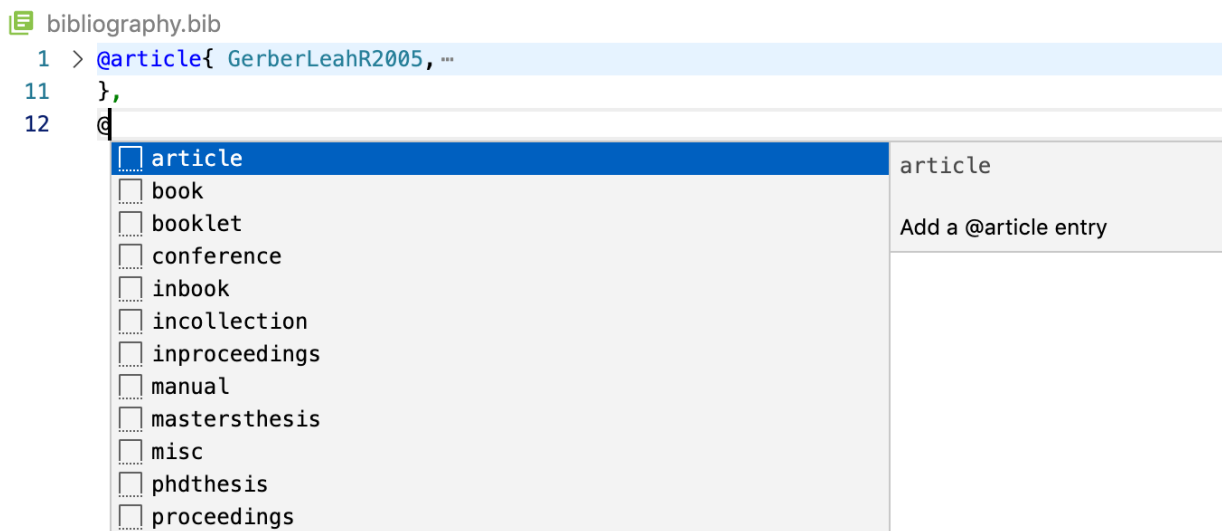


Figure 3.2: Autocompleting a bibliography entry in a .bib file.

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

```
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:

Citations are made so easy with L^AT_EX, 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.

```
1 \usepackage[round]{natbib}
2 ...
3 \citep{GerberLeahR2005}
4 ...
5 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 enormous list.

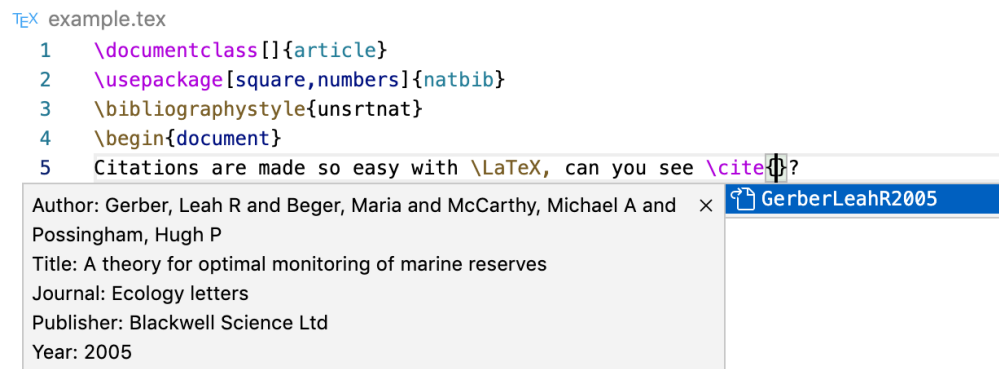


Figure 3.3: Autocomplete from our bibliography file.

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.

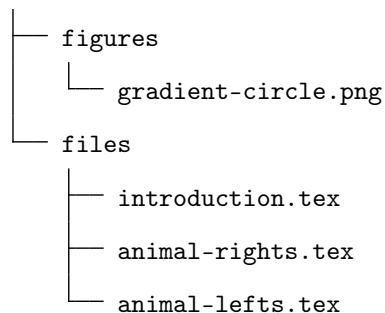
Let's expand our working directory like seen below. You can also just copy the files from `Example2`.

Example2

```

|— bibliography.bib
|— cheese-prices.dat
|— main.pdf
|— main.tex
|— code
|   |— code.m

```



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}
9 \lstset{
10     numbers=left, frame=single, breaklines=true, %Keep text inside a frame, and number each line.
11     basicstyle = \scriptsize\ttfamily, %smaller size, monospaced
12 }
13
14 \begin{document}
15 \section{Introduction}
16     \subimport{files/}{introduction.tex}
17 \section{Animal Rights}
18     \subimport{files/}{animal-rights.tex}
19 \section{Animal Lefts}
20     \subimport{files/}{animal-lefts.tex}
21 \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.

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 `subimported`, and this is the key to organisation.

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.

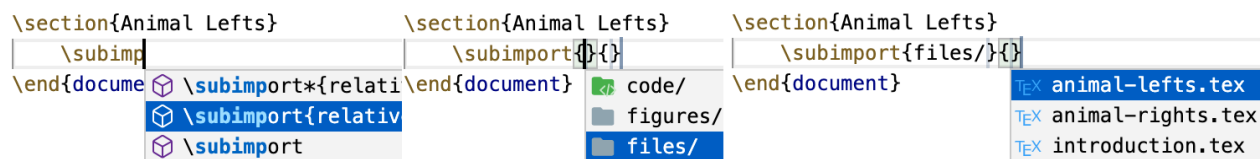


Figure 3.4: Intellisense-assisted picking files.

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.

3.3.2 Figure and caption

Our `introduction.tex` file, then, looks like this:

Listing 3.4: `introduction.tex`

```

1 \begin{figure}[h]
2   \centering
3   \includegraphics{figures/gradient-circle.png}
4   \caption{This is our first picture}
5   \label{fig:gradient-circle}
6 \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-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 Introduction

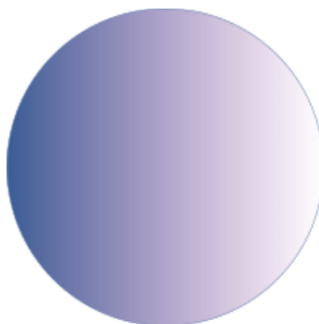


Figure 1: This is our first picture

Figure 3.5: Output from `introduction.tex`

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

Note: 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 so 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.

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

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,

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.

because `\` is also a special character, so you have to use `\textbackslash`.

Table from data

From data we can create `tabular` environments, which can be wrapped in a `table` environment and given captions and labels.

Generally we need the packages `pgfplotstable`, `booktabs`, `array`, that is

```
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{}`. Because we have text in our table, we indicate the `string` type option, and similarly `"` indicating text. By default, it will just import the data and put it on a table, so we can add the horizontal line after the titles with the code in line 4.

```
1 \begin{table}[h]\centering
2   \pgfplotstabletypeset[
3     text indicator=", string type,
4     every head row/.style={after row=\hline},
5   ]{Examples/Example2/cheese-prices.dat}
6   \caption{Replicated from a table!}
7 \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!

There is a lot of customising that can be done, and you can find the documentation for `pgfplotstable` [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`

```
1 \begin{enumerate}
2   \item First numbered item
3   \item Second numbered item. Let's nest another list
4   \begin{itemize}
5     \item First itemised
6     \item Second itemised
7   \end{itemize}
8 \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:

```
1 \begin{enumerate}
2   \item First question
3     \begin{enumerate}
4       \item Sub question
5         \begin{enumerate}
6           \item Item on subquestion
7         \end{enumerate}
8       \end{enumerate}
9     \end{enumerate}
10  \item Second question
11 \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:

```
1 \lstset{
2     numbers=left, frame=single, breaklines=true, %Keep text inside a frame, and number each line.
3     basicstyle = \scriptsize\ttfamily, %smaller size, monospaced
4 }
```

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.

```
1 \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. L^AT_EX is usually in text mode, but anything between $\$...\$$ or $\backslash(...)\backslash$ becomes math mode¹. A math environment allows subscripts, superscripts, greek letters and for most things one would associate with mathematics.

```
1 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:

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

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

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.

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

$$\begin{aligned} f(x) &= (x+1)(x-1) \cdot \left(\frac{\alpha}{5} \right) \\ &= (x^2 - 1) \left(\frac{\alpha}{5} \right) \end{aligned}$$

The usual product symbol is `\cdot`. \times 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 $[]$.

¹The recommended use is $\backslash(...)\backslash$, but it's not handled very well in VSCode. When snippets are covered, you will see a solution, but feel free to use either.

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 fractions.

3.4.1 Making it 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 greek letters is done by typing `@` followed by the first letter of the english name. Some special letters have a “variable” variant, like φ (variable phi) compared to ϕ (phi). So γ is produced with `@g`, `@f` = ϕ and `@vf` = φ .

Inserting brackets is made easy with VSCode 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 + indentations to keep everything tidy.

Finally, the snippet for fractions is `@/`, and to create the unnumbered **align*** environment, type **bsal**, then **tab** — **begin**, **s** for unnumbered, **align**. Similar snippets exist for other environments, such as figure — **bfi**; align — **bal**; align* — **bsal**, etc. A full list can be seen [here](#).

3.4.2 Organisation

Let's look at a simple example

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

$$= \dots$$

The following code describes two “extremes” of how split-up the code can be. Neither is ideal, but as we work through the next examples, it should become clear what a reasonable middleground looks like.

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

Compared to:

```
1 f(x) &
2   = {
```

```

3      \left(
4          \frac{3+x}{x^{2x}}
5          + \left[
6              \sqrt{x} + \sin(2x)
7          \right]
8      \right)
9  }^2
10 \\
11 &= \dots

```

3.4.3 Variables

It is also worth pointing out how to define L^AT_EX 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 accidentally define/redefine a variable within the same *scope*. Variables in L^AT_EX are bound to their immediate environment. If this doesn’t mean anything, just follow these 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.

```

1 \begin{align} \label{eq:test}
2 \newcommand{\x}{\alpha+5}
3 \quad \quad \quad \x
4 \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, redefine the `\vec` variable in the preamble to math bold, like so: `\renewcommand{\vec}{\mathbf}` \Rightarrow **F**.

The `matrix` environment produces matrices without brackets, 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

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

```

1 \begin{align*}
2   \vec{F} = \begin{bmatrix}
3     1 & 2 & 3 \\
4     3 & 2 & 1
5   \end{bmatrix}
6   , \quad \text{\%Long space}
7   A = \begin{pmatrix}
8     1 \\
9   \end{pmatrix}
10 \end{align*}
```

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

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

```

1 \begin{align*}
2   f(x) = \begin{cases}
3     1, x \geq 5 \\
4     -1, x < 4 \\
5     0, 4 \leq x < 5
6   \end{cases}
7 \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` (∞); `@2` \implies `\sqrt{}` ($\sqrt{\quad}$); `@-` \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:

$$f'(x) = \lim_{\delta x \rightarrow 0} \frac{f(x + \delta x) - f(x)}{\delta x} \quad (3.2)$$

$$= \lim_{\delta x \rightarrow 0} \frac{(x + \delta x)^2 + 5 - (x^2 + 5)}{\delta x}$$

$$= \lim_{\delta x \rightarrow 0} \frac{x^2 + 2x\delta x + \delta x^2 + 5 - (x^2 + 5)}{\delta x}$$

$$= \lim_{\delta x \rightarrow 0} \frac{\cancel{x}(2x + \delta x)}{\cancel{\delta x}}$$

$$f'(x) = 2x \quad (3.3)$$

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, δx 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 $x \xrightarrow{0} 5$.

```
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:

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

```
1 \frac{d}{dx} f(x) \quad \frac{d^2}{dx^2} f(x) \quad \frac{\partial^3}{\partial^3 x} f(x, y) \\
2 \quad \frac{\partial^3}{\partial^3 x} f(x, y) \quad \nabla^2 \vec{f} \quad \nabla \cdot \vec{\omega} \quad \nabla \times \vec{F} \\
3 \quad \nabla^2 \vec{f} \quad \nabla \cdot \vec{\omega} \quad \nabla \times \vec{F} \\
4 \quad \nabla \cdot \vec{\omega} \quad \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 $f : [a, b] \rightarrow \mathbb{R}$ 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:

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

Now if f is integrable within the interval and Δx_i approaches zero, we have an integral:

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

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

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

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:

1

```
\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.

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

1

```
\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.

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

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

3.5 Graphs with Tikz

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](#).

```
1 \begin{figure}[h]\centering
2   \begin{tikzpicture}
3     \begin{axis} [
4       xlabel=\(x\) , ylabel=\(f(x)\), title=Example title,
5       domain=0:5, legend pos = north west,
6       axis lines = left,
7     ]
8       \addplot[blue] {x^2};
9       \addlegendentry{\(x^2\)}
10      \addplot[red] {x^3+2};
11      \addlegendentry{\(x^3+2\)}
12    \end{axis}
13  \end{tikzpicture}
14  \caption{Example caption}
15 \end{figure}
```

`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, and the best approach is to look through examples choose what you want to replicate. The names tend to be self-evident, but we will briefly touch on them with each example.

```
1 \begin{axis}[
2   %options here separate by commas,
3 ]
```

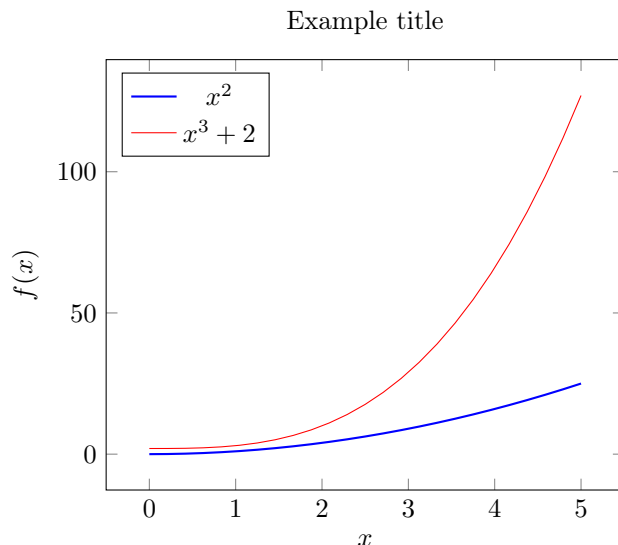


Figure 3.6: Example caption

```

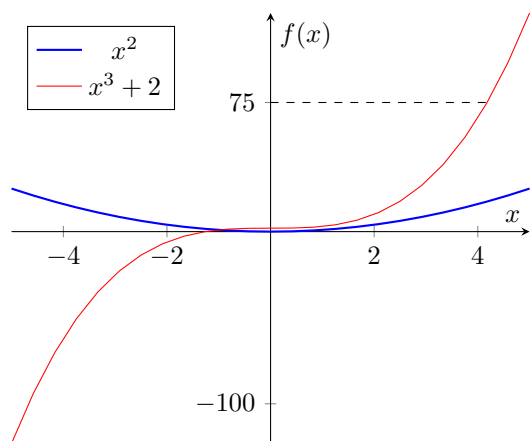
4      %plots here separated by colons;
5      \end{axis}

```

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.

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.



```

1 \begin{tikzpicture}
2   \begin{axis} [
3     xlabel=\(x\), ylabel=\(f(x)\),
4     legend pos = north west,
5     domain=-5:5, axis lines = center,
6     ytick={-100,0,75},
7   ]
8     \addplot[blue, thick] {x^2};
9     \addlegendentry{\(x^2\)}
10    \addplot[red] {x^3+2};
11    \addlegendentry{\(x^3+2\)}
12    \addplot[dashed, domain=0:4.18] {75};
13  \end{axis}
14 \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 $\backslash(x^2\backslash)$.

3.5.1 Bar plot

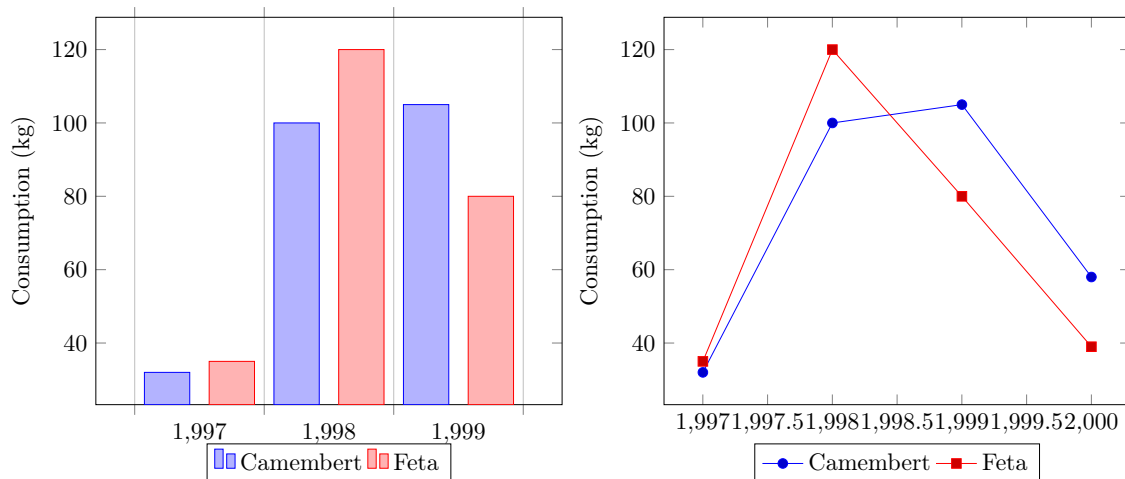


Figure 3.7: A bar plot and the same data without `ybar`.

```

1 \begin{tikzpicture}[scale=0.9]
2   \begin{axis}[
3     ybar interval=0.7,
4     ylabel=Consumption (kg),
5     legend style={
6       at={(0.5,-0.1)}, anchor=north, legend columns=-1
7     },
8   ]
9     \addplot coordinates {
10      (1997,32) (1998,100) (1999,105) (2000,58)
11    };
12     \addplot coordinates {
13      (1997,35) (1998,120) (1999,80) (2000,39)
14    };
15     \legend{Camembert, Feta}
16   \end{axis}
17 \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 };
```

3.5.2 Plot from data

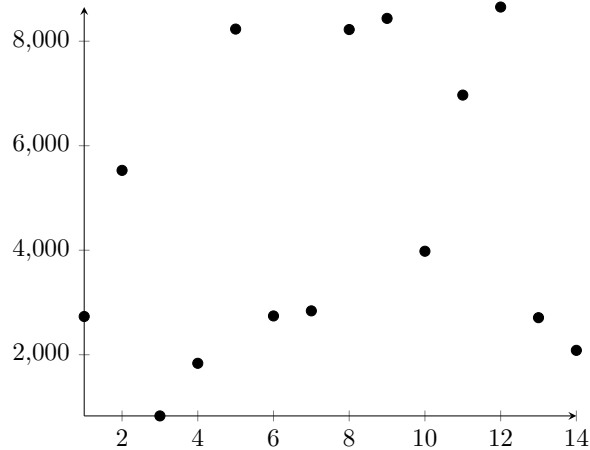
Data files are usually either `.csv` (with columns separated by commas), `.dat` or `.txt` (columns separated by tabs). Generally if copying directly from LibreOffice/Excel, columns will be tab separated; whilst Octave/Matlab data by default is comma separated.

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
2	1	2731.60851756311	13.2894415066652
3	2	5527.96064928581	23.0248555701667
4	3	828.88421543021	26.0238681538563
5	4	1835.9477498787	27.3949401553229
6	5	8232.23568836794	27.9277183310726
7	6	2742.10952781195	35.574656094906
8	7	2839.26593759326	32.6529698844285
9	8	8223.29229933961	35.585410610198
10	9	8436.78393620134	43.2400122448311
11	10	3980.50189677834	43.1362128057345
12	11	6969.0856706613	46.5513604389675
13	12	8653.93008312339	48.190253854583
14	13	2709.0352967109	57.5333844238011
15	14	2083.69661702771	59.2944882816431

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.

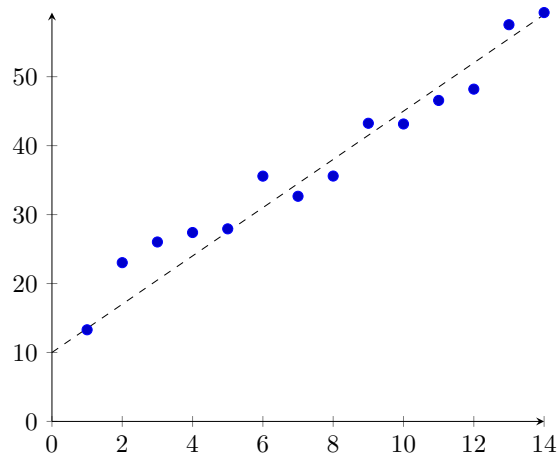


```

1 \begin{tikzpicture}
2 \begin{axis}[axis lines = left]
3   \addplot [only marks] table {Examples/data.
4     dat};
5 \end{axis}
6 \end{tikzpicture}

```

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**.



```

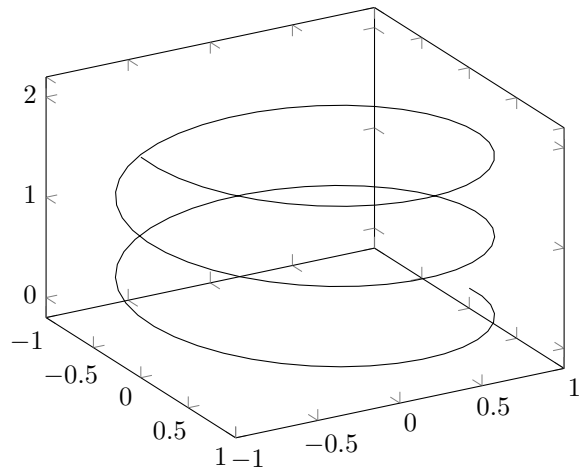
1 \begin{tikzpicture}
2 \begin{axis}[ axis lines = left, ymin=0, ]
3   \addplot+ [only marks] table[x=Var1, y=Var3
4     ] {Examples/data.dat};
5   \addplot [dashed, domain=0:14] {10+3.5*x};
6 \end{axis}
7 \end{tikzpicture}

```

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. Nevertheless, we will present a couple of examples here and go more in-depth with importing graphs in the following section.

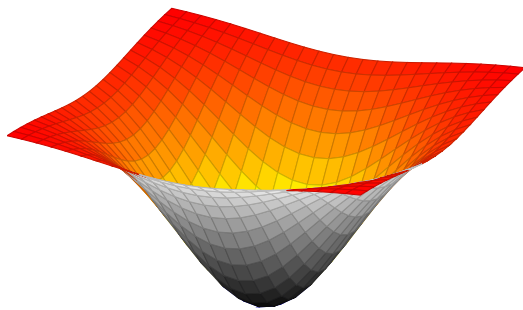
Parametised plots are the simplest to get started, but pay attention to the use of **x** in all directions. Note that these examples are taken from **pgfplots**' documentation.



```

1 \begin{tikzpicture}
2   \begin{axis}[view={60}{30}]
3     \addplot3 [domain=0:5*pi,samples=100,
4               samples y=1,]
5       (
6         {sin(deg(x))}, {cos(deg(x))}, {2*x
7           /(5*pi)}
8       );
9   \end{axis}
10 \end{tikzpicture}

```



```

1 \begin{tikzpicture}
2   \begin{axis}[hide axis,mesh/interior
3               colormap name=hot,colormap/blackwhite
4               ,]
5     \addplot3 [domain=-1.5:1.5,surf] {-exp
6       (-x^2-y^2)};
7   \end{axis}
8 \end{tikzpicture}

```

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