KOMPENDIUM

Writing science with LaTeX

Peter Jansson

Institutionen för naturgeografi 1 October 2020

Stockholms universitet

Contents

1 Scientific writing and LaTeX **5**

2 A quick start to writing in LaTeX **5**

3 Expanding your manuscript **9**
3.1 Types of commands . . . 9
3.2 The document structure . . . 9
3.3 Document classes . . . 10
3.4 Reserved characters . . . 10
3.5 Type face styles . . . 11
3.6 Typographic features . . . 12
3.7 Breaks and space . . . 13
3.8 Headings . . . 14
3.9 Table of contents . . . 14
3.10 Environments . . . 15
3.11 cross-referencing . . . 16
3.12 Special commands . . . 17

4 Tabular information **17**

5 Graphics **19**

6 Floats; tables and figures on the run **20**

7 Mathematical typesetting **21**

8 Extending the functionality of LaTeX, packages **23**
8.1 amslatex . . . 23
8.2 babel . . . 23
8.3 booktabs . . . 24
8.4 caption . . . 24
8.5 dcolumn . . . 25
8.6 float . . . 25
8.7 geometry . . . 25
8.8 graphicx . . . 26
8.9 hyperref . . . 27
8.10 inputenc . . . 27
8.11 lineno . . . 28
8.12 lipsum . . . 28
8.13 natbib . . . 28
8.14 siunitx . . . 29
8.15 tikz/pgf . . . 29
8.16 xcolor . . . 32

9 Managing references **33**
9.1 Integration of Overleaf and Zotero . . . 34
9.2 The `natbib` package . . . 34
9.3 The fully automated reference system . . . 35
9.4 The semi-automated reference system . . . 36

10 Automating LaTeX or creating your own commands **37**

11 Copying documents from Word to LaTeX **38**

12 Making presentation slides in LaTeX **39**

13 Reinventing the wheel? ...or not **40**

14 How does it really work? **41**

15 Saving the worst to last **43**

A Running LaTeX on your own computer **45**
A.1 LaTeX . . . 45
A.2 LaTeX editors . . . 45
A.3 Reference management . . . 45
A.4 Yet more software that might be of use . . . 46

B Install LaTeX on your computer **46**
B.1 Windows 46
B.2 Mac 46
B.3 Linux 47

C Learning more on LaTeX **47**

D Revision history **48**



Scientific writing with LaTeX is copyright © 2020 by Peter Jansson and available through its doi: http://dx.doi.org/10.17045/sthlmuni.3205753
This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/

Preface

This document is intended to provide you with a good working knowledge of LaTeX so that you can freely use it for your scientific output. LaTeX was, and is, created by scientists for scientists and is widely used by publishers for both journals and books. To have working knowledge of LaTeX should be part of every scientist's toolbox.

There are two choices for running LaTeX, on-line or off-line tools. These should not be considered mutually exclusive, you may need to work both ways. In this text I will assume you are working with the on-line platform Overleaf which is what I would also recommend to anyone, beginner or experienced, since all you need is a web-browser and an internet connection, no installation on your computer. Overleaf is similar to Google Docs in that documents are stored in the cloud and that several persons can interactively collaborate on a document. The drawback is of course that you cannot work on your documents without internet access. You can, however, synchronise your files in the cloud with Dropbox or Git so that you always have updated versions on your own computer. To work off-line you need to install a LaTeX environment locally on your own computer; this is covered in appendix A.

So I strongly advice the beginner to sign up for and use Overleaf. Note that you may be covered by an institutional license if you study or work at a university. Please visit the Overleaf site and look at the introductory video to familiarise yourself with the Overleaf environment. This document was entirely written on and using Overleaf but does not detail how you use the interface. Of particular use is the keyboard shortcuts sheet.

Although not covered in any detail in this document, I also strongly urge you to start using a reference manager for handling your scientific references. Unless you already use a software I would recommend using the online manager Zotero which integrates with Overleaf. This integration is briefly covered in section 9.1.

In this text, I will use `typewriter` type face to highlight LaTeX code. In the cases where a complete section of code is given

```
it will be presented in a blue field.
```

Apart from what is covered in this text, you probably should download the LaTeX 2_ε Cheat Sheet and the LaTeX quick reference and keep them handy. The comprehensive list of LaTeX symbols is also useful to have handy. You should also be aware of the TeX StackExchange site, which is a question and answer site where probably all your questions are already answered. At the time of writing there are 203 733 answered questions on the site. You can also find many solutions to type-setting issues by typing 'latex' followed by whatever you are looking for. It is almost certain some of the highest hits are from the TeX StackExchange.

More details on how different macro packages work can be obtained from their documentation which is available on the Comprehensive TeX Archive Network (CTAN) site. At the time of writing there are 5910 macro packages contributed by 2715 different users.

1 Scientific writing and LaTeX

Apart from the linguistic aspects of scientific writing, such as clarity, brevity and density, what distinguishes scientific writing is the use of many technical terms, technical notation and cross-referencing of citations, figures, tables and equations as well as the need to follow precise document formatting instructions. Clearly you can accomplish the first aspect with any type of writing tool from the Goose pen to a modern word processor, it is the second part that requires some special tools. This is where LaTeX fits.

Now LaTeX is not the universal response to all writing issues; as with everything there are definite pro's and con's. Since LaTeX involves knowing, or at least familiarity with, a fair number of commands and understanding how to use them, it is associated with a learning curve similar to that of learning a programming language. In fact, it is a programming language (Turing complete). It is therefore evident that it makes little sense to learn LaTeX for menial tasks such as writing a letter or a memo to colleagues or a list for making purchases, that is the realm of Word. LaTeX comes into its own when the document is more complex such as a thesis or scientific paper.

There are probably many different imaginable tasks where LaTeX is preferable, I will just mention a few. When you write a scientific article you greatly benefit from using LaTeX because you can clearly express, for example, complicated notation and mathematics. It is also very likely that the journal accepts LaTeX manuscripts in their own journal format. LaTeX is also very useful when you try to write a long document, book or similar, where you need cross referencing, you may need to create a table of contents and perhaps an index; tools built into LaTeX. A third type of document is that which has to be formatted exactly the same time after time. This could, for example, be a report series from a project or organisation or from ongoing research. In all such instances, there is little competition.

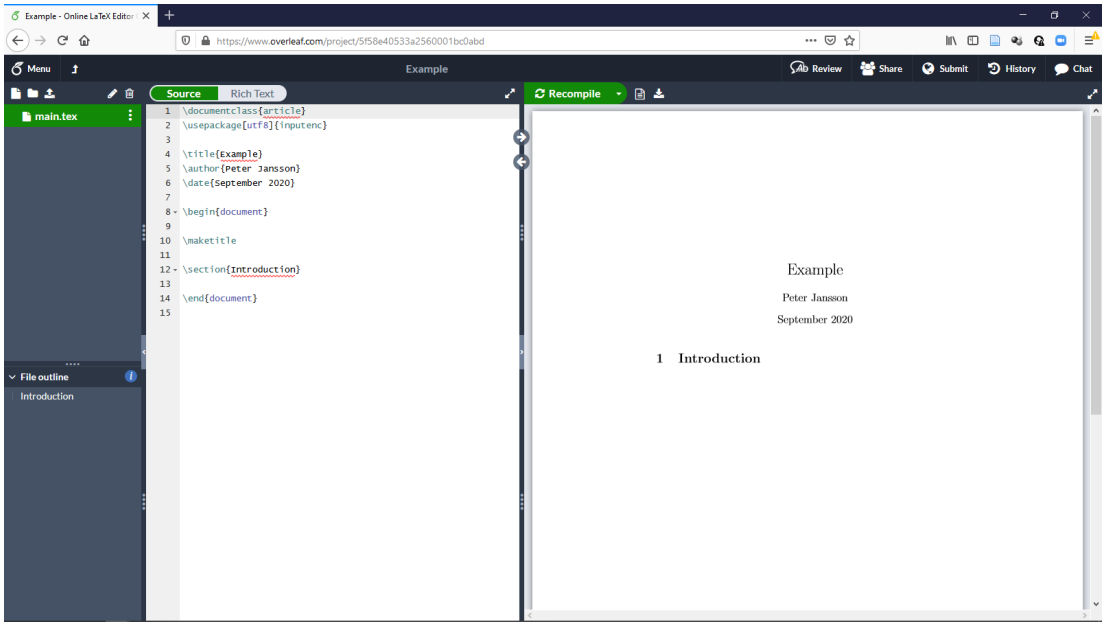
So the conclusion is that the more complex your writing task the more you will benefit from using LaTeX. Another way to describe this is that what is difficult in Word is simple in LaTeX and *vice versa*.

This compendium starts with a basic introduction on how to write a simple document. This covers almost everything you need to know to work with LaTeX. Once you understand the basics you have the foundation to build more advanced features which is what most of the compendium concerns. By expanding the capabilities you tap into a wealth of features that makes LaTeX unique.

2 A quick start to writing in LaTeX

In this section I will provide the basics to produce a document in LaTeX. When you study this section I expect you to follow along and try the examples in your own document. Let's dive in!

In Overleaf you start a new blank project and provide a project name. In my example, I called it 'Example'. Overleaf will then open the document in a four pane display as shown in the figure below. The top left is the *project pane* which displays what files are included in the project. By default the file you receive will be named `main.tex` but this can be changed to something more meaningful. Below the project is the *File outline* pane which will show the document outline. The center pane is the LaTeX *code pane* which is where you enter your text and code to create your document. Finally to the right is the *preview pane* which shows how the document looks.



At this point it may have become evident that we essentially are programming the document instead of writing in a 'what-you-see-is-what-you-get' (WYSIWYG) environment such as that of an ordinary word processor. What you write in the code window must be compiled by LaTeX to yield the final layout. This allows LaTeX to not only consider what comes before a particular point in a document but also what comes later to optimise the layout throughout. This also means that we need to make Overleaf compile our document before we see anything new that we have added to the document. This is accomplished by pressing the green 'Recompile' button at the top of the preview pane.

The simple document we receive in Overleaf shows us several things. First you notice is that there are commands identifiable by the backslash and often with an argument within curly braces, such as `\documentclass{article}`. LaTeX produces documents in response to such commands, similar to how you build a web-page in html. An important point to note is that all commands in LaTeX are *case sensitive* so make sure you do not accidentally use the wrong case in a command. It is useful to know that all basic LaTeX commands are written in lower case. You will with time learn many commands but it is still a surprising few that are needed to get you far.

Let look at the basic document example:

```
\documentclass{article}
\usepackage[utf8]{inputenc}

\title{Example}
\author{Peter Jansson}
\date{September 2019}

\begin{document}

\maketitle

\section{Introduction}

\end{document}
```

We can identify three commands:

```
\documentclass{article}

\begin{document}

\end{document}
```

These are mandatory for all LaTeX documents and must hence be present to tell LaTeX what kind of document to produce (in this case an article; `\documentclass{article}`) and where the document text that should be type set begins and ends (`\begin{document}`–`\end{document}`).

The content of your document must be placed between the `\begin`–`\end{document}`. If you place anything after the `\end{document}` it will simply be ignored. You are not allowed to place anything before the `\documentclass{article}` command. The space between the `\documentclass` and `\begin{document}` commands is reserved for adding features that add to or modify your document. This is refereed to as the *preamble* of the document. We will get back to this in more detail but the basic rules outlined above must be followed.

In our example we can identify three commands in the preamble

```
\title{Example}
\author{Peter Jansson}
\date{September 2020}
```

These commands are allowed to be placed in there. The first command `\title{}` allows you to provide a title for your document and the second `\author{}` allows you to provide your name to the document. The thirds allows you to enter a date. In my case Overleaf automatically entered 'September 2020' because this is when I started the project.

The three commands creates the document header we see in the preview pane. However, they will only display if we enter a fourth command `\maketitle` which takes the first three and produces a title for your document. Note that this command must be placed after the `\begin{document}` because it generates the title as part of our document. It is possible to remove the `\date{}` command, `\maketitle` will then extract the current date from the computer and display the date instead of the text provided in the example.

What we now have is the header of the document. We now turn to the content.

In the remaining document text you see the command `\section{Introduction}` which produces the heading '1 Introduction'. Note that headings are numbered by default. LaTeX has predefined the look of the headings so that all you have to do to start a new section in your text is to use the `\section{}` command and add your heading text within the curly braces. The heading will always be on a separate line with spacing above and below the heading.

After the heading you can simply type in the text you want, the text body that belongs to that heading. When you do so you will probably notice a few unexpected things. If you try to start a new paragraph by pressing the enter (or return) key 'Word style', no new paragraph will start. Instead the text runs on continuously. This is because LaTeX uses an empty line to identify a new paragraph. So, you need one return to end the line and a second to cause the paragraph break. Adding several returns between paragraphs will still result in a normal paragraph break.

By default LaTeX uses indentation rather than empty vertical space to signal paragraph breaks. This can of course be changed. LaTeX also uses the typographic standard that the first paragraph after a heading is *not* indented but following paragraphs are. The default settings in LaTeX always follow typographical standards.

Another aspect that can cause confusion is that LaTeX determines all relevant spacing in a document. This means that, for example, if you type in four consecutive spaces LaTeX still interprets this as one space. This also applies if you try to use the tab key to offset text. It will still result in a single space. So you will not be able to push text around by adding several spaces or tabs. We will return later to how we can create both horizontal and vertical space in a document if we really need to.

A final command that I have not commented on is `\usepackage[utf8]{inputenc}`. This line asks LaTeX to load a macro package called 'inputenc' (input encoding; section 8.10) into your document. The reason for this is that LaTeX by default does not recognise accented letters. The package allows LaTeX to understand accented letters such as the Swedish å, ä and ö. If you type in accented letters in a LaTeX document without having loaded the 'inputenc' package, they will simply not be visible.

Having studied these few basic commands there is an important issue we need to realise. Whenever we use curly braces ({ }) or square brackets ([ ]) in a command they need to be paired. If you accidentally pair a curly open brace with a close square bracket ({]) or *vice versa* LaTeXwill issue an error and the command will not be executed. So as in any programming language, make sure opening and closing braces and brackets match. In Overleaf you always get suggestions for how the commands should be typed and constructed which means you get the correct matching braces and begin-end forms.

Do it

- Add some text of your own and observe what happens if you put many consecutive spaces or tabs in the text and use the return key to make new paragraphs.

Let us now look at some basic features of a document. A mostly overused but nevertheless useful feature of a document is the *list*. A bullet list in LaTeX is referred to as `itemize` and is invoked by the following code

```
\begin{itemize}
  \item first item
  \item second item
  \item third item
\end{itemize}
```

- first item
- second item
- third item

As you can see the list created by the `\begin`–`\end{itemize}` environment. Each item, bullet point, is created by writing the command `\item` followed by the content of that bullet. There are other forms of lists and you can create your own but we will cover that in section 3.10.

Do it

- Make your own list. Try nesting lists.

Once we have made a list we may consider adding a table

```
\begin{tabular}{l c r}
\hline
a  &BB  &CCC\\
aa &B   &CC\\
aa &BBB &C\\
\hline
\end{tabular}
```

a	BB	CCC
aa	B	CC
aa	BBB	C

A table is thus created by a `\begin`–`\end{tabular}` environment. After the `\begin{}` is a set of curly braces showing the letters `l c r`. This tells LaTeX there will be three columns and shows the alignment for them: left, centre and right in this case.

The table itself is made up of several rows of data. Each row has the form `  & & \\`. The ampersand indicates a tab separating columns and each row of information is ended by a double backslash, which is the LaTeX instruction to end a line. So the first row in our table should be understood as '`a - new column - BB - new column - CCC - new line`'.

In the table, you also see the command `\hline` which produces a horizontal line. The basic table we get is not exactly beautiful but we will return to tables in section 4 and how to improve on this later. For now you know the principles to make a table.

Do it

- Make your own larger table. Use your own data or try to mimic something from a journal.

Now let us look at an original strength in LaTeX, typesetting mathematical notation and equations. Type in the following into your document

```
The definition of
$X_Z$ is given by
\begin{equation}
X_Z = \sum x_z +  \beta^n
\end{equation}
```

The definition of X_Z is given by

$$X_Z = \sum x_z + \beta^n \qquad (1)$$

This equation is of course nonsensical but serves the purpose of showing some mathematics. First, you notice that there is code in the sentence: `$X_Z$` which yields X_Z. The dollar signs signals to LaTeX that it should switch in and out of *mathematical mode*. LaTeX works in two modes, mathematical and *text mode*. Certain commands and features only work in either mathematical or text mode, rarely in both. If you use them in the wrong mode they give rise to an error. In this example the underscore `_` is LaTeX mathematical code for a subscript.

It is very important to remember to keep the dollar signs paired. If you left the code above without a dollar sign, `$X_Z`, LaTeX would continue to think everything afterwards until the end of the paragraph is mathematics and eventually yield an error message.

After the sentence there is a construction using the environment `\begin`–`\end{equation}`. This is the way you create *display mathematics*, equations that are on a separate line and as in this case also centred and numbered. In the equation given by the code `X_Z = \sum x_z + \beta^n` you see a couple of mathematical commands `\sum` which yields the summation sign and `\beta` which yields the Greek letter β. You also notice the LaTeX command for superscript, `^`.

More on mathematical typesetting will be discussed in section 7 but for information, use, for example, the LaTeX cheat sheet to see examples of mathematical notation.

Do it

- Make your equations using the cheat sheet.

You now have almost all you need to write a complete scientific manuscript, but you obviously wonder about figures (graphics) and references. This is a bit more involved and we will return to these aspects in sections 8.8 and 9.

3 Expanding your manuscript

In the quick start, we briefly touched on several types of formatting in LaTeX. You can go a long way with what you have already learned to write a scientific document, but what you lack is the knowledge of how to change the layout. It is therefore time to provide a more comprehensive view.

3.1 Types of commands

LaTeX has three different forms of commands. The simplest form is a command that does not take any argument. An example is `\LaTeX` which yields the formatted word LaTeX. These commands may produce unwanted effects if you just type them in as they are. If you, for example, type `\LaTeX is` you will get LaTeXis, where the space between 'LaTeX' and 'is' disappears. The reason for this is that `\LaTeX` and any simple command needs to signal to LaTeX that the command is completed. A space is one way to accomplish this but then the space is 'consumed' and cannot be used to create a space between words. This means that the space we typed will not be used as an inter-word spacing but just to complete the command and 'is' will start immediately after the command. So to get the space we need we must remember to provide a completion of the simple commands. This can be accomplished in several other ways. We can add a backslash after the command as in `\LaTeX\ is` or add a pair of curly braces as in `\LaTeX{} is`. If the simple command is followed by a punctuation such as in `\LaTeX. Is` and `\LaTeX, is` the punctuation acts as completion. Forgetting this detail is the cause for many inadvertent typographical errors in LaTeX.

The second type of command is the command that takes one or more arguments. An simple example of this is `\textit{italics}` which yields *italics*. This type of command uses curly braces, sometimes additional braces for input to the command. These commands do not need any completion since the braces provide the information LaTeX needs to understand that the command is complete. A classical mistake with these types of commands is to forget to add the closing brace which then will yield an error.

The third type of command is the 'environment' `\begin`–`\end{}` where the command has an opening and a closing statement and where the argument is everything you place between the two. We have already looked a three such environments `document`, `itemize` and `equation`. In this case it is important to match any `\begin` with a matching `\end` statement.

It is important to know how the different commands should be written and what may be optional arguments. But you will build your knowledge as you continue working with LaTeX.

3.2 The document structure

In section 2 we saw that the basic LaTeX document consisted of the following three lines of commands

```
\documentclass{article}
    %preamble
\begin{document}
    %document text
\end {document}
```

and we also began writing our text in the `\begin`–`\end{document}` environment. Although we briefly touched upon this structure in the quick start it is worth reiterating the significance again.

First, the `\documentclass` command must be the first command encountered in a LaTeX document. It may be preceded by comments but not by any other command or plain text.

Second, all text that we want displayed in a document must occur between the `\begin`–`\end{document}`. In fact anything you write after the `\end{document}` will be ignored. This can be quite useful since you can move the `\end{document}` up into the text in order to compile only a portion of the text file. It also means you can cut and paste bits of text into your document file after the `\end{document}` for later use.

Third, the space between `\documentclass` and `\begin{document}` is usually called the *preamble* and is reserved for LaTeX code that is used to influence the behaviour of what you write in the document environment. You will see how this part of the document usually gets filled with several useful tools. It is however, important to realise that no ordinary text can be written in this part of the document, only commands. Errors will otherwise occur.

3.3 Document classes

When you begin writing a LaTeX document, you need to select a class or type of document. Most commonly we find ourselves working with 'article' as we have already seen, but, there are other classes that are standard in LaTeX

Class	Description
`article`	Common scientific article
`book`	Book with chapters and parts
`letter`	Letters with letterheads
`report`	Report with chapters
`slides`	Overhead slides

Apart from the standard classes, there are numerous other classes available. Many are written by enthusiasts for particular purposes such as CV, sheet music, sudoku, etc. Other classes have been developed by publishers and journals to facilitate authoring in journal specific formats. For scientific publishing, the latter is probably what we should look for.

To provide some examples, the American Geophysical Union and the Open Access publisher Copernicus has classes for their journals. Most journals on, for example, Wiley, Springer and Taylor & Francis also have class files for their own style. Note that classes may contain new or redefined commands which can lead to unexpected behaviour if you do not carefully check how the class works. In most cases, you will not notice any differences but not all classes may be up-to-date or well written.

Do it

- Look at some key journals in your field and locate their class files. Or, search Overleaf for journal templates. You may want to try them out when you feel ready.

3.4 Reserved characters

When you write text in LaTeX there are a number of characters that are *reserved* because they are used as part of commands and instructions for LaTeX. These cannot be used directly in the text without causing either an error or some unexpected result. You have already come across most: the comment percent sign, the tab stop ampersand in tables and the mathematical mode switch dollar sign as well as super and subscripts hat and underscore signs. In the table below

you also see the commands you need to use if you wish to use these reserved characters in the text.

Character	LaTeX use	Command
%	comment	`\%`
&	tab stop	`\&`
$	mathematical mode	`\$`
^	mathematical mode superscript	`\^{}`
_	mathematical mode subscript	`\_`
{ }	start stop grouping	`\{ \}`
~	non-breaking space	`\~{}`
#	Parameter in macros	`\#`

3.5 Type face styles

As with all word processors, basic text is typed out with the normal type face. If you need to change this there are several different choices summarised in the table

Type	Command
italics	`\textit{}`
bold face	`\textbf{}`
SMALL CAPS	`\textsc{}`
slanted	`\textsl{}`
sans serif	`\textsf{}`
`typewriter`	`\texttt{}`
underline	`\underline{}`

The size of the text can also be varied but not in the way you may think. LaTeX scales fonts relative to the 'normal size', that is the size of the main text. Thus in order to get larger or smaller font sizes you can use the following set of commands, relative to the 'normal size'.

Font size		Type	Example
5	6	`\tiny`	the quick brown fox
7	8	`\scriptsize`	the quick brown fox
8	10	`\footnotesize`	the quick brown fox
9	11	`\small`	the quick brown fox
10	**12**	`\normalsize`	the quick brown fox
12	14	`\large`	the quick brown fox
14	17	`\Large`	the quick brown fox
17	20	`\LARGE`	the quick brown fox
20	25	`\huge`	the quick brown fox
25	25	`\Huge`	the quick brown fox

The benefit of this is that as you change the global font size all relative scaling will also change.

The default type size for LaTeX is 10pt. The basic sizes you can use are 10, 11, and 12pt and this can be set using the `\documentclass` command as an *optional argument*. If you want to make an article in 11pt you need to provide

```
\documentclass[11pt]{article}
```

at the beginning of your document. Here the square bracket indicates that the command takes an optional argument stated within the brackets. If you do not specify any optional argument LaTeX will use a default value.

Changing type faces is not necessarily straight forward; I am almost tempted to state you should not even try, yet. The reason for the hesitation is that while LaTeX typesetting has remained more or less constant since its infancy, fonts have not. This is because as computers

have gone from 8-bit to 64-bit and capacity has sky-rocketed, Fonts have developed from containing a basic 128 character ASCII set to now over a million characters. With the backwards compatibility of LaTeX it has not been possible to develop with the new standards but rather find ways to implement the new while keeping the old.

In Overleaf, the default is an implementation of LaTeX called pdfLaTeX. This directly generates pdf files from your LaTeX source and is probably the most widely used form of LaTeX. In the core LaTeX distributions (incl. Overleaf) many fonts are installed that work with pdfLaTeX.

Another implementation called XeLaTeX was developed to make use of the newer Unicode type faces containing over a million characters. If you run XeLaTeX you can quite easily access your system type faces (the same you access in Word). This is quite tempting but remember that in order for someone else to run your document, they need to have the same type face. So there are benefits and pitfalls with going either way. We will leave the type face question for now.

In section 2 we had issues with accented letters. This problem is easily circumvented by switching to the XeLaTeX compiler in Overleaf since we can then access such accented letters the way they are entered from our computer keyboard. So if you are writing in languages that use accented letters it may be useful to use XeLaTeX instead of pdfLaTeX.

That said, I can mention the LaTeXFont Catalogue, which contains an overview of generally available free type faces and also how to make them usable in your document. An important point to make is that only certain type faces have support for mathematics. These are identified in the font catalogue. If you use a type face that does not include mathematics anything written with mathematics will be written using the default font. In some cases you will probably barely notice the difference but in other cases it will just be ugly. The freedom to chose can thus be quite limited.

You can try to change fonts by following the instructions in the font catalogue and try other solutions found on TeX StackExchange.

Do it

- Play around with the different ways to change font and size, including the switch between 10, 11, and 12 pt in the `documentclass` command.

3.6 Typographic features

LaTeX contains many special characters and features. A list of some of these is

Feature	Command	Function
‘ ’	single grave accent and apostrophe	British open–close quote
“ ”	double grave accent and apostrophe	American open–close quote
–	`--`	en-dash
…	`\ldots`	–
©	`\copyright`	–
®	`\textregistered`	–
TeX	`\TeX`	–
LaTeX	`\LaTeX`	–

Most of these need no further comment but I will point out a couple of details. First, when it comes to using quotes, it is possible to use the common double quote on the keyboard but it does not look very good. In addition, double quotes in English is American typography so unless you are American or come from a country that uses the American typesetting standard, you should stick to the British English single quote.

Second, a common problem with texts is that authors do not know the difference between a hyphen (dash) and a longer dash (en-dash). The hyphen is part of your key board and used when you hyphenate words or put two words together. The longer dash is used when you want the meaning to be ‘to’ as in 4–7 (four to seven). The longer dash looks like a normal minus sign but LaTeX actually have a special minus sign produced in math mode, compare – (en-dash) and − (math minus). Since it is very easy to distinguish between the different types of dashes in LaTeX you should make an effort to use them appropriately.

3.7 Breaks and space

When you write text you need to occasionally deliberately break the text or words to fit the space available. One of the strong points of LaTeX is to automatically manage these duties but it is still not possible to automate typesetting to 100%.

Let us start by looking at a few commands that are useful for breaking text over pages and lines.

Command	Function
`\newpage`	page break
`\pagebreak`	page break keeping justification
`\\`	line break without new paragraph
`\newline`	line break without new paragraph
`\linebreak`	line break keeping justification
`\nolinebreak`	–
`\noindent`	prevent indentation of a paragraph
`\indent`	indent a non-indented paragraph
`\-`	conditional hyphenation
`\ `	keep normal space
`\@`	end of sentence space after capital letter

Most of these are quite self-explanatory and you can test their function in your own text.

The conditional hyphen and hyphenation as a whole may deserve a comment. If you end up with a word that is too long and it does not seem to hyphenate at all or perhaps wrong (LaTeX uses a set of rules for hyphenation), then you can insert a conditional hyphen. If we take the word 'multispectral', we can prepare the word with conditional hyphens so that it will know where breaks can be, the code will be `mul\-ti\-spec\-tral`. These conditional breaks will not show unless used.

In addition, if you use a word, for example, a complex scientific term that is unlikely to be hyphenated correctly you can provide LaTeX with information how to treat the word in the text. By placing

```
\hyphenation{mul\-ti\-spec\-tral}
```

in the preamble, you pass this information to the hyphenation engine to correctly deal with the word any time it is encountered in the document.

By default LaTeX uses English hyphenation rules. If you write in another language you may therefore get word breaks that do not fit that language. You can correct this problem by using a language package called 'Babel' which we cover in section 8.2.

LaTeX uses an intricate system to keep track of good word spacing. By default you will get a slightly wider space after a period, indicating the end of sentence and beginning of a new. This rule is, however, not always correct. If we have an abbreviation with periods each period will be interpreted as an *end-of-sentence*. If we look at the following example

Example	Code
J. Geophys. Res.	`J. Geophys. Res.`
J. Geophys. Res.	`J. Geophys.\ Res.`

You can see that the space between 'Geophys.' and 'Res.' becomes shorter in the second case, in fact it is a common inter-word space. This means that you should add a backslash after abbreviations written in common letters unless the period really *is* the end of the sentence.

Another issue that can arise is that LaTeX by default interprets a period after a capital letter as an abbreviation and hence provides inter-word spacing. If you end a sentence with such an abbreviation the space is too short. To solve this you can add the command `\@` between the abbreviation and the period as in `…in UNESCO\@. Another …`.

Lastly, in the old days of days of monospaced typewriter fonts, it was customary to insert two spaces between sentences. You sometimes see people still do this in word processors. LaTeX inserts a space that is slightly longer than a single space, but still shorter than two, by default. It is possible to get single space between sentences by adding the command `\frenchspacing` at the

beginning of the document. You can turn the French spacing off by adding `\nonfrenchspacing`. Since adding two or more spaces between sentences in LaTeX does not affect the spacing, it does not matter if you accidentally hit the space bar a couple of times extra, the only way to really easily affect the spacing is through the French-spacing commands.

Do it

- Test the different commands affecting spacing within your own text so that you understand the difference they make. It may be beneficial to work on several identical paragraphs in parallel so that you can visually compare the results.

3.8 Headings

The headings in LaTeX come in two flavours, numbered (default) and unnumbered. The unnumbered versions can be obtained by using so called 'starred' versions of the command. The section command `\section{}` is thus converted to unnumbered by writing `\section*{}`.

Numbered	Unnumbered
`\part{}`	–
`\chapter{}`	`\chapter*{}`
`\section{}`	`\section*{}`
`\subsection{}`	`\subsection*{}`
`\subsubsection{}`	`\subsubsection*{}`
`\paragraph{}`	`\paragraph*{}`
`\subparagraph{}`	`\subparagraph*{}`

The levels part and chapter are only available in the book class.

It is possible to produce special numbering for appendices by using the command `\appendix` where the appendices start. This will force LaTeX to start alphabetic sectioning numbering instead of numeric, which is the common form for an appendix.

There are two commands called `\frontmatter` and `\mainmatter` that numbers pages with roman numerals from the frontmatter command until the main matter command. After that numbering will be with Arabic numerals. this is common in type setting books where pages with roman numerals are used for pages containing material that is not part of the main book content such as table of contents, preface etc.

Do it

- Try out the different forms of headings. Note, however, that for some to work you need to switch your document to the `book` class.

3.9 Table of contents

To create a table of contents is very simple in LaTeX. You need to do one thing and that is to type the command `\tableofcontents` where you wish the table of contents to appear. There is, however, one caveat. The table of contents will only be automatically generated from the numbered sections. If you use starred heading commands you will not see a table of contents. This can be used for special purposes but also remedied to obtain a table of contents for stared heading commands.

If you, for example, have numbered headings in your document and edit in a star in one of the heading commands, that heading will no longer be visible in the table of contents. There is a work-around for this by changing the way the table of contents work. If you add

```
\setcounter{secnumdepth}{0}
```

to the preamble this tells LaTeX not to number any sections. By adding or removing this line from your file you can essentially turn numbering on or off.

Another way to accomplish this is to add

```
\addcontentsline{toc}{type}{heading title}
```

after each heading. `type` refers to the heading level. If you for example have an unnumbered section and subsection that should be in the table of contents you will provide the following after each heading

```
\section{The section heading}
\addcontentsline{toc}{section}{The section heading}

\subsection{The subsection heading}
\addcontentsline{toc}{subsection}{The subsection heading}
```

The `\addcontentsline` command tells LaTeX to add information about a section to a file that it keeps to create the table of contents. The first argument is the name of the file. This will be a file named the same as your document but with the extension `.toc`. The second argument tells LaTeX what level of heading you add to the file. The third is the heading text.

In LaTeX you can also provide similar table of contents for your figures and tables using the commands `\listoffigures` and `\listoftables`. We will get back to details how this can be accomplished in section 6.

If you have many sections of all levels it may make sense to remove the lowest level (subsubsection) from the table of contents in order to shorten it. The command `\setcounter{tocdepth}{1}` is then useful because it allows you to limit the 'depth' of the table of contents. With the argument '1' only sections and subsections will be visible.

3.10 Environments

In the quick start we came a cross several environments. When you enter text there are several more that you should be aware of. Common to all is that the use the environment structure `\begin{}`–`\end{}`.

Environment	Action
`center`	centers content
`flushleft`	flushes content left
`flushright`	flushes content right
`quotation`	decreases text area width for a quote with indentation
`quote`	decreases text area width for a quote without indentation
`verse`	for poetry
`minipage`	produces a separate environment with its own dimensions
`itemize`	produces bullet lists
`enumerate`	produces numbered lists
`description`	produces lists with key words as 'bullets'
`verbatim`	reproduces all text including protected symbols as written

Most of these environments are self-explanatory and something you can explore on your own.

The `minipage` environment is worth looking into a bit deeper. In this text I have used the minipage to create 'side-by-side' looks at code and result. The following display

```
\begin{itemize}
  \item first item
  \item second item
  \item third item
\end{itemize}
```

- first item
- second item
- third item

is created by

```
\noindent\begin{minipage}[c]{0.4\textwidth}
\begin{verbatim}
\begin{itemize}
  \item first item
  \item second item
  \item third item
\end{itemize}
\end{verbatim}
\end{minipage}\hfil
```

```
\begin{minipage}[c]{0.4\textwidth}
\begin{itemize}
  \item first item
  \item second item
  \item third item
\end{itemize}
\end{minipage}
```

This may seem daunting but if you look closely, it consists of two `minipage` environments typeset side by side (no paragraph break) and where the mini-pages are set to be 40% of the text width (`0.4\textwidth`) each. What is not obvious from the typesetting is that I have pushed the two mini-pages apart by inserting the command `\hfil` which is a basic command for inserting space if there is extra space to be had. There are a whole series of such commands but they are beyond the scope of this text. I can only urge you to start a little self study on the more advanced topics that allows you to dive into the interior of LaTeX.

The `description` environment is very useful and works just like the `itemize` and `enumerate` environment but with the exception that you need to provide an argument.

```
\begin{description}
  \item[1] first item
  \item[two] second item
  \item[III] third item
\end{description}
```

1 first item

two second item

III third item

It is also possible to change the look of all list environments. A simple way to change the bullets in `itemize` is to add an argument after `\item[]`. If you want an en-dash instead of the bullet you simply write `\item[--]` for each of the items you want to list.

Do it

- Try the different environments. In the case of lists, try to nest them to see the effects of nesting on their behaviour.

3.11 cross-referencing

LaTeX comes with a powerful cross referencing system. The system works on the principle that you put a label on everything you want cross-referenced and then use a command to take information about that label and insert it into the text. The following are the commands you may encounter.

Command	Action
`\label{}`	establishes a label
`\ref{}`	takes a label and replaces it with a number
`\eqref{}`	specific references to equation numbers
`\pageref{}`	specific references to page numbers
`\cite{}`	generic literature references (see section 8.13)

The `\label` command is used to assign a unique name (label) to a section, figure, table or equation. You need to provide a name for the label and this can be any name you chose. If we, for example, have a map in our manuscript, I would label this figure `\label{fig:map}`. First, I would recommend that you have a descriptive word in the label so that you know what that figure is. Second, I prefer to add a 'tag' `fig:` for figure `tab:` for table `eq:` for equation and `sec:` for section so that I do not confuse a tag for a figure with one for an equation. You can come up with your own scheme, however, you will be limited to letters a–z, numbers and basic punctuation for your labels. So no protected symbols or accented letters can be used.

Once you have assigned a label to an object you can reference, for example the map we labelled above as `\ref{fig:map}`. In the resulting layout LaTeX will take the number of the figure labelled `fig:map` and insert that instead of the `\ref{fig:map}` command. The benefit of this is that all figures (and tables and equations) are numbered implicitly, which means that if

you rearrange figures the labels will be assigned new appropriate numbers and so will all you references in the text. You simply do not have to worry.

There are a set of specific reference commands. `\eqref{}` takes only labels from equations but you can just as easily use the `\ref{}` command. The `\pageref{}` will display the page number where your label is defined. This means that if you have labelled your map somewhere and wish to refer to it, not by its figure number but the page on which it occurs, then you should use `\pageref{fig:map}`.

Finally, LaTeX comes with a cross-referencing system for citations and reference lists. I will not go into any details here because there is more to the system than just cross-referencing. Please refer to section 8.13 for a detailed description. At this point it is sufficient to know that the `\cite` command exists for literature references coupled to a reference list.

3.12 Special commands

LaTeX has several built in commands that perform tasks you want to use. Here is a list of ones we will look more carefully at

Numbered	Unnumbered
`\today`	provides current date
`\the`	allows typesetting of internal variable values from LaTeX
`\year`	variable containing current year
`\day`	variable containing current day
`\month`	variable containing current month
`\footnote{}`	typesets footnote
`\noindent`	cancels indentation of a new paragraph
`\marginpar{}`	adds text in the margin where the command occurs

LaTeX can take information from the computer and display in documents. The most common command for this is `\today` which displays the current date. When you created the title of the document with `\maketitle` LaTeX implicitly uses the `\today` command to add a date to the title. You can use the command anywhere if you want to have the current date in your document but do not confuse this with a fixed date, it will be updated every time you run your document.

A usually little known but powerful command is `\the`. This command takes a value from some parameter within LaTeX and displays it as text in your document. Since LaTeX has variables for year, month and day called (of course), `\year`, `\month` and `\day`, it is possible to extract that information into the text by using the combination of commands `\the\year` etc. But, this will only yield the numbers, not, for example, the month in text. It is however, possible to program LaTeX to convert the number to text.

Footnotes are rarely used in scientific articles but is of course an integral part of any text. LaTeX has a footnote command that will take the text you enter as an argument and place it at the bottom of the page like in the example here[1]. To create a footnote you simply add the command `\footnote{}` right where you want the reference to occur. LaTeX will insert an index number and add the footnote at the bottom of the page automatically.

The `marginpar` command is useful if you want to, for example, add visual reminders to your self in the text. I find it useful to define a new command (see section 10) based on the `marginpar` where the type size is smaller and perhaps also in some colour to make it more visible.

Do it

- Try the different special commands.

4 Tabular information

Typesetting good tables is not easy anywhere. Quite often we see tables made to look like a heap of boxes. This is not good type-setting practise. We will thus focus on simple publication quality tables.

[1] This is footnote text entered in the paragraph on footnotes above

In LaTeX, the basic environment for creating tables is called tabular and a simple table would be made as follows (with output to the right):

```
\begin{tabular}{l c c}
\hline
1 &2 &3 \\
\hline
A &B &C \\
a &b &c \\
\hline
\end {tabular}
```

1	2	3
A	B	C
a	b	c

This is neither sophisticated or beautiful but shows the principles. Obviously the tabular uses a `\begin`–`\end` structure. After the `\begin{tabular}` is a set of curly braces containing the letters `l c c`. These letters tells LaTeX how to align the content in the three columns, first column `l` for left-adjusted and the other two `c` for centred. There is of course an `r` for right-adjusted columns.

The core of the table contains rows such as for example `1 &2 &3 \\`. We can see that the ampersand is used to mark columns 2 and 3 but is not used for column 1. This is how LaTeX uses the ampersand in the tabular environment, it indicates a jump, tab, to the next column. The line ends with a double backslash which tells LaTeX that the line is complete and to switch to a new line. All lines of data in a table must end with the double backslash. I have also inserted some horizontal lines using the `\hline` command. This code does not take a double backslash to mark that it is on a separate line.

This example shows us a few things worth noting. First, that a table should be as simple as possible when it comes to lines, this is good typography. Second, LaTeX will determine the width of columns and the width of the table based on the content. This is of course fine as long as our data and column headers are simple and short.

By loading an extension called `booktabs` (see section 8.3) you have access to a series of additional lines, called rules that will help you organise more complex tables. In the following example we use the non-native commands `\toprule`, `\midrule`, `\bottomrule` and `\cmidrule`. They are all self-explanatory except the last.

`\cmidrule[]()[]{}` takes three arguments (note the different types of brackets used). The first specifies the thickness of the line and the second if it should be trimmed. Both the thickness and trim arguments are optional. If not trimmed, the lines will go the full width of the columns and possibly meet with neighbouring lines. Trimming can be achieved by entering `l` for left trim and `r` for right trim or both into the argument. The third argument takes the column numbers across which the rule should span.

As an example we can look at this small but relatively complex table.

Year	b_w	H_{ss}		$b_w(k)$		S_{wi}	$b_c(k)$		$b_a(k)$	
		29	28S3	29	28S3		29	28S3	29	28S3
	(m w.e.)	(m)		(m w.e.)		(%)	(m w.e.)			
						6	0.06	0.06	0.09	0.12
1997/98	1.35	2.7	2.4	0.03	0.06	7	0.07	0.07	0.10	0.13
						8	0.08	0.09	0.11	0.14
						6	0.06	0.08	0.09	0.13
1998/99	1.33	2.9	2.6	0.03	0.06	7	0.07	0.09	0.10	0.15
						8	0.08	0.10	0.11	0.16

was created by

```
{\begin{tabular}{l c c c c c c c c c}
\toprule
Year &$b_w$ &\multicolumn{2}{c}{$H_{ss}$} &\multicolumn{2}{c}{$b_w(k)$}
      &$S_{wi}$ &\multicolumn{2}{c}{$b_c(k)$} &\multicolumn{2}{c}{$b_a(k)$}\\
\cmidrule(rl){3-4} \cmidrule(rl){5-6} \cmidrule(rl){8-9} \cmidrule(rl){10-11}
        & &29 &28S3 &29 &28S3 & &29 &28S3 &29 &28S3\\
        &(m w.e.) &\multicolumn{2}{c}{(m)} &\multicolumn{2}{c}{(m w.e.)}
     &(\%) &\multicolumn{4}{c}{(m w.e.)}\\
\midrule
 & & & & & &6 &0.06 &0.06 &0.09 &0.12\\
1997/98 &1.35 &2.7 &2.4 &0.03 &0.06 &7 &0.07 &0.07 &0.10 &0.13\\
```

```
          &     &    &    &     &     &8 &0.08 &0.09 &0.11 &0.14\\
          &     &    &    &     &     &6 &0.06 &0.08 &0.09 &0.13\\
1998/99 &1.33 &2.9 &2.6 &0.03 &0.06 &7 &0.07 &0.09 &0.10 &0.15\\
          &     &    &    &     &     &8 &0.08 &0.10 &0.11 &0.16\\
\bottomrule
\end{tabular}
```

An important command we encounter here is the `\multicolumn{}{}{}` command. The command places a single entry across any number of columns. It takes three arguments. The first is the number of columns it should span. The second is the alignment we want. The third is the text that should be placed across the multiple columns.

Do it

- Typeset some of your own tables. Start with a simpler table.

5 Graphics

To include graphics files into a LaTeXdocument we need to add a set of macros that can accomplish this; adding graphics is thus not part of generic LaTeX. To enable graphics we need to add a package called `graphicx` (section8.8) by adding:

```
\usepackage{graphicx}
\end{figure}
```

to our document preamble. this then allows us to use the command

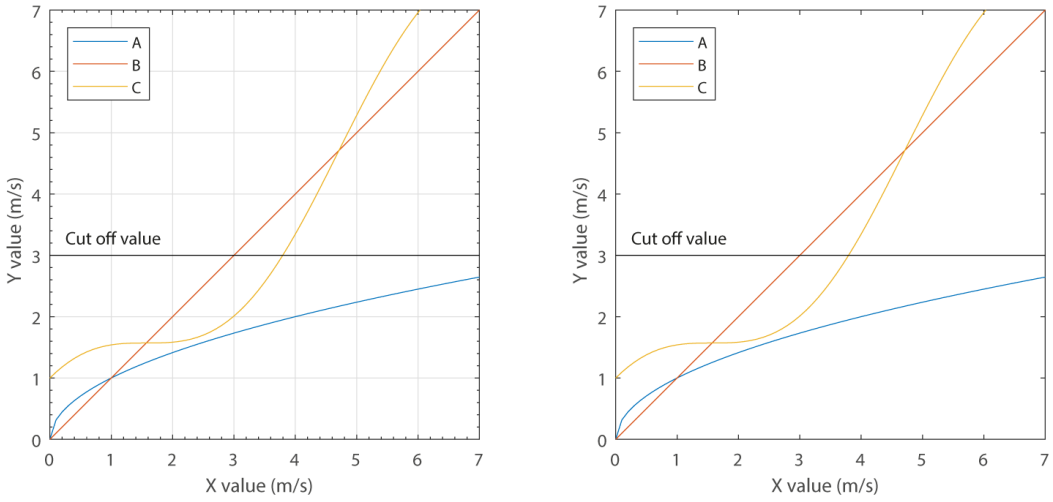
```
\includegraphics{filename}
\end{figure}
```

to include a graphics file.

In overleaf I prefer to keep included graphics separate from other files belonging to the project so I usually add folder called `pics` to the project and add my graphics files there. So when I want to use a graphics file I will write, for example:

```
\includegraphics{pics/BasicPlot.png}
```

to yield the following:



It is also possible to add the command`\graphicspath{ }` to the preamble and specify the paths where LaTeX should look for graphics. So in my case I can add

```
\graphicspath{{pics/}
```

to the preamble and then skip the path when providing the graphics file name in the `\includegraphics` command. It is also possible to avoid providing the grahics file extensions in the `\includegraphics` command by placing the `\DeclareGraphicsExtensions` command in the preamble to define graphic file extensions, such as:

```
\DeclareGraphicsExtensions{.pdf,.png,.jpg}
```

The `\includegraphics` command can handle JPEG, PNG, and PDF when we compile with pdfLaTeX. Note that TIFF bitmap files cannot be included directly but it is possible if we include the TIFF in a pdf file. This can be accomplished by either opening the file in a graphics software that can save the graphics in pdf format or by using a pdf software to 'convert' the TIFF image to a pdf. This process may require some trial and error to ensure that the TIFF file is reproduced correctly in the conversion. To learn more about LaTeX graphics I recommend visiting this TeX.stackexchange post.

6 Floats; tables and figures on the run

LaTeX provides special two environments for figures and tables that have the property that they can be moved by LaTeX from the location where you have entered them to a typographically preferable place. This feature is called a float because they can 'float around'. Many beginners can find this feature irritating, not knowing exactly before hand where the figure may appear. But, in this case, LaTeX has much better knowledge of good layout and typesetting than the average user so the irritation is just a lack of knowledge.

To make a float you use the two environments `figure` and `table`. For a figure you would use the following code

```
\begin{figure}
\includegraphics{filename}
\caption{This is the figure caption text}
\label{fig:x}
\end{figure}
```

The figure environment should contain three commands. First is a command `\includegraphics{}` which brings in a graphics file into the document. We will discuss this command in more detail in section 8.8. Second is a new command called `\caption` which produces a figure caption using the text you enter as the argument to the command. This caption should always be typeset below the figure graphics, which is why we place the commands in that order, and the figure–caption pair will never separate across pages etc. Third is the `\label` command which allows you to refer to the figure using the label name (see section 3.11).

A table follows the same basic recipe but the table itself is made up of a tabular environment (section 4)

```
\begin{figure}
\caption{This is the figure caption text}
\label{tab:x}
\begin{tabular}{c c}
   . . .
\end{tabular}
\end{figure}
```

In this case we place the caption before the tabular environment because the table caption must go above the table. We also have a label working the same way as for the figure environment.

Creating these floats is thus quite simple. It is, however, possible to influence the placement of the floats in the document. To do this, we can add an optional argument to the `\begin{}[]` command. This optional argument can be one or a combination of

Option	Action
h	*h*ere, approximately where it occurs in the source text
t	*t*op of the page
b	*b*ottom of the page
p	on a special floats *p*age
!	Override internal parameters for 'good' float position
H	*H*ere, precisely at the location in the code, similar to !ht[1]

[1] This requires a package called 'float', section 8.6

The way float placement works has been described by Frank Mittelbach on TeX StackExchange. and I will summarise the flow here.

When a float is encountered, LaTeX attempts to place it immediately according to its rules. If this does not succeed, then LaTeX places the float into a holding queue to be considered when the next page is started. Once a page is completed, LaTeX tries to place remaining floats in the holding queue as best as possible. To do this it will first try to generate as many float pages as possible. If this is not possible, it will try to place the floats into top and bottom of pages. It looks at the remaining floats and either places them or defers them to a later page. After that, it starts processing the text for the page. In the process, it may encounter further floats and the process goes on. If the end of the document is reached before the queue is emptied, LaTeXwill add pages and dump remaining floats there.

As you can tell the process is quite involved and never random. If there is need to try to create a page of floats inside a document, you can use the command `\clearpage` which will empty the queue onto pages before continuing with the document.

The option `H` is not generic LaTeX and requires additional work which we will discuss in section 8.6.

Do it

- Test the different float environments with a simple figure and table and change the positioning arguments to see their effects. Note that you need to have a few pages of text with plenty of paragraph breaks to be able to really test the floats. I recommend loading the `lipsum` package (section 8.12) and adding text by typing `\lipsum[n]` where n can be a number or a range (e.g. `2-5`). Check the package documentation for details; lipsum is useful if you need text to see how a layout works.

7 Mathematical typesetting

Since mathematics is one of LaTeX original strong points there are too many introductions to count on writing mathematics out on the Internet. I will therefore not try to add another one to the mix but draw up some basics that will allow you to go along way and form a basis for your own research. As we saw in the quick start, mathematics come in two flavours, in line and display mathematics. The same commands are used in both cases in order to produce the mathematical notation so we will first focus on the different forms for displaying mathematics and then onto the details of how to create the equations.

The basic display mathematics environment is `equation`. As we have seen earlier it works just like any other environment

```
\begin{equation}\label{eq:x}
a^2 = b^2 + c^2
\end{equation}
```

$$a^2 = b^2 + c^2 \tag{2}$$

In this example I have also added the label which means you can now cross-reference the equation number as `\ref{eq:x}` in the text. There is also an environment for sets of equations

```
\begin{eqnarray}
c_1 &=& a_1x + b_1x \label{eq:a} \\
c_1 &=& a_2x + b_2x \label{eq:b}
\end{eqnarray}
```

$$\begin{aligned} c_1 &= a_1x + b_1x && (3)\\ c_1 &= a_2x + b_2x && (4) \end{aligned}$$

The alignment is set up by the ampersands in the two equations, in this case to make sure the equal signs line up. You can also use the `eqnarray` to take care of longer equations

```
\begin{eqnarray}
y = a_1x &+& b_1x + c_1x + d_1x
          \nonumber \\
  &+& e_1x + f_1x \label{eq:y}
\end{eqnarray}
```

$$\begin{aligned} y = a_1x &+ b_1x + c_1x + d_1x \\ &+ e_1x + f_1x \end{aligned} \tag{5}$$

With AMSLaTeX (see below), you can access another environment to do what we did with `eqnarray` above. This is the `align` environment

```
\begin{align}
y = a_1x &+ b_1x + c_1x + d_1x
         \nonumber \\
  &+ e_1x + f_1x \label{eq:y}
\end{align}
```

$$y = a_1x + b_1x + c_1x + d_1x + e_1x + f_1x \tag{6}$$

In this case the alignment is much better. This is but one example of how using AMSLaTeX improves your capabilities.

When you build equations, you need to know the commands that produce all the different mathematical symbols and notation. A good source is the American Mathematical Society (AMS). It is worth mentioning at this stage that while LaTeX does excellent mathematical type setting, AMS has improved on the capabilities through their AMSLaTeX extension. If you just need to type set a few formulas and ordinary equations, basic LaTeX should be more than sufficient. But if you need some more special features, they are almost certainly available through the AMSLaTeX extension.

Now turning to formulating an equation. When you type in an equation, you do generally not need to worry about spacing. As we noted earlier all letters will be typeset in mathematical italics, this is the norm for mathematical variables. The italics only applies to Latin letters, Greek letters are not italicised. This means that if you want to type in a function such as sine you need to be careful as this example shows

```
\begin{eqnarray}
\tau &=& \rho gh sin\alpha \\
\tau &=& \rho gh \sin\alpha
\end{eqnarray}
```

$$\tau = \rho ghsin\alpha \tag{7}$$
$$\tau = \rho gh \sin\alpha \tag{8}$$

In the upper row we typed in the letters 's i n' and they came out in italics but with a similar spacing as between the variables g and h In the second row we used the mathematical command `\sin`. In the first case, 'sin' is treated as three variable names. LaTeX, therefore has all the mathematical functions established as commands so that they are type set correctly. Adding letters after each other is just interpreted as a series of variables multiplied by each other. This has a bearing on those who persist to form variables more or less at abbreviations such as ET for evapotranspiration or worse $NDVI$ for Normalized Difference Vegetation Index

$$NDVI = \frac{NIR - VIS}{NIR + VIS} \tag{9}$$

where NIR is the near infrared band and VIS is the visible. This clearly does not look good partly because of the way LaTeX handles text i mathematical mode but also because it is a poor way to defined variable names. So let this be a warning when you define your own variables and equations.

Despite the poor look of equation 9 we see that we can produce division. This is easily accomplished with the `\frac{}{}` command. `frac` takes two arguments. In the first is everything that goes in the numerator and in the second, everything that goes in the denominator.

```
\begin{equation}\label{eq:NDVIb}
I_\mathrm{NDV} = \frac{E_\mathrm{IR}
               - E_\mathrm{VIS}}
                {E_\mathrm{IR}
               + E_\mathrm{VIS}}
\end{equation}
```

$$I_{\mathrm{NDV}} = \frac{E_{\mathrm{NIR}} - E_{\mathrm{VIS}}}{E_{\mathrm{NIR}} + E_{\mathrm{VIS}}} \tag{10}$$

In this example we can see how the `\frac` command works but we can also see that it is possible to type in the equation in almost any form, in this case to try to make it structured, sine it has no consequence for the output. You can also see a new command `\mathrm{}` which produces regular text inside mathematical mode.

When you use the `frac` command and you need to put a parenthesis around the equation, you can obtain scalable parenthesis using the `\left` and `\right` commands

```
\begin{equation}\label{eq:NDVIb}
I_\mathrm{NDV} = \left(
          \frac{E_\mathrm{IR}
              - E_\mathrm{VIS}}
               {E_\mathrm{IR}
              + E_\mathrm{VIS}}
           \right)
\end{equation}
```

$$I_{\mathrm{NDV}} = \left(\frac{E_{\mathrm{NIR}} - E_{\mathrm{VIS}}}{E_{\mathrm{NIR}} + E_{\mathrm{VIS}}} \right) \qquad (11)$$

The `\left` and `\right` commands must pair up. You can use `( )`, `[ ]` and `{ }` and also `|` with these commands. It is also possible to set only one side of the pairs you should use a period for the side you want to omit: `\left. \right.` . Note that when you use `\left` and `\right`, both must be included.

Making mathematical type setting is otherwise mostly finding the right commands to obtain what you want. There are many lists floating around that summarizes these so I will not take up space here. I would argue that mathematics is one of the easiest things you can do in LaTeX.

Do it

- Typeset some of your own equations or try to reproduce an equation out of an article or book, or both.

8 Extending the functionality of LaTeX, packages

Because LaTeX is OpenSource, many enthusiasts contribute to LaTeX functionality by providing extensions, so called *packages*. The the Comprehensive TeX Archive Network (CTAN) contains packages and their documentation and is thus a good source for information. I strongly recommend you to look at any package documentation posted on CTAN before using any package.

Out of all the available packages, only a small fraction will likely be of use to you. A well known effect of being new to LaTeX is to go 'package crazy' and load almost anything. If this happens to you, then it will pass; but more severely, you may end up with unforeseen problems due to conflicts between certain packages. Some packages are also tailored to solve a particular problem for a given situation that may not apply to you. Much of this is stated in the package documentation.

In the following sections I will provide information on some packages that vary from a *must* to being of general interest. A package is loaded by adding the command `\usepackage` to the preamble:

```
\usepackage[]{}
```

As can be seen the command has a normal argument field (denoted by the curly braces) and an optional input field (denoted by the square brackets). Many packages can be loaded without any options which means you can ignore the square brackets. Other packages require or has options for providing specific features. Such options are described in the package documentation.

I have ordered the packages alphabetically.

8.1 amslatex

The `amslatex` package provides many improvements and extension to LaTeX native mathematics capabilities. The package is loaded by

```
\usepackage{amslatex}
```

There is no point to try to describe the package here. You need to carefully look at the AMSLaTeX information at the American Mathematical Society.

8.2 babel

The `babel` package contains ability to switch many built in features that are language dependent to a language other than English. If you wish to write a document n Swedish you simply load the package as

```
\usepackage[swedish]{babel}
```

The document will now be shown with many small differences to the English version. Certain headings such as 'Contents' and 'References' will be in your chosen language, dates will be written in the standard of your language, etc. Please refer to the babel package documentation for details.

With babel it is also possible to write multilingual documents and switch between languages mid-document.

8.3 booktabs

The `booktabs` package is a small package that improves table formatting. The package is loaded by

```
\usepackage{booktabs}
```

It is the `booktabs` package that allows you to use the commands `\toprule`,`\midrule` and `\bottomrule` in tables. I have consistently used these for the tables in this text unless stated otherwise. The package provides a few other commands as well but the `booktabs` package documentation is well worth reading because it outlines ideas around good practises in table design.

Do it

- Try the different rules in your tables to replace the `\hline`.

8.4 caption

The `caption` package provides easy tools for changing how the figure and table captions are formatted but also extends on the native functionality. The package is loaded by

```
\usepackage{caption}
```

In the standard layout the the captions start with 'Figure 1:' and 'Table 1:' Since this part of the caption is created automatically, you cannot manually change the colon to a period, which is the format required by most journals, in the text. The following commands changes the formatting of the caption start:

```
\renewcommand{\figurename}{Figure.}
\renewcommand{\tablename}{Table.}
```

Note that you can enter whatever formatting you want in the second argument. A perhaps quicker way is to load the package with options for fonts

```
\usepackage[labelsep=period,font={small,it},justification=justified]{caption}
```

With this you will receive a caption written in 'small' size italics font, and with the words Figure and Table followed by a period. This is shown in 'Figure' 1 (which is just a caption). Please refer to the `caption` package documentation for more details.

Figure 1. This is a trial caption to see how the different changes will look in the case of a caption in a float

Do it

- Use the `caption` package to change the look of your captions.

8.5 dcolumn

The `dcolumn` package provides means to define new alignments for tabular environments. The package is loaded by

```
\usepackage{dcolumn}
```

To provide an example of what can be accomplished with `dcolumn` we can define a new column type called '.' (period) using the command `\newcolumntype`

```
\newcolumntype{.}{D{.}{.}{-1}}
```

This new column type will align numbers on the period. As you can see the command takes many arguments. The first argument is the name of the column type ('.'). The second argument is divided into three parts, the first part indicates what character should be used for the alignment (the period), the second part shows what symbol should be used for the separator (usually the same as the first), the third part indicates how many decimal places should be shown. With `-1` as the third argument, the column will be centred on the decimal point. An example

```
\begin{tabular}{. . .}
\toprule
45.73  &43.894  &5.463\\
33 &0.0001   &0.02\\
7.76 &.2 &9.75\\
\bottomrule
\end{tabular}
```

45.73	43.894	5.463
33	0.0001	0.02
7.76	.2	9.75

And then the same table but with a different column type called ':' (colon) which has the `\cdot`) as decimal separator and 4 decimal places (the maximum number of decimals in the table; `\newcolumntype{:}{D{.}{\cdot}{4}}`)

```
\begin{tabular}{: : :}
\toprule
45.73  &43.894  &5.463\\
33 &0.0001   &0.02\\
7.76 &.2 &9.75\\
\bottomrule
\end{tabular}
```

45·73	43·894	5·463
33	0·0001	0·02
7·76	·2	9·75

As you realize you can do a lot with this package and define your own numeric column types. The `dcolumn` package documentation is quite brief so experimentation is best way to learn this package.

The `siunitx` package also has table alignment support which may be useful to study.

Do it

- Try the `dcolumn` package by making your own definition or changing parameters in examples given above and using the new column definitions in your own table.

8.6 float

The `float` package adds a placement option `H` to the `figure` and `table` environment (see section 6). The option is a stronger placement directive the the native options. The package is loaded by

```
\usepackage{float}
```

8.7 geometry

The `geometry` package provides an easier interaction with LaTeX page layout settings. The package is loaded by

```
\usepackage[a4paper]{geometry}
```

In this example, the option `a4paper` has been added in order for margins etc. to be scaled correctly for that paper size. This is the minimum of what you need to add to your preamble. It is worth noting that the `documentclass` command also takes `a4paper` as an argument but I prefer to use it in the call to `geometry`.

If you need to change your margins, you do this by simply using the following

```
\usepackage[a4paper,left=35mm,right=30mm,top=30mm,bottom=25mm]{geometry}
```

which just happens to be the call made for this document. This sets the left and right margins to 25 mm wide and the top and bottom margins to 30 mm. You should study the `geometry` package information for more details

Do it

- Use the arguments to the `geometry` package to change margin widths, paper size etc.

8.8 graphicx

The `graphicx` package is a *must*. Without this package you will not be able to include graphics into your document as we mentioned in section 5. The package is loaded by

```
\usepackage{graphicx}
```

in the preamble of your document.

When you have `graphicx` loaded into your document you have access to the command for including graphics, appropriately named `\includegraphics`. The command come with several optional arguments and looks as follows

```
\includegrphics[width=0.5\textwidth,angle=45]{fname.ext}
```

The file name is provided as the main argument in curly braces. It is not necessary to type the extension. It is also possible to provide a path to the graphics file if it is not in the folder where your LaTeX document resides. valid graphics formats are JPEG, PNG, and PDF. Note that SVG and TIFF are not supported directly but can be included by converting the graphics to PDF.

The optional arguments in the example provide LaTeX with the information that the figure should be reproduced with a width that is 0.5× the width of the text. This type of scaling is convenient but you can also specify a fixed width in, say millimetres, by writing, for example `[width=57mm]`. In addition we have also asked LaTeX to rotate the figure by 45°, which, of course, is unusual. In addition you can scale the figure by its height in a similar way to the width. There is a command `\textheight` that allows you to scale relative to the height of the text area.

If you provide only one of `width=` or `height=` the figure will be scaled proportionally in the other directions. You can scale an image dis-proportionally by providing different scaling factors for width and height.

In section 5 we also introduced the commands

```
\graphicspath{}
\DeclareGraphicsExtensions{}
```

that allow us to specify paths to graphic files as well as expected extensions of graphics files. If used, it will not be necessary to provide such information in the `BS includegraphics` command. The commands need to be placed in the preamble of the document.

Do it

- Add a figure to your document and test the scaling and rotation capabilities.

8.9 hyperref

The `hyperref` package enables you to make workable links in your pdf output. The package is loaded by

```
\usepackage{natbib}
```

When you use `hyperref` all the cross-links in your document becomes clickable and allows you to move around in the document. The table of contents is also clickable by default as are any references. In addition, you have access to the command `\href{}{}` which allow you to produce links to web URLs outside of your document. To show an example how this works we can set up a link to TEX StackExchange

```
\href{http://tex.stackexchange.com}{\TeX\ StackExchange}
```

The command takes two arguments. The first is the complete URL and the second is the text that will be highlighted as a link in the text.

You can modify the way links are shown by the command `hypersetup`

```
\hypersetup{
    pdftoolbar=true,        % show Acrobat's toolbar?
    pdfmenubar=true,        % show Acrobat's menu?
    pdffitwindow=true,      % window fit to page when opened
    pdfstartview={FitH},    % fits the height of the page to the window
    pdftitle={title},       % title
    pdfauthor={name},       % author
    pdfsubject={subject},   % subject of the document
    pdfkeywords={keyword1} {key2} {key3}, % list of keywords
    pdfnewwindow=true,      % links in new window
    colorlinks=true,        % false: boxed links; true: colored links
    linkcolor=black,        % color of internal links
    citecolor=SUBlue,       % color of links to bibliography
    filecolor=blue,         % color of file links
    urlcolor=blue           % color of external links
    }
```

Note that you do not need to list all settings, simply the ones you wish to change from default values. Just remember to separate them by a comma. With this setting, the links to external ages are blue, internal links are black except citations which are in a blue defined as `SUblue` (see section 8.16). As you can tell, the `hypersetup` also dictates how the pdf should be opened by Acrobat reader. You should refer to the `hyperref` package documentation for more information.

Do it

- Try the `hyperref` package by adding some urls and also changing the parameters in the `hypersetup` command.

8.10 inputenc

While scientific writing is primarily done in English and LATEX is made with English in mind it is often necessary to include words, for example, place names with accented letters in the text. Generic LATEX does, for example, not support the å, ä, and ö on the Swedish keyboard. You can of course create these letters through LATEX accenting commands, in this case by typing `\aa`, `\"a` and `\"o`, respectively. If this happens a few times it is not a major issue. There is however a way to ensure that what is on your keyboard is also possible to handle in LATEX.

The package `inputenc` (short for 'input encoding') allows LATEX to use an extended encoding, that is beyond the letters and numbers of the English keyboard. this is not limited to Swedish letters but to a large variety of other accented letters. The package should be called with the option `utf8`. The UTF8 encoding allows a computer to manage over a million different characters which includes all characters in modern Unicode type faces. By adding the following to your preamble, you have full use of your font character set. The package is loaded by

```
\usepackage[utf8]{inputenc}
```

Note that this package is not necessary when you use XeLATEX to compile your document.

8.11 lineno

The `lineno` package does one thing, it allows you do add line numbers to, for example a manuscript. Some journals, for example, ask for such manuscripts. The package is loaded by

```
\usepackage{lineno}
```

You turn the line numbers on by inserting the command `\linenumbers` where you want the numbers to start. You can stop line numbers by placing the command `\nolinenumbers` where you want them to stop. You should look at the `lineno` package documentation for more details on different options.

Do it

- Try `lineno` package in your document

8.12 lipsum

The `lipsum` package produces a body of dummy text in your document. The text is commonly used in type setting to assess the look of the layout design and consists of a text body in improper Latin. By adding a number or range of numbers as an optional argument (1–4 in the example below) it is possible to display a specific number of paragraphs in your text. the benefit of this is that you can quickly add some text to see effects of your type setting design decisions without having to type anything into your document. The package is loaded by

```
\usepackage{lipsum}
```

and to add text to your document you add, for example,

```
\lipsum[1-4]
```

to where you want the text to appear.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

8.13 natbib

The `natbib` package is a must if you need Harvard style author-year referencing system in your work. The package is loaded by

```
\uepackage{natbib}
```

I will not dwell on `natbib` functionality here since we will go into much more detail when discussing referencing in section 9. What may be of more interest here is a section of code that I recommend you to use in conjunction with the `natbib` package. This code referred to as `natbibspacing.sty`

```
\newdimen\bibspacing
\setlength\bibspacing\z@
\renewenvironment{thebibliography}[1]{%
 \bibfont\bibsection\parindent \z@\list
   {\@biblabel{\arabic{NAT@ctr}}}{\@bibsetup{#1}%
    \setcounter{NAT@ctr}{0}}%
    \ifNAT@openbib
      \renewcommand\newblock{\par}
    \else
      \renewcommand\newblock{\hskip .11em \@plus.33em \@minus.07em}%
    \fi
```

```
    \sloppy\clubpenalty4000\widowpenalty4000
    \sfcode`\.=1000\relax
    \let\citeN\cite \let\shortcite\cite
    \let\citeasnoun\cite
            \itemsep\bibspacing %
            \parsep\z@skip %
 }{\def\@noitemerr{%
  \PackageWarning{natbib}
     {Empty `thebibliography' environment}}%
  \endlist\vskip-\lastskip}
%---------------------
\setlength{\bibspacing}{0pt}
```

You may be able to load this code by adding the name to the call for `natbib`

```
\uepackage{natbib,natbibspacing}
```

but only if it exists in your LaTeXdistribution. Otherwise, you can enter the code into your preamble after you have loaded `natbib`.

The `natbibspacing` is an example of pure code and may get an insight into how much details can be changed but also that it involves learning LaTeX as a programming language in detail. Fortunately, others do these things for you in the community.

8.14 siunitx

The `siunitx` package is a very complex package that deals with how to write numbers and units (with focus on SI units). The package is loaded by

```
\usepackage{siunitx}
```

This package may to many seem a bit over the top. Let us look at some of the core ideas. Let us say we want to write a complex unit such as square volt cubic lumen per farad ($\mathrm{V}^2\,\mathrm{lm}^3\,\mathrm{F}^{-1}$). This seems like a disaster waiting to happen. With the `siunitx` package and its command `\si{}` we will get the correct units through the following

```
\si{\square\volt\cubic\lumen\per\farad}
```

As you can see all the words we used to describe the unit in text is available as a command and the result is a perfect SI unit form. If you need to add a number to the unit there is a second command `\SI{}`

```
\SI{203}{\kilo\gram\metre\per\second}
```

which yields $203\,\mathrm{kg\,m\,s^{-1}}$. Now why is this good? There are several benefits. First it becomes quite clear what your unit is. Second, the spacing between numbers and units and between the units themselves are constant and of accurate length. It is true that you end up writing a lot but the benefits definitely outweighs the disadvantages.

The package also introduces a new alignment for the tabular `environment`. The new `S` alignment is more advanced than what is accomplished by the `dcolumn` package because it can take into consideration material that should go in front of or after a number.

The package contains much more on representing data in text so you should study the extensive `siunitx` package documentation.

Do it

- Try `siunitx` package by typing in some units that you are familiar with.

8.15 tikz/pgf

The `tikz/pgf` package is actually two packages. Tikz is a front end for the drawing capabilities set up by `pgf`. TikZ is a recursive name, *Tikz ist kein zeichenprogram*. The package is loaded by

```
\usepackage{tikz}
```

The tikz package documentation is 1320 pages (at the time of writing, version 3.1.6) and hence extremely detailed. The manual doubles as a tutorial and dictionary. We will briefly look at two examples that shows you what is needed and what can be done. It is noteworthy that this is still just scratching the surface of what can be done. Please visit TEXsample.net for numerous examples of Tikz output.

The first example shows a flow diagram for data collection at Tarfala Research Station.:

```
\tikzstyle{block} = [rectangle, draw, fill=blue!20,
    text width=5em, text centered, rounded corners, minimum height=4em]
\tikzstyle{rblock} = [rectangle, draw, fill=red!20,
    text width=5em, text centered, rounded corners, minimum height=4em]
\tikzstyle{line} = [draw, -latex']

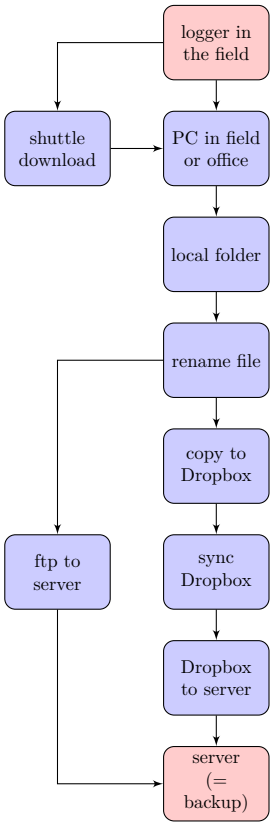
\begin{tikzpicture}[node distance = 2cm, autoscale=0.7,
                    every node/.style={scale=0.7}]

    \node [rblock] (logger) {logger in the field};
    \node [block, below of=logger] (pc) {PC in field or office};
    \node [block, below of=pc] (infolder) {local folder};
    \node [block, below of=infolder] (rename) {rename file};
    \node [block, below of=rename] (copy) {copy to Dropbox};
    \node [block, below of=copy] (sync) {sync Dropbox};
    \node [block, below of=sync] (toserver) {Dropbox to server};
    \node [rblock, below of=toserver] (server) {server\\ (= backup)};

    \node [block, left of=pc, node distance=3cm] (shuttle) {shuttle download};
    \path [line] (logger) -|  (shuttle);
    \path [line] (shuttle) -- (pc);

    \node [block, left of=sync, node distance=3cm] (ftp) {ftp to server};
    \path [line] (rename) -|  (ftp);
    \path [line] (ftp) |- (server);

    \path [line] (logger) -- (pc);
    \path [line] (pc) -- (infolder);
    \path [line] (infolder) -- (rename);
    \path [line] (rename) -- (copy);
    \path [line] (copy) -- (sync);
    \path [line] (sync) -- (toserver);
    \path [line] (toserver) -- (server);
\end{tikzpicture}
```



The second example is a figure I made for describing the energy fluxes at the ice surface on a glacier.

```
\begin{tikzpicture}[>=latex,scale=0.8, every node/.style={scale=0.8}]
\fill[top color=cyan!20,bottom color=black!30]
                      (0,-1) -- (0,0) -- (10,0) -- (10,-1);

\draw[thick] (0,0) -- (10,0);% Ground

\draw[<->] (5,-.9) -- (5,0);% Geothermal heat
 \node at (5,-0.57) [right] {$G$};

\draw[decorate,decoration=snake,segment length=11,->,blue!50!green]
                       (0,2.9) -- (0.8,0); % Incoming SW radiation
 \node at (0.15,2.5) [right] {$I_S\downarrow$};
\draw[dashed,decorate,decoration=snake,segment length=11,->,blue!50!green]
                       (0.8,0) -- (1.7,2.9);
\draw[dotted,decorate,decoration=snake,segment length=11,->,blue!50!green]
                       (1.2,0) -- (2,2.9); % Outgoing SW radiation
 \node at (1.9,2.5) [right] {$I_S\uparrow$};
\draw[dotted,blue!70!white] (0.8,0) -- (1,-0.4) -- (1.2,0);

\draw[decorate,decoration=snake,segment length=18,->,red!70!black]
                       (3,2.9) -- (3,0); % Incoming LW radiation
 \node at (3,2.5) [right] {$I_L\downarrow$};
\draw[decorate,decoration=snake,segment length=18,->,red!70!black]
                      (4,0) -- (4,2.9); % Outgoing LW radiation
 \node at (4,2.5) [right] {$I_L\uparrow$};

\draw[decorate,decoration=coil,->] (5.5,2.9) -- (5.5,0); % Latent heat
 \node at (5.5,2.5) [right] {$L$};

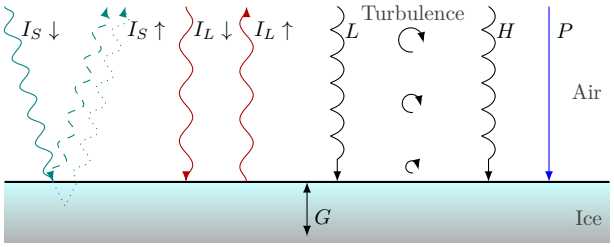
\draw [->] (6.75,.15)  arc (280:0:.1cm)  -- +(283:0.05cm); % Turbulence
\draw [->] (6.75,1.15) arc (280:0:.15cm) -- +(283:0.05cm);
\draw [->] (6.75,2.15) arc (280:0:.2cm)  -- +(283:0.05cm);
\node at (6.75,2.8) [gray!50!black] {Turbulence};

\draw[decorate,decoration=coil,->] (8,2.9) -- (8,0); % Sensible heat
 \node at (8,2.5) [right] {$H$};
```

```
\draw[->,blue] (9,2.9) -- (9,0); % Precipitation heat
 \node at (9,2.5) [right] {$P$};

\node at (10,-.6) [left,gray!50!black] {Ice};
\node at (10,1.5) [left,gray!50!black] {Air};

\end{tikzpicture}
```



The examples above require certain tikz libraries to be loaded in the pre-amble. Which are required depends on what sort of graphic elements are required. The manual details this.

```
\usetikzlibrary{arrows,shadows,positioning}
\usepackage{pgfplots}
\usetikzlibrary{decorations.pathmorphing}
\usetikzlibrary{decorations.shapes}
\usetikzlibrary{patterns}
```

Although these examples may be daunting, they indicate the flexibility of LaTeX.

Do it

- Visit the TeXsample.net site and try out a few examples. If possible try to make changes to the code to familiarize yourself with the tikz programming.

8.16 xcolor

The `xcolor` package introduces colour to LaTeX. In the `hyperref` and `tikz` packages we could see colour already and and `xcolor` is in fact loaded already by those packages. The package is otherwise loaded by

```
\usepackage{xcolor}
```

With this package you can access a large number of predefined colours, the basic set is shown in Table 1. These are described in the `xcolor` package documentation.

Table 1. The basic set of colours provided by the `xcolor` package.

Name	Colour	Name	Colour
red		lightgray	
green		brown	
blue		lime	
cyan		olive	
magenta		orange	
yellow		pink	
black		purple	
gray		teal	
white		violet	
darkgray			

In addition you can define your own colours using the `\definecolor` command. The command takes three arguments, first the name you wish to give to the colour, second the color space (cmyk, rgb etc.) and third the colour combination that makes up the colour. The following define the official Stockholm University colour palette in cmyk space

```
\definecolor{SUBlue}{cmyk}{1.00,0.70,0.00,0.60}
\definecolor{SUOlive}{cmyk}{0.25,0.10,0.60,0.20}
\definecolor{SUSky}{cmyk}{0.35,0.00,0.10,0.00}
\definecolor{SUWater}{cmyk}{0.40,0.15,0.00,0.05}
\definecolor{SUFire}{cmyk}{0.00,0.65,1.00,0.00}
\definecolor{SUSilver}{cmyk}{0.12,0.08,0.08,0.23}
\definecolor{SUGold}{cmyk}{0.30,0.40,0.80,0.15}
```

Table 2 shows the official Stockholm University palette defined in CMYK through the combinations given on the SU web site.

Table 2. The official Stockholm University colours in CMYK.

Colour name SU	LaTeX	Colour	CMYK
Blue	`SUBlue`		1.00 0.70 0.00 0.60
Olive	`SUOlive`		0.25 0.10 0.60 0.20
Sky	`SUSky`		0.35 0.00 0.10 0.00
Water	`SUWater`		0.40 0.15 0.00 0.05
Fire	`SUFire`		0.00 0.65 1.00 0.00
Silver	`SUSilver`		0.12 0.08 0.08 0.23
Gold	`SUGold`		0.30 0.40 0.80 0.15

A useful web resource providing the RGB component define commands corresponding to colour swatches can be found at LaTeX color site.

There are also ways to create hues out of the colours by mixing colours using the command `\color{}`. To make up a hue we specify how much of each to mix

```
\color{blue!40!red}
```

In the example the mix will be 40% green and thus 60% red (since they must sum up to 100%) resulting in the colour of this sentence. The colours blue and red are part of the basic set of colours provided by the `xcolor` package (Table 1).

Do it

- Define and use some of your own colours.

9 Managing references

LaTeX has the capability to handle literature references and generating reference lists in your documents. To accomplish this there a few points we need to cover.

You need to have a data base of the literature you wish to cite in your documents. This means that you should start building a data base of references. There are many reference managers available such as EndNote, JabRef, Mendeley, RefWorks and Zotero to mention a few. The Comparison of reference management software page provides an overview of available tools. What you should look for is that the tool can export BibTeX format, can provide a unique BibTeX 'cite key' (see below) for each entry, is continually supported, supports your platform and, I would suggest, is free and open.

I use and recommend the Open Source Zotero for your reference handling unless you already work with some other tool. With this tool you can save reference information from web pages and article pdfs directly into Zotero and quickly build your reference data base. Zotero works with a cloud based storage that is free up to 300 Mb. Start by signing up for a Zotero account

and download the Zotero software for installation on your computer.You also need to install a connector for your web-browser (Chrome/Firefox/Safari). Finally, you should also install the Better-BibTeX for Zotero plugin in your desktop Zotero software. Just follow instructions to do so.

Zotero allows you to also store pdf files with your references. This will, however, quickly fill your free data base space. I personally save only the reference information and at some point had 2974 references in Zotero which used up 0.3% of my 300 Mb. So the free space will be sufficient for quite some time as long as you store only the reference information and not pdf files.

9.1 Integration of Overleaf and Zotero

When using Overleaf you can link Zotero to Overleaf. This is accomplihed by linking your Zotero account to your Overleaf account in the Overleaf account settings. Once you have completed this you will be able to transfer data directly from Zotero into Overleaf documents.

When you need to reference articles in your data base in a document you first need to obtain a copy of the data base in `.bib` format to reside in your Overleaf project. When you press 'Upload' in your Overleaf project view to add new files, you will see several options in the pop-up Window that appears. One option will be 'From Zotero'. This will allow you to copy your online Zotero data base into you Overleaf project. When you make the selection you will also be asked to name the file. The file name you provide is what needs to go in the `\bibliography{}` command. You then also need to make sure you have the `.bst` file of your choice and the correct file name entered in the `\bibliographystyle{}` command.

When you copy the data base from Zotero to Overleaf, the process may take a little while depending on the size of your data base.

Whenever you update your data base in Zotero, you need to update the copied data base file in your project(s) to access these new references. This is very easy to do. You simply click on the `.bib` file name in your project view. This displays the data base file in Overleaf and you then have the possibility to 'Refresh' the data base. This transfers the latest version of the data base to your project. You then have access to any new items added to your data base.

When you work with the integration between Zotero and Overleaf, the data base will be provided Zotero default cite keys. These have the format First author name–first word of the title–year of publication. If you have set up Better BibTeX to generate a different cite key format in Zotero, that cite key format will revert to the default in the copy process. At least at the time of writing this there is no way to work around this with the direct transfer.

To get your reference data base with your own Better BibTeX cite key format into a document you need to save a copy of the data base from your desktop Zotero tool. Use the Export library function in Zotero and select 'Better BibTeX´ format. You will then get a copy of the data base on your computer that you can upload to your project as a file. If you add new material to the data base you need to redo this procedure to get your data base updated in the project.

To work with your data base in your document to generate references and cross reference with a reference list, you will need a few also need a few additions to your document. The goal is to be able to relate any article you reference in your manuscript with an entry in your data base. We also want all references we added in the document to correspond to a reference in the reference list. We of course also want the reference list to correspond exactly to what we want to reference in the text. All this will be accomplished by LaTeX with the `natbib` package.

9.2 The `natbib` package

When you want to use author-year style referencing, the common style for the Earth Sciences, you need to use the `natbib` package (section 8.13). This package adds many new commands dealing with different types of references.

When you make references in the text you basically need two forms for the reference.

active where the author name is part of the text and the year is within parenthesis as in 'Smith (2013)'

passive where both the author name and the year are within parenthesis as in '(Smith 2013)'

The `natbib` package provides two primary commands for this, `\citet{}` and `\citep{}`, respectively.

As with other cross-referencing the referencing works through labels also known as *cite keys* in the case of references. Each reference must therefore have a unique cite key. With Zotero the cite key will by default be created from the first authors last name, the first word in the title (Ignoring words such as 'The' and 'A') and the year of publication. This default is what you will receive when you link and synchronise Zotero directly with Overleaf and cannot be changed. With Better BibTeX you can change the cite key format. I prefer a format of first author-year-journal. However, to get this format into a document the normal synchronisation will not work and you will need to manually export the data base into your project. So, a reference such as

Smith AZ, 2013. Bad science – surviving in the jungle. *Journal of Long-winded Pseudoscience*, 23, 214–523.

will by default be assigned a cite key `Smith_Bad_2013`. This means you can reference this article in the text as either `\citet{Smith_Bad_2013}` or `\citep{Smith_Bad_2013}`. Either one of these cite commands will enable LaTeX to add the reference to the reference list. Hence you simply add references and the reference list is built for you. Note that when you start typing the cite commands your will get suggestions for what reference to add as soon as you type the first letter in the name. It is thus quite easy to add references and you do not need to memorise cite keys, just author names and possibly parts of the title of the paper.

The `natbib` package has several commands for referencing but the most commonly used are shown here

Command	Result
`\citet{}`	'active' citation 'Smith (2013)'
`\citep{}`	'passive' citation (Smith, 2013)
`\citeauthor{}`	cites only author name, not year
`\citeyear{}`	cites only year, not author name

In all cases the BibTeX key should be entered as argument to the commands.

If you need to reference many authors in a passive reference, you simply add them all as argument separated by a comma. `\citep{Smith_Bad_2013,Day_Two_2014,Knight_Three_2015}` will yield (Smith 2013; Day 2014; Knight 2015).

The `\citet{}` and `\citep{}` commands can also take optional arguments. If we want to have a reference that looks like (e.g., Smith 2013; and references therein) we can write `\citep[e.g.,][and references therein]{Smith_Bad_2013}`. What we want added before the reference goes as the first optional argument and what should be placed after the reference goes in the second. If you do not want one or the other, you just leave the argument empty.

If you make a spelling mistake in the BibTeX key in your document, you will see this as two question marks in the text (??). This is how LaTeX signals such errors. It will also appear as a warning in your log file.

With these commands you should be able to add references to the text but in order for the system to work, you need to provide somewhere from which BibTeX can obtain its information and this is where we have two ways to go, the fully and the semi automated way. I will cover both but there is little point in making the semi automated system your standard.

9.3 The fully automated reference system

The fully automated system handles both the cross-referencing and builds the reference list for you. In order to do this you need to have several things in place. First you need to have your data base as a BibTeX `.bib` file in your project, including all references you need to reference in your document. You also need a file that tells BibTeX how to format your references. This is called a *BibTeX style file* (`.bst`). It is possible to custom make your own style but that is not something that is easily described here. Instead you can browse CTAN's repository for bibliographic styles and try them. Note that you should look for author-year formats. The repository contains many other formats as well.

Journals that take LaTeX manuscripts usually have their own styles that can be downloaded from or come with template files for the journal located on their instructions for authors page. Otherwise, I would otherwise recommend going with the Council of Science Editors (CSE) style. It is a neutral international standard that is simple.

When you have your data base file and your style file, you can start using it in your document. You need to add the following code where you want your reference list to appear.

```
\bibliography{foo}
\bibliographystyle{CSE}
```

The first command `\bibliography{}` tells BibTeX which `.bib` file to look for, your bibliography file. The file name does not need to be written out with the extension `.bib` and you can add a path if your bibliography file is not located with your LaTeX document file. It is also possible to have more than one reference file from which to extract references, which can be useful when collaborating with other authors and each want to add 'their' references. The second command tells BibTeX which bibliographic style that should be used, the name of your `.bst` file here exemplified by the `CSE.bst` (Council of Science Editors).

With all this in place your reference list will be up to date whenever you add a new reference and recompile the document. Every reference in the text will correspond to an entry in the reference list and there will not be references in the list that are not referenced in the text. As mentioned earlier, a prerequisite is that the reference you you reference must be in the data base file and all entries in the data base must have unique BibTeX cite key.

Please remember that any references added in your data base software must be synchronised or uploaded to Overleaf before you can use the new reference.

Do it

- Add a few references to a `.bib` file and cite them (using `natbib` citing commands) using the fully automated method described above.

9.4 The semi-automated reference system

The semi-automated reference system does not make use of a reference manager or an external bibliography file as discussed above. In the semi automated referencing system, LaTeX keeps track of the cross-referencing but you need to manually type in all references the way they should appear. This means that you need to create a reference list at the end of your document following a specific format.

```
\begin{thebibliography}{}

\bibitem[Smith, 2013]{Smith_Bad_2013}
Smith, A.Z., 2013. Bad science -- surviving in the jungle.
\textit{Journal of Long-winded Pseudoscience}, 23, 214--523.

\end{bibliography}
```

As you can see the bibliography has its own environment within which you put all your references. each reference must start with the command `\bibitem[]{}` which takes two arguments. the first argument is the in-text reference, the information you want LaTeX/BibTeX to place in the text. Note that the comma is not a style, it is for LaTeX to separate author and year. The second argument is the BibTeX key that is unique for all references. This is what `natbib` uses to link a unique reference in the list with a `natbib` cite command in the text. After the `bibitem` command follows the reference formatted the way it should look according to any author instructions that may be given for the reference list.

Using the system this way is very simple but has the draw back that while all references in the text will be matched by a reference in the reference list, there is nothing that prevents you from adding references in the list that are not matched in the text. This means the cross-referencing will always be correct but LaTeX cannot do anything about the content in the reference list. That will be up to you. You also have to manually type in the references and check that each gets its unique cite key.

So to sum up, this way of using references in LaTeX is useful to know but the fully automated system, described above, is what you should aim for.

Do it

- Add some references using the semi-automated method described above and cite them (using `natbib` citing commands) in your document.

10 Automating LaTeX or creating your own commands

Since LaTeX is programmable you can accomplish almost anything. The problem is that it can become complicated. There is, however, a simple way to make life easier through the command `\newcommand`. This allows you to create your own command from the very simple to the very complicated. In its simplest form you can use the command to create a new command that will type something complicated for you in an easier way. Take δ^{18}O, for example. To type set this you need to write

```
$\delta^{18}$O
```

So if you were writing a manuscript on stable isotopes, you may get really tired of repeating this over and over. We can then create a new command that types this out for us

```
\newcommand{\dO}{$\delta^{18}$O}
```

In this case we tell LaTeX that the code generating δ^{18}O will be accomplished by the new command `\dO`.

When you create a new command, it is important not to use an existing command name. LaTeX commands are always in lower case and LaTeX is case sensitive. I therefore suggest you write yours with, for example, some capital letters, so called *camel case*. If you, for example, wanted to create a command called 'do it now' this would be written as `\DoItNow`, that is the first letter in each word is capitalised.

The `\newcommand` can be used with additional input. Let us look at the new command `\Nada` defined by

```
\newcommand{\Nada}[3]{\textsc{#1} \textit{#2}\\ \textbf{#3}}
```

The first argument defines the command name `\Nada`. The second tells LaTeX that there will be three arguments. The third argument shows LaTeX what to do with the three arguments. In our case the first given by `#1` will be set as small caps, the second as italics and the third on a new line as bold. With input like

```
\Nada{What On Earth}{kind of joke typesetting}{is this?}
```

we get

WHAT ON EARTH *kind of joke typesetting*
is this?

Another example is to simplify an existing LaTeX command. In section 4 we used the command `\multicolumn{}{}{}` to add table entries that could span several columns. To simplify this we can, for example, create

```
\newcommand{\MC}[3]{\multocolumn{#1}{#2}{#3}}
```

(MC for MultiColumn) and simply write `\MC{}{}{}` in the table instead.

Even though the first example is pointless in content the message is that you can automate repetitive and boring tasks. The `newcommand` can contain up to nine arguments. In fact the cover page of the compendium is produced by a command with five arguments used as follows

```
\Cover{Writing science with \LaTeX}%
         {Peter Jansson}%
         {\includegraphics[width=23cm]{LaTeX_text_cover.png}}%
         {-1.5mm}%
         {-1.5mm}
```

which means I can use the same command to generate a different compendium cover quite easily.

There is also a related command `\renewcommand` that allows you to use an existing command name and redefine it. This can be very useful but also quite 'dangerous' if you do not know what you are doing.

Do it

- Create your own new command.

Try to replicate this using the `marginpar` (section 3.12) as a starting point

11 Copying documents from Word to LaTeX

Why have a section on moving Word content to LaTeX? By now you probably see there are some significant differences between how Word works and how LaTeX works. Most evident is the fact that while Word does all formatting in the hidden, most such work is upfront in LaTeX through commands. This can cause much grief when trying to copy text originally written and formatted in Word to LaTeX. Since you are likely to encounter persons who do not work with LaTeX, you may find yourself in a situation where most of a document is prepared in Word and left for you to, for example, transfer into a journal class template. Or, you may have an old favourite document you wish to move to LaTeX. In this section I will provide some hands-on guidelines for transferring material between the two tools as smoothly as possible.

Before continuing I need to mention that there are tools that translate Word files into LaTeX. The tools are usually good at replicating the look and feel of the Word document in LaTeX, but since that is rarely what we want, we end up with a code that mimics Word documents well but with a LaTeX code that is unnecessarily complicated. I recommend you to try such tools just to get a sense of what they can do but in the following we will work on the principle of preparing the word document so that the content can be copied using 'copy–paste' between Word and your LaTeX editor.

The first thing we need to realise is that no Word-based formatting will carry over into the LaTeX editor, only the text and the returns indicating end-of-paragraph. What order we make the following changes is immaterial, each find their own way. In short, anything that has required some form of formatting in Word needs to be addressed to work in LaTeX.

The basic strategy I recommend is to 'LaTeXify' your Word document already in Word before cutting and pasting it into the LaTeX editor.

Paragraph breaks LaTeX uses empty lines to identify paragraph breaks, Word does not. You need to go through and add empty lines between all paragraphs otherwise, you will otherwise have a hard time finding them in the LaTeX editor.

Italics and bold face The italics and bold face typesetting will not be carried over. It is much simpler to go through the Word document and look for your italics and bold text and use the commands `\textit{}` and `\textbf{}` to mark those words in Word than in your LaTeX editor where the formatting will be lost.

Super and subscripts The super and subscripts will not be carried through so format these with `\textsuperscript{}` and `\textsubscript{}` or if it concerns mathematical notation the `\^` and `\_`.

Section headings The section headings formatting (levels) will not carry through to LaTeX (unless they are numbered so that you at least can identify them). This means it is easier to insert the commands `\section{}`, `\subsection{}`, and `\subsubsection{}` in the word file prior to copying.

Mathematics If you use equations and you have formatted variables with super and subscripts in the text, you should rewrite these using the mathematical mode already in Word.

Scientific units Mixing text and mathematical mode to produce super and subscripts when writing scientific units can be quite a mess. I strongly suggest you retype units using the commands in the `siunitx` package.

Dashes You need to replace en-dashes in Word with the `--` (double dash) format used by LaTeX.

Tables There is no easy way to prepare the tables for LaTeX in Word. Just remember that any fancy formatting will be lost so it is best to keep the table in as simple form as possible.

Complex tables are hard to typeset anywhere so this is where you probably need to spend some time.

A useful tool to try is excel2latex, a plugin for Excel that allows you to save LaTeX code from a section of an Excel spreadsheet. If you take your tables from Word into Excel then this tool will yield a good basis for your table, quick and easy. At the time of writing the macro should work with Excel 2012–2016 (both 32 and 64-bit). In fact if you are uncertain about the coding of tables, this macro may be a good way to learn by making a table in Excel and then saving it for LaTeX.

Figures Since you cannot copy graphics from Word into LaTeX the only thing that will be copied is the figure caption. If you have your figures in Word and cannot produce originals for LaTeX you can right-click on the figure in Word and save it as a picture. Use PNG format for line graphics and JPG for photographs if you do so.

References If you have used EndNote or some other reference manager to produce references and reference list in your Word document then all such couplings disappear when you copy the text. Your references should, however be correct unless you decide to do some serious editing in LaTeX after copying. You basically have two ways to deal with the references. One is to stick to the manual way and simply manually check for correct cross-referencing. The other is to load the `natbib` package and either enter/import references into BibTeX and use that system or go halfway and use the `\bibitem` and different `\cite{}` commands to produce the cross referencing. Either way this could cost you some time.

Extended type faces Depending on what system you use for compiling your LaTeX document, extended type faces may be come an issue. This stems from the introduction of type faces that contain a much larger number of characters than used by older computers. XeLaTeX uses these modern type faces and may compile a text containing such extended character sets correctly while pdfLaTeX generally cannot. Switching to XeLaTeX may solve issues and for pdfLaTeX there are work arounds (see the pdfLaTeX documentation). However, care should be taken when moving text containing, e.g. greek letters or mathematical signs to ensure they become type set correctly.

There may be additional details in the Word document that either do not translate well or are simply lost but the list above should normally cover 99% of a normal scientific document in Word.

12 Making presentation slides in LaTeX

LaTeX can be used for much more than creating regular documents. There are several different classes for creating presentation slides in LaTeX. The most popular is called 'Beamer'. Beamer is the German 'term' for a computer projector and has been so named by its German author.

Creating a simple Beamer presentation is not difficult. A single slide is made by the following code

```
\documentclass{beamer}
\begin{document}

\begin{frame}
  \(a=b\)
\end{frame}

\end{document}
```

What you see is that you need to make sure you use the `beamer` class. A slide is created by an new environment called `frame`. anything you place within the `frame` environment will be on your slide. Adding new slides is done by simply adding a new `frame` environment before or after the first.

There are of course many things you can add to your basic beamer presentation. The command `\frametitle{}` can be placed in a frame environment and will then produce title for that slide. As with a regular document you can add a title page by providing the commands

`\title{}` and `\author{}` in combination with `\titlepage` (note not `\maketitle`). A two page presentation with a title page can thus look as

```
\documentclass{beamer}

\begin{document}

\begin{frame}
  \title{Presentation title}
  \author{Your name}
  \titlepage
\end{frame}

\begin{frame}
  \frametitle{The title of the frame}
  \(a=b\)
\end{frame}

\end{document}
```

Beamer comes preloaded with a series of colourful layouts. These can be invoked by using the `\usetheme{}` and `\usecolortheme{}` commands. The themes and colour schemes have specific names which are described in detail at the Beamer theme gallery. You can also create your won layouts but that requires a little bit of involvement.

In the end you should consult the Beamer class user's guide to learn more about all the possibilities available in Beamer.

Do it

- Make a short Beamer presentation consisting of a title slide and a couple of text and figure slides.
- Add some colour by using a few 'beamer themes'.

13 Reinventing the wheel? ...or not

As with all programming languages, there will be bits and pieces of code that you will need over and over. You will soon find several such pieces as you develop your own 'style', good examples are all the packages that you will soon learn not to be able to live without. To prevent having to re-enter all this information in every new document you use, you can start to build up your own 'style' file.

LaTeX allows you to read in external files into the file you are authoring. This could literally be anything from a file containing some plain text to a file containing more or less a complete document. There are several commands for entering files such as `\input{}` and `\include{}`, where both commands take a file name as argument. These two commands are quite similar but differ in that the `include` command adds the content on a new pages whereas `input` simply adds the input text right where the command occurs. But a third possibility is to input a file of commands, to replace much of the preamble. This is accomplished with the now familiar `\usepackage{}` command and where your file should be named with the extension `.sty`.

What you need to do to start building your own style file is to add all the preamble information you think you will use repeatedly and unaltered into a file `fname.sty`. This file must not contain the `\documentclass` and `\begin--\end{document}` commands and environments or any text. It can only contain material you would normally put in the preamble. If we assume you have a file you are working on and you take everything in the preamble and paste that into a file called `MyStyle.sty` then your bare document will look like the following

```
\documentclass{beamer}

\usepackage{MyStyle}

\begin{document}

\end{document}
```

The `\usepackage` command will then take your file and use its content in order to format the current document.

It is important to realise a couple of things here. First, you will always need to place a copy of your style file along with whatever document you are working on. LaTeX will not keep track of it for you. Second, if you enter additional instructions in the preamble, you need to make sure they do not interfere with your style file. If you add something before the `\usepackage{MyStyle}`, you run the risk of having whatever you want to do over-ruled by the content in your style file and if you add something after the call the new material may over-rule something you have decided will be part of your style.

It is possible to add some additional safety to the style file so that it will only be used with appropriate versions of LaTeX etc. We can start the style file by adding the following commands

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{MyStyle}[2016/04/01 my LaTeX style]
```

This tells LaTeX that the latest format of LaTeX (LaTeX2e) is needed and that this is 'my' style file of such and such date. At the end of the file we can add

```
\endinput
```

which tells LaTeX that the style file content is ended. This command means you can add more material to the file after the `\endinput` command but which will be ignored. This can be useful if you add material you are unsure you want to use but do not want to lose.

So a complete style file should look like the following example

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{MyStyle}[2016/04/01 my LaTeX style]

% your preamble material:

\endinput
```

It is possible to take the style file much further and essentially build your own class. This is beyond the scope of this compendium but the document LaTeX2e for and package writers provides much more in depth information on the subject.

14 How does it really work?

For those who wonder how the LaTeX system really works, I will provide a brief overview of what goes on when a document is being typeset by the LaTeX engine. First we should recognise that beneath LaTeX is the original typesetting language TeX created by Donald Knuth in 1977. LaTeX is essentially a, albeit huge, set of macros that facilitate type setting of documents of various kinds. The name LaTeX is short for Lamport TeX after Leslie Lamport who wrote the LaTeX system.

In its most basic form a LaTeX text document is compiled by feeding a TeX compiler the document and the LaTeX format macros to generate a so-called Device Independent file (DVI). This file can then be used by different drivers to show the document on screen or to print it on a printer or create a pdf. To respond to different needs variants of the LaTeX compiler has been created. One such is pdfLaTeX which directly generates a pdf-file as output without creating an intermediate DVI file. pdfLaTeX is probably the most widely used way to generate documents in LaTeX (Figure 2).

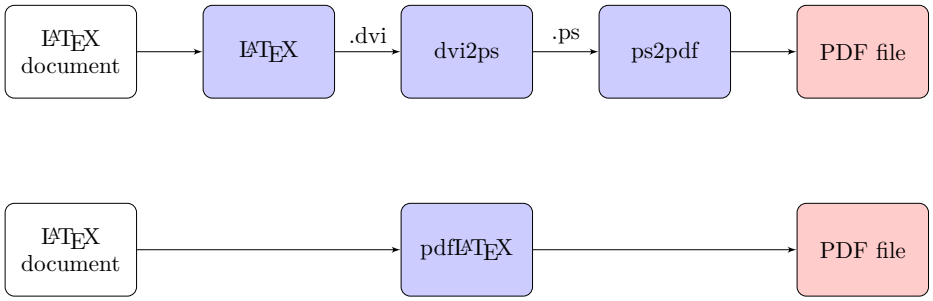


Figure 2. The basic way to generate a pdf file from a LaTeX file. (upper work flow) Generic work flow running LaTeX to generate a DVI file followed by running dvi2ps to produce a postscript file from the DVI and then ps2pdf to generate the pdf from the postscript file. (lower work flow) Work flow running pdfLaTeX to directly generate the pdf file.

Since LaTeX in its generic form cannot easily handle system fonts a variant called XeLaTeX has been created. XeLaTeX can access system fonts in a more direct way but because there are internal differences between pdfLaTeX and XeLaTeX we must be careful when trying to use certain packages in documents. Some (most) work with pdfLaTeX while some require XeLaTeX. This is the price paid by maintaining backwards compatibility. However, any problem is easily handled with some understanding of the two systems and by not overusing packages that we do not know anything about.

The way LaTeX handles information that requires recursive treatment is to store results in external files (Table 3). This applies to, for example, the table of contents, figure and table references, reference citations. The first run of LaTeX generates lists of all such cross-referencing information while LaTeX type-sets the document. This way the location of each cross-referenced object becomes known. LaTeX then needs a second run to find all references to the objects and replace these locations with the number of the figure or table or reference of the article. The result is that the document is compiled, not once, but in fact several times in order for all cross-referencing information to be properly located. Once this is accomplished you will find several files generated by this process that contains information from the process.

Table 3. Some of the more common files (extensions) generated when compiling a LaTeX document. Note that some files are generated only when specific implementations are used.

Extension	Description
`.aux`	Generic information, mostly cross-referencing
`.bib`	BibTeX reference data base file
`.bbl`	**bibl**iography produced by BibTeX
`.bst`	BibTeX **b**ibliography **s**tyle **f**ile
`.blg`	**b**ibliography (BibTeX) **l**o**g** file
`.cls`	LaTeX **cl**ass**s** file
`.dvi`	**d**e**v**ice **i**ndependent file
`.lof`	**l**ist **o**f **f**igures
`.log`	complete compilation diagnostics reported by LaTeX
`.lot`	**l**ist **o**f **t**ables
`.out`	**hyperref** PDF bookmarks
`.sty`	LaTeX package file
`.tex`	LaTeX or TeX document file
`.toc`	**t**able **o**f **c**ontents

Depending on what you include in your document LaTeX will need to be run several times and maybe also involve other supplementary programs such as BibTeX. As an example, we can view the series of runs necessary in order to produce a table of contents and a reference list using the fully automated bibliography (Figure 3).

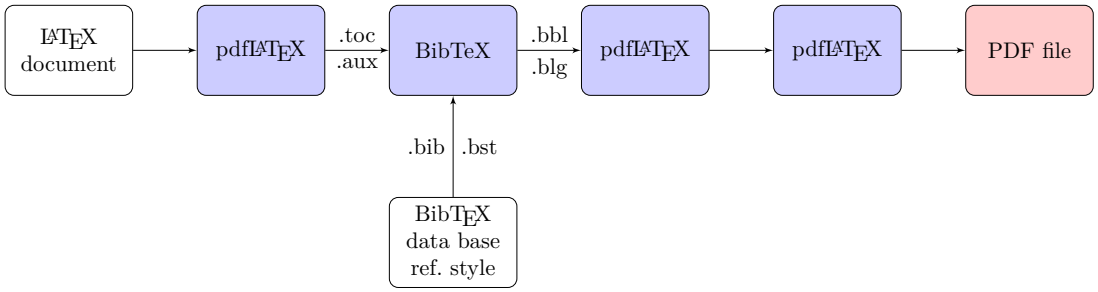


Figure 3. The pdfLaTeX work flow for a document containing a table of contents (generating the `.toc` *and a reference system based on a a BibTeX data base. Note that pdfLaTeX needs to be run twice after the BibTeX run in order for the cross referencing and reference list to be inserted correctly in the document. Many, if not all, LaTeX editors take care of this work flow automatically.*

15 Saving the worst to last

Errors. The one thing we all do not want to see. It is common to get errors in LaTeX, some are simple to fix others may seem inexplicable. The one thing they have in common is that we have done something wrong. It is therefore useful to lay down a few ground rules to, first, reduce the numbers of errors and, second, to see how we can solve them. The best way to solve errors is to make sure they do not happen.

The first rule is thus, *be meticulous when you write your document.* The by far most common error comes from having forgotten to pair `{` and `}`. The second most common error is that you misspell a command. A third, concerns mathematical mode; you have either forgotten to switch back to text mode or you use a mathematical command in text mode. In all cases, the errors should be quite easy to spot and correct.

The second rule is to *write slowly* when you are doing more advanced LaTeXing. When you have some complex code you need to enter, do not rush and put a lot in at once. Instead complete the code bit by bit and make sure each bit works before you continue with the following parts. A useful tip is to maintain an empty document, I usually call `test.tex`, where I can build and test code before pasting it into the main document. This has two advantages, one is that you are working only on one piece of code in the test document and the other is that a small document compiles much faster than if you work in your main and most likely much longer document.

Sometimes, you will come across errors that may be ridiculously difficult to spot and to solve. Such errors can come about for a multitude of reasons but, remember, you entered something into the document which did not agree with LaTeX, directly or indirectly. Therefore, go back to what it was you just entered and try to figure out why this may have caused an error. Is something incompatible? Have you misunderstood how a certain command works? There is no easy answer as to how to solve these problems, you need to figure out what you did to cause it. If it is possible, try to remove the part in the document that appears to cause the problem and put it into a separate file (the test file) and see if you can replicate the error there.

Occasionally, you end up having a recurring problem that just does not seem to go away and you cannot find a single problem. One detail to remember is the set of files generated during a LaTeX run (Table 3). It sometimes happens that one of these files contain material that causes the problem. If you have run LaTeX and then made some changes that will also change the content of these files, there may be conflicting information left from earlier runs. It is therefore useful to make a clean compile of the document by first removing all the temporary or intermediate files. You need to be careful, however, since they all will have the same main file name and just differ in their extension. What you should remove are typically the `.aux`, `.bbl`, `.blg`, `.dvi`, `.lof`, `.lot`, `.log`, `.out`, and `.toc` files; if they exist. You should at all costs *avoid* to remove any `.tex`, `.bib`, `.bst`, `.cls` and `.sty` files, of course. Many editors can do this for you safely. In Overleaf this is called a `recompile`. If the problem persists after such a recompile, then you unfortunately still have a problem in your document.

With time, and as you have created enough errors to solve, you will improve your problem solving ability. This may seem as a cumbersome way, but it is the price we pay for the flex-

ibility and power of LaTeX, and something quite familiar to those who do programming. As stated earlier, the best way to avoid errors is to not make them so write slowly and carefully, particularly when you enter commands., and compile your document frequently to that you see when an error appears.

Do it

- It is tempting to write here that you should create your own error, but most likely you have already done so in the process of trying things out. So, try to solve your errors instead.

Appendix

A Running LaTeX on your own computer

This section outlines the basic installations necessary to run LaTeX on your own computer. In order to run LaTeX you need an implementation of LaTeX, a *distribution*, and a LaTeX-compatible editor.

A.1 LaTeX

Which LaTeX-distribution you chose depends on which operating system platform you are on. For Linux, you most likely already have LaTeX already installed. For Mac and Windows, you definitely need to install it. There is one implementation call TeXLive which can be installed on all platforms; on Macs using its Mac version MacTeX. The web page provides good instructions on how to download and install the distribution. For Windows there is also a distribution called MikTeX. I would recommend installing MikTeX on Windows platforms. MikTeX has utilities to easily update LaTeX. It is also possible to create a portable LaTeX-environment by installing on a memory stick.

A complete distribution of LaTeX can be large (1–2 Gb). I would nevertheless recommend to make a full installation rather than a partial. The reason for this is that LaTeX contains a huge number of so-called packages that simplifies tasks and provides new features to the basic functionality. If you install the entire distribution, you are sure to have them within reach if you need them. It is, however, also possible to make minimal installations and have the LaTeX-implementation download packages as necessary (applies at least to MikTeX).

LaTeX in its basic form is a command-line driven software. It is thus possible to use it in command line mode. This is not a very smooth way to use it and so a front-end editor is a useful complement.

A.2 LaTeX editors

Since a LaTeX file is a simple text file any editor can be used to write a LaTeX document as long as you remember to save it as a plain text file with the extension `.tex`. Since LaTeX contains many specially formatted commands, it is very handy to have a designated LaTeX editor. Fortunately there are many such editors and most of them freeware. I can recommend two similar editors called TeXstudio and TeXmaker. These editors are implemented on all three major platforms. In fact, TeXstudio is a separate version of the original TeXmaker and they therefore have very similar characteristics. TeXstudio is completely open source while TeXmaker is free but maintained by a closed group of persons. The choice therefore largely depends on which version you like the most. TeXstudio has the advantage of being more customizable than TeXmaker. Another popular editor is TeXnicCenter. The MikTeX distribution also contains its own simple editor which is installed along MikTeX.. A good shareware editor is WinEdt. All editors should work with whatever implementation of LaTeX you use. WinEdt is not exclusively intended for LaTeX and can also be used for other programming languages. For Linux users, there are good ways to use the Emacs editor.

TeXstudio allows you to use spell checking (and thesaurus) from Openoffice. You can download spell checking dictionaries for (currently) 101 languages (not that you need them) from the OpenOffice extensions page page. Simply search on the word 'dictionary' and the language you are looking for. Please note that spell checking files for British and US English as well as French and German come with the TeXstudio installation. The files you download will have the extension `.oxt` but are in fact Zip-files. You can therefore rename the file to `.zip` and unzip the files into the 'dictionaries' folder in your TeXstudio installation and, *presto*, you have a new language installed. Please check the TeXstudio manual for details.

A.3 Reference management

LaTeX contains its own reference management system called BibTeX. BibTeX is a database system for references that consists of a standardized database format for reference material. Many other systems can read and export to BibTeX, such as Zotero, Mendeley, RefWorks and

EndNote, and may data bases including Web of Science. To manage your database you will need some form of software. There are two ways forward, either you chose an on-line service such as Zotero or Mendeley, or you go for a local service installed on your computer.

The online services Zotero and Mendeley provide means to build reference databases by accessing material directly from the web and is probably the best choice if you have not begun building such a database with any particular software. These services can provide the necessary information to LaTeX to enable you to reference data from your data base directly in your document. The data is stored on-line but you also have the option of installing a standalone version that can be synchronized with the on-line version. This is of course most valuable if you wish or must to work off-line.

If you want to keep your reference database only on your computer I strongly recommend JabRef. JabRef is a Java-based reference manager. This also means it works equally well on all platforms and it is free! JabRef installs through the downloaded installer. Since JabRef is Java-based, it will also require Java to be installed on the computer. In Windows, you will be prompted to download it if you try to start your already installed JabRef without Java. Java installs easily through an interactive installer.

A.4 Yet more software that might be of use

Converting documents from Word to LaTeX or *vice versa* is not difficult in principle but can be really tedious. There are some tools that can be helpful for such conversions. One of the most advanced is the GrindEQ suite of applications. GrindEQ is not free and whether or not you find it useful is a matter of preference. You should try it out and see what you think. My own experience is that it is useful to a point. The LaTeX created from a word file still requires a bit of editing to be usable. Most often manual editing is the best option since it gives you a chance to go through the document more carefully.

If you need to create tables, you can make use of a `excel2latex` Excel macro. This installs in Excel and allows you to generate table output for inclusion in LaTeX. Currently the macro is supported up to Office 2012–2016 so use in later versions will have to be tested.

B Install LaTeX on your computer

B.1 Windows

Installing LaTeX and the editor of your choice should be a breeze. In the case of MikTeX and TeXmaker you basically run automatic installers. MikTeX requires you to download an installer (be sure to chose 32 or 64-bit depending on your system) from the web page and to run it. You can chose to either download LaTeX or to install it directly. Depending on how fast or steady your internet connection is, you could chose either way but I recommend to use the download option. If you chose this option, make sure you download a full version and also note where the installer saves the file. You will find it in a folder called `MiKTeX 20.6 Setup` (depending on which is the current version you download). Once the download is complete you open the folder containing the downloaded files and start the installer in that folder (e.g. `basic-miktex-20.6.29-x64.exe`, again, depending on which is the current version). Running this installer will install LaTeX.

The editors install without problems by just running the installers.

B.2 Mac

The simplest way to install LaTeX on the Mac is via MacTeX . The site provides the `MacTex.pkg` file to install everything needed for running and maintaining your LaTeX installation. There is also an optional `MacTeXtras.zip` which provides some additional tools for editing etc. Once installed you will find a Preference Pane for the selection of your TeX distribution (i.e. version) but you can ignore this unless you have an older installation on your Mac. The basic `MacTex.pkg` installer provides BibDesk, LaTeXiT, TeXShop, TeXWorks and Excalibur as well TeXLive Utility. The TeXLive Utility is used for updating and installing new packages (functions or tools) and can be found under `/Applications/TeX/TeX Live Utility.app`.

The other tools installed are useful but there are alternative applications available that may be more to your liking. bbedit is a very useful, freeware text editor that can parse TEX using a set of scripts. Alternatively, Latexian (available through the App Store) is a very clean, 'Mac-like' program that provides live previews of the compiled text as well as a good navigation interface.

B.3 Linux

A basic TEX-environment is included in most Linux distributions.

C Learning more on LATEX

As with all high quality and hence popular freeware, the internet is a very good source for more information. There are several sources for information that deserves mention: The LATEX Project home (see additional links on the site), the TEX StackExchange, and perhaps the LATEX Community. The TEX StackExchange is highly recommended because you can easily search for older asked questions, pose new ones (if searching the site did not yield an answer) and take part in discussions. In fact if you do a search with the keyword 'latex' and something you are looking for, StackExchange entries will inevitably be high on the search.

In addition to the web-based sources above, the Comprehensive TEX Archive Network (CTAN) repository contains both files and manuals for thousands of packages that solves specific tasks in LATEX. The two main distributions MikTEX and TEXLive both use CTAN as their source for packages so if you have installed a complete set-up of these you also have most of what is on CTAN installed and ready to use.

Getting to know the details of LATEX may be a little difficult because of the wealth of material available on the internet. There are, however a few books that may be of use such as Mittelbach et al. (2008), Voss (2010), and van Dongen (2012). Mittelbach et al. (2008) is a comprehensive book about the core features and packages of LATEX. It works well both for learning but also to be used as a dictionary. Voss (2010) is also mostly a reference handbook of different commands. It does not provide as much explanations as Mittelbach et al. (2008) but is well organized and much more condensed. van Dongen (2012) includes discussion of a very useful subset of packages in LATEX. It is a very good start and overview that is hard to find elsewhere but does not go into depth. An additional book worth mentioning is Voss (2011) which is devoted to typesetting tables. One book that also deserves mentioning but for a different reason is Kottwitz (2011). This book is aimed to be an introduction to LATEX and will only work as such. It is not very useful as a reference after its introductory values. You will also find many introductions to LATEX on the web, the most used is probably the so-called `lshort`, 'The Not So Short Introduction to LATEX'. There is also a Wiki book on LATEX which is under development.

… and then there is this document.

References

Kottwitz S. 2011. LATEX beginner's guide. Birmingham: Packt Publishing.

Mittelbach F, Goossens M, Braams J, Carlisle D, Rowley C. 2008. The LATEX companion. New York: Addison-Wesley.

van Dongen M. 2012. LATEX and friends. Berlin: Springer.

Voss H. 2010. LATEX quick reference. Cambridge: UIT Cambridge.

Voss H. 2011. Typesetting tables with LATEX. Cambridge: UIT Cambridge.

D Revision history

v.4. 1 Oct., 2020 Revisions made to follow developments in Overleaf. Reorganisation in the section of package descriptions. Added a brief chapter on graphics in addition to presenting the `graphicx` package. General updates of links and typos.

v.3. 15 Oct., 2019 The part on referencing is completely rewritten with notes on Zotero and Overleaf 2 integration included. Many minor typos and errors in v.2 have also been corrected throughout the document.

v.2. 19 Sep., 2018 Added appendix on how to install LaTeX on a personal computer. Minor typos in v.1 corrected. Adjusted for Overleaf v2.

v.1. 29 Apr., 2016 First public version