Stockholms universitet | Institutionen för naturgeografi

Scientific writing with LaTeX

Peter Jansson

April 29, 2016

Contents

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 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.

LaTeX can be installed locally on your computer by downloading appropriate distributions. I suggest you visit the Comprehensive TeX Archive Network (CTAN) page outlining recommended distributions for the different platforms, Windows, Macintosh and Linux. You also benefit from a dedicated editor and I recommend TeXstudio as a starting point. That said, I would strongly advice the beginner to sign up for and use Overleaf which is a web-based writing tool. Overleaf is free up to 1 GB storage space, which should be more than sufficient unless you endeavour on a book project. Please visit their site and look at the introductory video to familiarize yourself with the Overleaf environment. This document was entirely written using Overleaf.

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. 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. 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.

1 Scientific writing

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. 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 in.

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 a fair number of commands and exactly 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 in using LaTeX for menial tasks such as writing a letter or a memo to colleagues or a list for making purchases, that is the realm for Word. LaTeX comes into its own when the document is more complex.

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, and it is also very likely 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 simple in Word is difficult in LaTeX and *vice versa*.

2 A quick start to writing in LaTeX

In this section I will provide the basics to produce a document in LaTeX. I will focus on working knowledge more than understanding. In the following sections, I will go more in-depth with the understanding of how LaTeX works. When you read the remaining part of this section I expect you will follow along and try the examples.

Let us start with a basic example of a document.

```
\documentclass{article}

\begin{document}
\title{Title of the document}
\author{Your name}
\maketitle

\section{This is the fist heading}
Here you can type in your text
% This is a comment

\section{This the second heading}
Here is where you type in additional text.

\end{document}
```

This simple document shows us several things. The first thing you notice is that there are commands identifiable by the backslash and curly braces, such as `\documentclass{article}`. LaTeX produces documents in response to such commands, similar to how you build a web-page in html. You will with time learn many key commands but it is a surprising few that are needed to get you far.

If we look at the code we can identify three mandatory commands:

```
\documentclass{article}

\begin{document}
```

```
\end{document}
```

These must be present in all documents because they tell LaTeX what kind of document to produce (in this case an article) and where the document starts and ends. When you start a new document these three lines should be the first you enter.

The part of the document that you will see comes from what you write between the `\begin`–`\end{document}`. In the example the document starts with three lines

```
\title{Title of the document}
\author{Your name}
\maketitle
```

The first command `\title{}` allows you to provide a title for your document and the second `\author{}` allows you to provide your (and other's) name(s) to the document. The third command `\maketitle` takes the first two and produces a title for your document. When you enter these three lines in your document you may notice that nothing will be visible until you type in the last line. Another surprise is that you also see the current date printed. `\maketitle` does this as well automatically. What we have now produced is the header of the document. We now turn to the content.

In the remaining text you see one command `\section{}` which produces your first heading. 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.

One line starts with the `%`-sign. As you see on your screen this line does not show. This i because the percent sign is used for 'commenting out' text. You can in other words write things in your document, reminders to yourself or a co-author, that is only visible in the LaTeX code but not in the type set document.

After the heading you can simply type in the text you want. 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. If you typed in some accented letters such as the Swedish å, ä and ö, you will notice they do not appear. You can be calm, we can make them appear by adding a few details described in section 7.1, for now just try to ignore this issue.

Do it

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

A mostly overused but nevertheless useful feature of a document is the list. A bullit 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{}` environment called `itemize` is a bullet list. 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 won but we will cover that in section 3.9

Do it

- Make your own list.

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{}` environment called `tabular`. After the `\begin{}` is a set of curly braces showing the letters `l c r`. This is the alignment for the columns in the table, right, centre and left in this case.

The table itself is made up of several rows of data. Each row has the form `& & \\`. The ampersand indicates a column but as you see only columns 2 and 3 are marked. the first column is always without an ampersand. The ampersand is the tab that separates columns. Each row of information is ended by a double backslash. This is the LaTeX instruction to end a line.

In the table, there is also the command `\hline` which produces a horizontal line. This basic table 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

```
I will write an equation
explaining the meaning
of $X_Z$.
\begin{equation}
X_Z = \sum x_z +  \beta^n
\end{equation}
```

I will write an equation explaining the meaning of X_Z.

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

This equation is of course nonsensical but serves the purpose of showing 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. 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 code `\begin`–`\end{equation}`. This is the way you type so called '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 dealt with in section 6 but for information, use the LaTeX cheat sheet to try out some mathematics.

Do it

- Make your equations using the cheat sheet.

You now have almost all you need to make a complete scientific manuscript, but you obviously wonder about figures (graphics) and references. This is a bit more involved and cannot be described until you have gained some deeper understanding. We will return to these aspects in sections 7.2 and 8.

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 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. The general structure requires a few more comments.

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 for storage after the command.

Third, the space between `\documentclass` and `\begin{document}` is 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 realize that no ordinary text can be written in this part of the document, only commands. Errors will otherwise occur.

3.2 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 others 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 large publishing house Elsevier, the American Geophysical Union and the Open Access publisher Copernicus has classes for their journals. Most journals on Wiley and Springer also have class files for their own style.

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.

3.3 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 are

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

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, the tab stop in tables and the mathematical mode switch as well as super and subscripts.

3.4 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 summarized 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.

Font size		Type	Command
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 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. If you want to make an article in 12pt you need to provide

```
\documentclass[12pt]{article}
```

in your document.

Changing type faces, is not necessarily straight forward; I am almost tempted to state you should not 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 218 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, you are most likely using an implementation of LaTeX called pdfLaTeX. This generates pdf files directly from your LaTeX source and is probably the most widely used. In part to make use of the newer Unicode type faces including over a million characters, another implementation called XeLaTeX has been developed. 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 pitt falls with going either way.

That said,I will nevertheless show a few simple ways to change type face in LaTeX. The LaTeXFont Catalogue contains an overview of generally available 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.

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.

3.5 Typographic features

LaTeX contains many special characters and features that can be accessed by commands. A list of some common features is

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

Most of these need no further comment but I need to point out a couple of details. First, when it comes to using quotes, it is possible to use the common double quote on the key board 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 minus (en-dash). The hyphen is part of your key board and used when you hyphenate words or put two words together. The minus is used when you want the meaning to be ‘to’ as in 1–9 (1 to 9). Since it is very easy to distinguish between the two types of dashes in LaTeX you should make an effort to use them appropriately.

3.6 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[`n`]`	page break keeping justification $n = 1 \ldots 4$
`\nolinebreak[`n`]`	–
`\\`	line break without new paragraph
`\newline`	line break without new paragraph
`\linebreak[`n`]`	line break keeping justification $n = 1 \ldots 4$
`\nolinebreak[`n`]`	–
`\-`	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 on 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.

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 wordprocessors. 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.

3.7 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. In addition the book class contains several other features. By adding the command `\appendix` LaTeX will start alphabetic sectioning numbering instead of numeric. There are also 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.

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.

3.8 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 go. There is, however, one caveat. The table of contents can only be generated from the numbered sections.

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}
```

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 5.

If you have many sections of all levels. Then 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 sections and subsections will be visible.

3.9 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 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.

3.10 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 7.4)

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 and 'eq:' for equation 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.

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 `\{}`. 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`.

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 7.4 for a detailed description. At this point it is sufficient to know that the `\cite` command is for literature references coupled to a reference list.

3.11 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 LaTeXhas 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. I will provide some examples of such programming in section 9.

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 based (see section 9) 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.

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.

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

The core of the table contains rows such as for example `1 &2 &3 \\`. We can see that the ampersand is sued to mark columns 2 and 3 but is not used for column 1. This is how LaTeX uses the ampersand in the tabular environment. 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 7.6) you have access to a series of additional lines, called rules that will help you organize 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. 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 number 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

```
{\small\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 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 new 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 7.2. 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 will always be typeset below the figure 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 3.10).

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 table caption 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.

The way float placement works has been described by Frank Mittelbach on TeX StackExchange. and I will summarize 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 7.13.

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 package `lipsum` 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.

6 Mathematical typesetting

Since mathematics is one of LaTeX original strong points there are too many introductions to count on writing math 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 math and then onto the details of how to create the equations.

The basic display math 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, you obtain 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}
```

$$\begin{aligned} y = a_1x &+ b_1x + c_1x + d_1x \\ &+ e_1x + f_1x \end{aligned} \tag{6}$$

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 LaTeXshould 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 italicized. 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}
```

$$\begin{aligned} \tau &= \rho gh sin\alpha && (7)\\ \tau &= \rho gh \sin\alpha && (8) \end{aligned}$$

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 with 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}
```

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

The left and right commands must pair up. You can use (), [] and {} and also | with these commands.

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.

7 Extending the functionality of LaTeX, packages

Because LaTeX is OpenSource, many enthusiasts contribute to LaTeX functionality by providing extensions, so called *packages*. At the time of writing this the main repository for LaTeX, the Comprehensive TeX Archive Network, CTAN, contains 5085 packages provided by 2335 contributors. Each package comes with its own instruction manual which you need to look at before using any package.

Out of all the available packages, only a small fraction will 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 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 options. Such options are described in the package documentation.

7.1 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}
```

7.2 graphicx

The `graphicx` package is a *must*. Without this package you will not be able to include graphics into your document. 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.

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 millimeters, 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 unproportionally by providing different scaling factors for width and height.

Do it

- Add a figure and test the scaling and rotation capabilities.

7.3 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.

7.4 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 8. 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.

7.5 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 width 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
    }
```

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 7.11). 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.

7.6 booktabs

The `booktabs` package is a small package that adds functionality to tables. 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`.

7.7 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 contains much more on representing data in text so you need to study the extensive `siunitx` package documentation.

Do it

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

7.8 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

7.9 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.\cdot4`)

```
\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.

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.

7.10 tikz/pgf

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

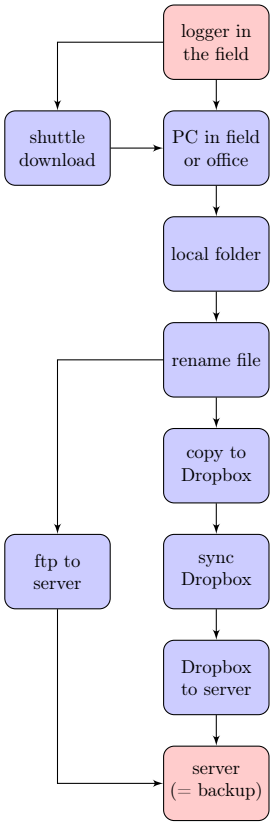
```
\usepackage{tikz}
```

The `tikz` package documentation is 1161 pages (at the time of writing, version 3.01) 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 i snoteworthy that this i sstill 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{decision} = [diamond, draw, fill=blue!20,
    text width=4.5em, text badly centered, node distance=3cm, inner sep=0pt]
\tikzstyle{block} = [rectangle, draw, fill=blue!20,
```

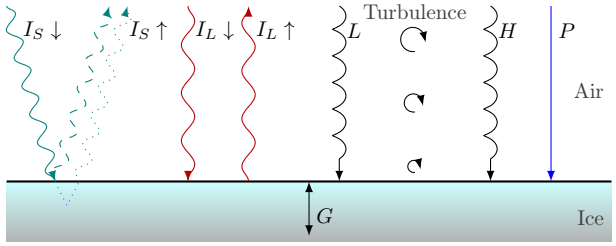
```
    text width=5em, text centered, rounded corners, minimum height=4em]
\tikzstyle{rblock} = [rectangle, draw, fill=red!20,
    text width=5em, text centered, rounded corners, minimum height=4em]
\tikzstyle{line} = [draw, -latex']

\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}        % NO introduction necessary
\usetikzlibrary{decorations.pathmorphing}
\usetikzlibrary{decorations.shapes}
\usetikzlibrary{patterns}
```

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.

7.11 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 loaded by

```
\usepackage{xcolor}
```

With this package you can access a large number of predefined colours. These are described in the `xcolor` package documenation. 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}
```

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{green!40!red}
```

In the example the mix will be 40% green and thus 60% red (since they must sum up to 100%).

Do it

- Define and use some of your own colours.

7.12 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.

7.13 float

The `float` package adds a placement option `H` to the `figure` and `table` environment (see section 5). The option is a stronger placement directive the the native options. The package is loaded by

```
\usepackage{float}
```

7.14 amslatex

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

```
\usepackage{}
```

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 Automating your bibliography

LaTeX provides a couple of ways to handle references and cross-referencing of references, one semi-automated and one fully automated. But before I get into those details I need to start with collecting and managing references.

It is very easy to start to build your own data base of relevant literature you need for your work. For Word, you have probably heard of EndNote, Reference Manager and RefWorks. There are also several web-based systems that quickly gain popularity such as Mendeley and Zotero. all these programs have different ways to interact with word processors. LaTeX was supplied with its own database system in 1985 when BibTeX was created. With this system a new standard for storing bibliographic information was created. If you go to a journal, look at an article and click on their link for exporting citation, you will see that BibTeX is one of the standard formats. LaTeXreference handling works with BibTeX format and all programs and online services provide export to BibTeX format. In addition there are programs that work directly with BibTeX such as the freeware JabRef. You should under all circumstances start to build a reference data base.

In LaTeX your cross-reference references in a similar way to how cross-referencing of figures, tables and equation works, through a label. The label is often referred to as the BibTeX key and must be a unique text string (or number) for each reference. In native LaTeX you have the `\Cite` command. If you have a reference in your data base with a unique BibTeX key, say, `Smith:2013`, all you need to do in the text is to enter `\citeSmith:2013` to put the correct reference in the text and have it correspond to the correct reference in the text. Unfortunately the native command is not enough to create the Harvard style or author-year type referencing we are used to. So I will now focus on the tools you need instead of the native LaTeX referencing.

A small comment: I usually keep my BibTeX keys as 'first author name:publication year' but there is nothing sacred about that format. You consider your own system that works for you.

8.1 The `natbib` package

When you want author-year style referencing you need to use the `natbib` package (section 7.4). This package adds many new commands dealing with different types of references.

Numbered	Unnumbered
`\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, you simply add them all as argument separated by a comma. `\citep{Smith:2013,Day:2014,Knight:2015}` will yield (Smith, 2015; day, 2014; Knight, 2015).

The 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: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, 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 semi and the fully automated way.

8.2 The semi-automated reference system

In the semi automated referencing system, LATEX keeps track of the cross-referencing but you need to provide all references in writing. this means that you need to crate a reference list at the end of your document following a specific format.

```
\begin{thebibliography}{}

\bibitem[Smith, 2013]{Smith:2013}
Smith, A.B., 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 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.

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 LATEXcannot do anything about the content in the reference list. That will be up to you.

Do it

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

8.3 The fully automated reference system

The fully automated system both handles 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 saved as a BibTEX file (it has the extension `.bib`). You can save such files from your data base system or if you use JabRef it is already available as such. 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. Journals that take LATEX manuscripts often have their own styles that can be downloaded. Otherwise, I would recommend going with the Modern Language Association (MLA) style. It is a neutral standard that is simple.

When you have your data base file and your style file, you can start using it in your document. You will reference the works the same way as discussed above but the reference list is now created

for you and instead of the bibliography environment yo need to add the following code where you want your reference list to appear.

```
\bibliography{foo}
\bibliographystyle{mla}
```

The first command `\bibliography{}` tells BibTEX which `.bib` file to look for. This is your bibliography file. The second tells BibTEX which bibliographic style that should be uses, the name of your `.bst` file here exemplified by the `mla.bst`.

With all this in place your reference list should be up to date whenever you ad a reference to a new publication in your document and you should not need to worry about any discrepancies in the document. 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 styles.

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 Automating LaTeX or creating your own commands

Sine LaTeX is programmable you can accomplish almost anything. The problem is that it can become complicated. There is, however, simple ways to make life easier and that is 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 this you need to provide

```
$\delta^{18}$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 all you need to type in your manuscript i `\dO` and you will receive δ^{18}O.

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 some capital letters, so called *Pascal case* (after the programming language Pascal). 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 capitalized.

The `newcommand` can be used with additional input. The command created by

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

will yield . In this case we specify that the command should have three arguments and in our definition we use the # to identify where each of these should occur in our command. With input like

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

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

Even though the example is pointless in content the point that you can automate repetitive and boring tasks is there. The `newcommand` can contain up to 9 arguments.

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 3.11) as a starting point

10 Copying documents from Word to LaTeX

Why have a section on moving Word content to LaTeX? As you probably have understood 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. This can cause much grief when trying to copy text originally written 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 LaTeXcode 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 LaTeXeditor.

The first thing we need to realize 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 praxis. In a way, one can say that anything that has required some form of formatting in Word needs to be changed.

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 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

Section headings The section headings formatting (levels) will not carry through to LaTeX (unless they are numbered). 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. This is very common in references; page numbers.

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 worked with Office 2013. 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 and save it as a picture. Use PNG format 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 reference 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.

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.

11 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{Presetation title}
  \author{Yourname}
  \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'.

12 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{docuemnt}` 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.tex` then you 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 realize 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 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 sued 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 LaTeXthat the lates 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 sueful 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 beyind the scope of this text but the document LaTeX2e for and package writers provides much more in depth information on the subject.

13 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 recognize that beneath LaTeX is the original typesetting language TeX created by Donaldh Knuth in 1977. LaTeX is essentially a, albeit huge, set of macros that facilitate type setting of documents of various kinds.

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).

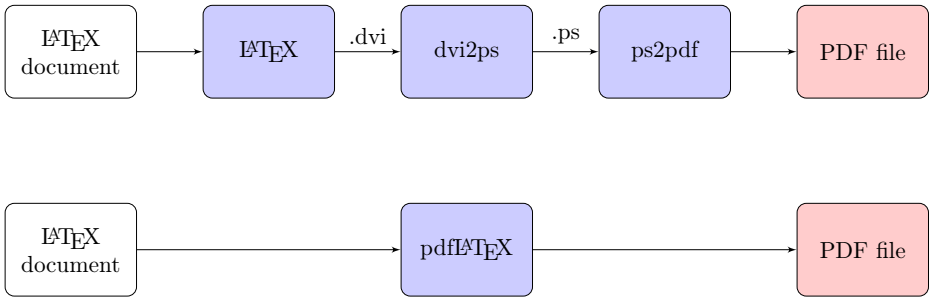


Figure 2. The basic way to generate a pdf file from a LaTeX file. (upper work flow) Generic work flow running LaTeX to generate a DVI file followed by running dvi2ps to produce a postscript file from the DVI and then ps2pdf to generate the pdf from the postscript file. (lower work flow) Work flow running 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 1). 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 LaTeXtype-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 1. 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 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).

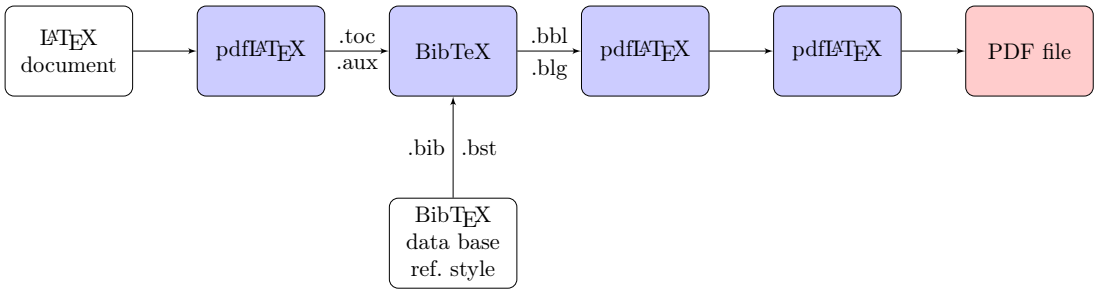


Figure 3. The pdfLaTeX work flow for a document containing a table of contents (generating the `.toc` and a reference system based on a a BibTeX data base. Note that pdfLaTeX needs to be run twice after the BibTeX run in order for the cross referencing and reference list to be inserted correctly in the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

14 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 a `{` or `}`. 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 yo 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 a part of the document to a separate file (the test file) and see if you can replicate the error there.

Sometimes, 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 (Table1). 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 from scratch`. 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 flexibility 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.