LaTeX Reference

Floris Westerman

January 9, 2023

This document is meant as a reference for basic LaTeX functionality. It should be sufficient to write basic reports and documents for the first years of your study. I the first chapter, the basic workings of LaTeX will be explained and I will guide you on starting to use it. After that, the basics of text formatting and mathematics will be discussed. Later on, more advanced subjects such as tables, matrices, figures and graphs are listed. In the appendix, some more complicated and involved examples are given that can be used as templates for your own document.

In the documents, a lot of packages are mentioned. Each mention of a package is clickable and will lead you to the list of packages that I mentioned or recommend. There, you can also find other places where the packages are mentioned. If you want to know more about some packages, simply Google 'CTAN [name]' and you'll easily find the documentation of the package for further reference.

In this document, I often mention the syntax of commands. Such a syntax may look like `\command[<options>]{<name>}`. Here, parts between `<>` are variable parts that you have to fill in yourself; a description is given of what should be put in. These `<>` should be omitted in your final code, the rest of the code should be exactly as it was given.

I plan to extend this document over time with more sections, such as making code listings and doing some chemistry. Therefore, when sharing the document, please share the link and not the pdf itself you will always get the latest version. The latest version can always be downloaded from `https://floriswesterman.nl/LaTeXReference.pdf`.

For any comments and remarks or if you've found a mistake, please contact me at me@floriswesterman.nl.

Contents

1 Setting up

Before you can start using LaTeX there are some things to set up, install and configure. Also you'll need some basic content in your document before you can start actually adding your text.

1.1 Installation

1.1.1 Windows, Linux, OS X

There are multiple ways to compile `.tex` files into `pdf`'s. On all common platforms (Windows, Linux, OS X) it is easiest to install MiKTeX as your distribution manager - the program that manages all packages and other prerequisites to use LaTeX.

MiKTeX also already includes a very basic editor. For more advanced functionality, such as code completion, highlighting and others, there are multiple editors available. I'd advise using TeXstudio, a versatile editor that features a lot of functionality and other options.

For OS X there is also TeXpad, that integrates both the distribution manager and the editor in one very nice program. One disadvantage is that this software is paid and will cost you around €23.

1.1.2 Online

It is also possible to work with LaTeX online. This saves you the trouble of installing programs on your computer and enables you to collaborate directly with others; it works like Google Docs so you can work on documents simultaneously. The main disadvantage is that these websites are generally quite slow to work with, building is often very slow with larger documents taking up to a minute to compile and of course it's required to have an internet connection. Commonly used websites are Overleaf and ShareLaTeX.

1.2 Structure of a LaTeX document

1.2.1 Document class

A LaTeX document has a general style and set-up. Firstly, each document has a `documentclass` set, a set of basic styles and commands available. Commonly used styles are `book`, `article` and `letter`. These all have their own default font sizes, whitespace sizes, etc. You can also download one of the many styles available on the internet, some specifically designed to make resumés, articles for specific scientific journals and many more.

A document class is set using the command `\documentclass[<options>]{<name>}` where your options are specific options for your document class. The options available differ for each class, but you can generally set the paper and font size you're working with. In between the curly brackets you put the name of the document class. This is a common pattern in LaTeX, options are generally placed in between square brackets in front or after the curly brackets with the actual content.

1.2.2 Packages

After that, you'll generally define the packages to be loaded. Packages are the add-ons for LaTeX, you can add a lot of functionality with them. They enable you to go past the standard feature set of LaTeX and make beautiful graphs, tables and lists. You can make your text more colorful or change the size of anything you want. When working with MiKTeX, these packages will be downloaded the first time you use them, after that they're permanently loaded.

Packages can be included using the `\usepackage[<options>]{<name>}` command. Again, within the square brackets you can put your options and in between the curly brackets you put the name of the package. In this document I'll sometimes state that some command requires some package or some option, this is how you do that.

1.2.3 Environments

All of the above together is called your *preamble*. After this will be the main content of your document. This content is divided into so-called *environments*, 'blocks' of content with a specific purpose. Environments are opened, or started, by `\begin{<environment>}` and ended by `\end{<environment>}` where `environment` is the name of your environment. The content in between both statements will be inside that environment. The main environment is `document`, it conatins all content of your document. There are separate environments for maths, your abstract, figures, tables, graphs, lists, etc.

1.2.4 Example document

Below is an example of a simple document in LaTeX. It'll make a document with document class `article` and consist of a title (the commands `\title{}`, `\author{}` and `\maketitle` are defined in the document class) and a short text.

Simple LaTeX document

```
\documentclass[a4paper,10pt]{article}

\usepackage{a4wide}

\title{My fancy report}
\author{Floris Westerman}

\begin{document}
    \maketitle
    This is a nice document
\end{document}
```

2 Text formatting

2.1 Titling

Titling can be done using the `\section`, `\subsection` and `\subsubsection` commands. Each 'sub' indicates a level in your headers. All of them will be included in the table of contents (`\tableofcontents`). There are multiple versions:

Command	Description
`\section{Title}`	Just a normal section.
`\section[Short title]{Very long title}`	The short title in between `[]` will be put in the table of contents, the long title will be shown in the document itself.
`\section*{Title}`	An unnumbered section title. Useful for when there is no ToC or when it is not relevant to have numbering (in a smaller document).
`\subsection`, `\subsubsection`, `\paragraph`, `\subparagraph`	Same as above, same options, but lower level and smaller size

2.2 Text sizes

It is possible to write text in smaller and larger font sizes. The following font sizes are available:

Command	Description
`{\tiny Text}`	This is tiny text.
`{\scriptsize Text}`	This is (super-/sub)script sized text.
`{\footnotesize Text}`	This is footnote sized text.
`{\small Text}`	This is small text.
`{\large Text}`	This is large text.
`{\Large Text}`	This is larger text.
`{\LARGE Text}`	This is very large text.
`{\huge Text}`	This is huge text.
`{\Huge Text}`	This is truly huge text.

2.3 Text styles

There are multiple text styles, such as underlines, italics, etc. You can also change your font size. These commands are available:

Command	Description
`\textbf{Text}` `{\bf Text}`	This text will be made **boldface**.
`\textit{Text}` `{\it Text}`	This text will be made *italics*.
`\textsl{Text}` `{\sl Text}`	This text will be made *slanted*.
`\textsc{Text}` `{\sc Text}`	This text will be made SMALL CAPS.
`\textsf{Text}` `{\sf Text}`	This text will be made Sans Serif.
`\texttt{Text}` `{\tt Text}`	This text will be made `monospaced`.
`\textsuperscript{Text}`	This text will be made $^{\text{superscript}}$.
`\textsubscript{Text}`	This text will be made $_{\text{subscript}}$.
`{\color{blue} Text}`*	This text will be made blue.
`\underline{Text}`	This text will be made underlined.
`\st{Text}`**	This text will be made ~~strike-through~~.

* **This requires the `color` package**

** **This requires the `soul` package**

2.4 Paragraphs and alignment

New paragraphs can simply be introduced using a double enter. If you want to force an 'enter' without introducing a new paragraph, one can use the double backslash `\\`. Furthermore, text can be justified left and right and can be centered.

Command	Description

`\begin{flushleft}`
`    Content`
`\end{flushleft}`
	This text will be aligned on the left-hand side of the page.

`\begin{flushright}`
`    Content`
`\end{flushright}`
	This text will be aligned on the right-hand side of the page.

`\begin{center}`
`    Content`
`\end{center}`
	This text will be centered on the page.

`\\` `\newline`	This forces a new line to be created.
Double enter `\par`	This will create a new paragraph.
`\newpage`	This will force a pagebreak, so text will continue on the next page.

By default, new paragraphs are indented on the first line. This is the standard in many reports and articles. However, you might not like it. The easiest solution is to simply include the package `parskip`, so put `\usepackage{parskip}` in your preamble.

2.5 Lists

It is possible to make lists with numbers and without numbers. The 'counted' lists use the environment `enumerate` and the others use `itemize`. It is also possible to nest multiple lists, so a basic example would be:

```
\begin{enumerate}
    \item First item
    \item Some sublist:
        \begin{itemize}
            \item Subitem
            \item Subitem
        \end{itemize}
    \item Last item
\end{itemize}
```

1. First item
2. Some sublist:
    - Subitem
    - Subitem
3. Last item

Using the package `enumitem` it is also possible to do some more advanced things. The most useful thing is to define custom labels for your items. You can do roman or letter-notation instead of normal digits and use other shapes in the itemize or even make a sort-of key-value table. Short overview:

Command	Example
`\item [Custom] And the text`	1. First Custom And the text! 2. Third
`\begin{enumerate}[label=\alph*)]`*	a) First b) Second
`\begin{enumerate}[label=\roman*)]`*	i) First ii) Second
`\begin{enumerate}[label=\arabic*)]`*	1) First 2) Second
`\begin{enumerate*}[label=\bf(\alph*)]`**	Inline list: **(a)** first **(b)** second.

* **This requires the `enumitem` package**

** **This requires the `enumitem` package with the `inline` option**

The labels `\alph*`, `\roman*` and `\arabic*` are defined in the `enumitem` package. For `itemize` it is possible to use any symbol from any library as label of a list item.

2.6 References

It is very easy to do references in LaTeX. With just two lines of code you can refer to any figure, table, section or equation in your document and the numbering will automatically be correct. You just place a `\label{<name>}` command close to whatever you want to refer to and you put `\ref{<name>}` at the place you want to put a reference. Here, `name` is a shorthand name of what you're referring to. In larger documents it can get difficult to distinguish equations from figures from tables, it is common to prefix your names with things like `ch:` for chapters, `fig:` for figures and `eq:` for equations. As an example:

```
\begin{equation}
    \label{eq:einstein}
    E = mc^2
\end{equation}

(..)

As we saw in equation
↪  \ref{eq:einstein}, we have that
↪  ...
```

$$E = mc^2 \tag{1}$$

As we saw in equation 1, we have that ...

It is also possible to use the command `\pageref{<name>}` that will print the page number the reference is on. Especially for equations, the `amsmath` package has define `\eqref{<name>}` that prints the number with parentheses around it to easily distinguish it from other references without putting 'formula' or 'equation' in your text all the time.

In longer documents it is also useful to include the section number in your numbering and reset your numbering with each new section. This will prevent your numbers from getting too high

and it makes it more convenient to look up the references in the document. Again, `amsmath` has a nice command for this: use `\numberwithin{<countera>}{<counterb>}` where any reference to `countera` will be replaced by `counterb.countera`. To append all equation numbers with their respective section number, use `\numberwithin{equation}{section}`. This will yield numbers like '(1.5)' where 1 is the section number and 5 is the equation number within that section.

It might also be useful to include the package `hyperref`, which will automatically make all your references clickable such that they immediately take the reader to the referenced item. It also includes the command `\autoref{<name>}` which will automatically put the relevant word ('section', 'figure', 'table', etc) in front and make the entire thing clickable instead of just the number.

3 Mathematics

To do proper mathematics in LaTeX you'll need to work inside a mathematics environment. This will make your maths look like $x + y = 10$ instead of x + y = 10. Most mathematics commands only work inside such an environment, and non-maths commands will only work outside such an environment.

3.1 Environments

There are multiple ways to initiate a mathematics environment:

Command	Description
`$x^2 = 4$`	Single dollar sign is for inline mathematics like $x^2 = 4$.
`$$E = mc^2$$`	Double dollar signs denote a single line of centered maths like this: $$E = mc^2$$ You can continue your text afterwards.

`\begin{equation}`
`  a^2 + b^2 = c^2`
`\end{equation}`
	The equation block is for (multiple) centered equations: $$a^2 + b^2 = c^2 \quad (2)$$ With an unnumbered version `equation*`.

`\begin{align}`
`  y &= x \cdot \sqrt{x^2} \\`
`  &= x^2`
`\end{align}`
	This requires the `amsmath` package The align block is for multiple lines of maths aligned. The `&` denotes the place where the equations will be aligned. $$y = x \cdot \sqrt{x^2} \quad (3)$$ $$= x^2 \quad (4)$$ With an unnumbered version `align*`.

`\begin{align*}`
`  x &= 5-5 & y &= 10-4 \\`
`  &= 0  &  &= 6`
`\end{align*}`
	This requires the `amsmath` package The align block also allows for multiple columns: $$x = 5 - 5 \qquad y = 10 - 4$$ $$= 0 \qquad = 6$$ The first `&` denotes the first point of alignment, the second denotes the space between both columns and the third denotes the second point of alignment.

3.2 Symbols

Most symbols in LaTeX are quite predictable, but are only available as command. Do not paste your symbols in from Word directly, that doesn't work. This is a basic list of symbols:

Symb.	Command	Symb.	Command	Symb.	Command
Operations and relations					
$+$	`+`	$-$	`-`	$\pm$	`\pm`
$\cdot$	`\cdot`	$\times$	`\times`	$=$	`=`
$\leq$	`\leq`	$\geq$	`\geq`	$\neq$	`\neq`
$\ll$	`\ll`	$\gg$	`\gg`	$\simeq$	`\simeq`
$\approx$	`\aprrox`	$\equiv$	`\equiv`	$\propto$	`\propto`
Set theory & Logic symbols					
$\exists$	`\exists`	$\forall$	`\forall`	$\neg$	`\neg`
$\subset$	`\subset`	$\subseteq$	`\subseteq`	$\land$	`\land`
$\in$	`\in`	$\notin$	`\notin`	$\lor$	`\lor`
$\cap$	`\cap`	$\cup$	`\cup`	$\bot$	`\bot`
$\emptyset$	`\emptyset`				
Arrows					
$\to$	`\to`	$\leftarrow$	`\gets`	$\mapsto$	`\mapsto`
$\rightarrow$	`\rightarrow`	$\leftarrow$	`\leftarrow`	$\leftrightarrow$	`\leftrightarrow`
$\Rightarrow$	`\Rightarrow`	$\Leftarrow$	`\Leftarrow`	$\Leftrightarrow$	`\Leftrightarrow`
$\uparrow$	`\uparrow`	$\downarrow$	`\downarrow`	$\longleftarrow$	`\longleftarrow`
$\Uparrow$	`\Uparrow`	$\Downarrow$	`\Downarrow`	$\longrightarrow$	`\longrightarrow`
Greek letters					
α	`\alpha`	β	`\beta`	γ	`\gamma`
Γ	`\Gamma`	δ	`\delta`	Δ	`\Delta`
ϵ	`\epsilon`	ε	`\varepsilon`	ζ	`\zeta`
θ	`\theta`	κ	`\kappa`	λ	`\lambda`
μ	`\mu`	ξ	`\xi`	π	`\pi`
ρ	`\rho`	ϱ	`\varrho`	Σ	`\Sigma`
σ	`\sigma`	τ	`\tau`	Φ	`\Phi`
ϕ	`\phi`	φ	`\varphi`	χ	`\chi`
Ψ	`\Psi`	ψ	`\psi`	Ω	`\Omega`
ω	`\omega`				
Others					
∞	`\infty`	$\hbar$	`\hbar`	∇	`\nabla`

3.3 Functions and other operations

Besides using some basic symbols it is often necessary to write integrals, powers, subscripts, etc. This is a list of various mathematical constructs:

Command	Result	Command	Result
`\frac{a}{b}`	$\frac{a}{b}$	`a/b`	a/b
`x^2`	x^2	`x^{2n}`	x^{2n}
`x_n`	x_n	`x_{n+1}`	x_{n+1}
`\sqrt{x}`	$\sqrt{x}$	`\sqrt[4]{x}`	$\sqrt[4]{x}$
`\int_0^x`	$\int_0^x$	`\iint`	$\iint$*
`\iiint`	$\iiint$*	`\oint`	$\oint$
`\sum_{i=0}^n x_i`	$\sum_{i=0}^{n} x_i$	`\prod_{i=0}^n f(i)`	$\prod_{i=0}^{n} f(i)$
`\sin \theta`	$\sin\theta$	`\sin^2 \theta`	$\sin^2\theta$
`\cos \theta`	$\cos\theta$	`\tan \theta`	$\tan\theta$
`\ln x`	$\ln x$	`\log_{10} x`	$\log_{10} x$
`\lim_{x\to0} x = 0`	$\lim_{x\to 0} x = 0$		

* **This requires the `amsmath` package**

3.4 Brackets

By default, brackets in LaTeX have one static size and do not correctly scale when you have larger formulas, such as in this example:

$$(\frac{x^2+10}{x+5})^2$$

To fix this problem and make the brackets scale correctly, put `\left` and `\right` before both brackets, so you get `\left(\frac{1}{x}\right)^2`. Alternatively, use the package `physics` and put `\qty(\frac{1}{x})`, this saves you from forgetting the `\right`. Both will render as

$$\left(\frac{1}{x}\right)^2$$

There are multiple kinds of brackets available, there are normal brackets: ( ) `( )`, square brackets [ ] `[ ]`, curly brackets { } `\{\}` and triangular brackets ⟨ ⟩ `\langle\range`. All of these can be combined with the `\left\right` from above to create dynamically sized brackets.

3.5 Custom alphabets

Some mathematical statements require other symbols than the ones available by default. Examples are $\mathcal{F}$ for the Fourier transform and $\mathbb{R}$ for the set of all real numbers. These characters are available in other alphabets, the following are available:

Alphabet	Command	Usage
ABCDEFGHIJLKLMN OPQRSTUVWXYZ	`\text{...}`	Used to add normal text inside your equations.
$\mathcal{ABCDEFGHIJKLMN OPQRSTUVWXYZ}$	`\mathcal{...}`	Used for CS $\mathcal{O}$ for complexity, the Laplace transform $\mathcal{L}$ and the Fourier transform $\mathcal{F}$.
$\mathbb{ABCDEFGHIJKLMN OPQRSTUVWXYZ}$	`\mathbb{...}`*	Used for special sets, for example $\mathbb{R}$, $\mathbb{N}$ and $\mathbb{C}$.

* **This requires the `amssymb` package**

3.6 Accents

Especially in physics, at some point or another you'll need to write vectors or other accents on your symbols. Luckily LaTeX, again, has a lot of flexibility and offers many accents. Again a table:

Symb.	Command	Symb.	Command	Symb.	Command
$\vec{a}$	`\vec{a}`	$\hat{a}$	`\hat{a}`	$\bar{a}$	`\bar{a}`
$\tilde{a}$	`\tilde{a}`	$\dot{a}$	`\dot{a}`	$\ddot{a}$	`\ddot{a}`
$\dddot{a}$	`\dddot{a}`	$\ddddot{a}$	`\ddddot{a}`		

There is one problem with the vector sign however, it doesn't scale when you have longer things. For example, in geometrics you might need the vector $\overrightarrow{OP}$, but with your native LaTeX vector you'll get $\vec{OP}$. To solve this problem, add the following code to your preamble (the part before `\begin{document}`). This requires the `tikz` package. Note that when using the `external` library for `tikz` (see section 6.7.2 for more information), you should uncomment line 21 (which starts with a %).

Better vectors

```
1  % Better vectors in latex
2  \makeatletter
3  \newlength\xvec@height%
4  \newlength\xvec@depth%
5  \newlength\xvec@width%
6  \newcommand{\xvec}[2][]{%
7      \ifmmode%
8      \settoheight{\xvec@height}{$#2$}%
9      \settodepth{\xvec@depth}{$#2$}%
10     \settowidth{\xvec@width}{$#2$}%
11     \else%
12     \settoheight{\xvec@height}{#2}%
13     \settodepth{\xvec@depth}{#2}%
14     \settowidth{\xvec@width}{#2}%
15     \fi%
16     \def\xvec@arg{#1}%
17     \def\xvec@dd{:}%
```

```
    \def\xvec@d{.}%
    \raisebox{.2ex}{\raisebox{\xvec@height}{\rlap{%
        \kern.05em%  (Because left edge of drawing is at .05em)
%        \tikzset{external/export=false}% (Uncomment when using tikzexternal package)
        \begin{tikzpicture}[scale=1]
        \pgfsetroundcap
        \draw (.05em,0)--(\xvec@width-.05em,0);
        \draw (\xvec@width-.05em,0)--(\xvec@width-.15em, .075em);
        \draw (\xvec@width-.05em,0)--(\xvec@width-.15em,-.075em);
        \ifx\xvec@arg\xvec@d%
        \fill(\xvec@width*.45,.5ex) circle (.5pt);%
        \else\ifx\xvec@arg\xvec@dd%
        \fill(\xvec@width*.30,.5ex) circle (.5pt);%
        \fill(\xvec@width*.65,.5ex) circle (.5pt);%
        \fi\fi%
        \end{tikzpicture}%
    }}}%
    #2%
}
\makeatother

% --- Override \vec with an invocation of \xvec.
\let\stdvec\vec
\renewcommand{\vec}[1]{\xvec[]{#1}}
% --- Define \dvec and \ddvec for dotted and double-dotted vectors.
\newcommand{\dvec}[1]{\xvec[.]{#1}}
\newcommand{\ddvec}[1]{\xvec[:]{#1}}
```

3.7 The physics package

The `physics` package adds a lot of shorthand notation for various often-used mathematical expressions, primarily focused on physics. As mentioned before, it introduces the `\qty(..)` notation, which replaces both `\left` and `\right` in one command. For more information, see section 3.4. Besides that one, the following commands are available:

Command	Result	Command	Result
`\abs{x}`	$\lvert x \rvert$	`\comm{A}{B}`	$[A, B]$
`A \cross B`	$A \times B$	`\grad{A}`	$\boldsymbol{\nabla} A$
`\div{A}`	$\boldsymbol{\nabla} \cdot A$	`\curl{A}`	$\boldsymbol{\nabla} \times A$
`\sin[2](x)`	$\sin^2(x)$	`\Re{z} + \Im{z}`	$\mathrm{Re}\{z\} + \mathrm{Im}\{z\}$
`\dd{x}`	$\mathrm{d}x$	`\dd[3]{x}`	d^3x
`\dv{x}`	$\dfrac{\mathrm{d}}{\mathrm{d}x}$	`\dv{f}{x}`	$\dfrac{\mathrm{d}f}{\mathrm{d}x}$
`\dv[2]{f}{x}`	$\dfrac{\mathrm{d}^2 f}{\mathrm{d}x^2}$	`\pdv{x}`	$\dfrac{\partial}{\partial x}$
`\pdv{f}{x}`	$\dfrac{\partial f}{\partial x}$	`\pdv[2]{f}{x}`	$\dfrac{\partial^2 f}{\partial x^2}$
`\pdv{f}{x}{y}`	$\dfrac{\partial^2 f}{\partial x \partial y}$	`\pdv*{f}{x}`	$\partial f / \partial x$

`\bra{a}`	$\bra{a}$	`\ket{b}`	$\ket{b}$
`\braket{a}{b}`	$\braket{a}{b}$	`\expval{a}`	$\expval{a}$
`\expval{\frac{1}{x}}{a}`	$\expval{\frac{1}{x}}{a}$	`\expval**{\frac{1}{x}}{a}`	$\expval**{\frac{1}{x}}{a}$

3.8 The siunitx package

Another very useful package for physicists is `siunitx` for formatting numbers and their units. I would advise using the package with the following settings, put this command after `\usepackage{siunitx}`.

```
\sisetup{separate-uncertainty=true, exponent-product=\cdot}
```

The package provides the following commands:

Command	Result
`\num{1.234}`	$\num{1.234}$
`\num{4.23e-4}`	$\num{4.23e-4}$
`\num{4.3 \pm .2 e-2}`	$\num{4.3 \pm .2 e-2}$
`\num{4.3 e-2 \pm 2 e-3}`	Will give an error!
`\si{\kilo\meter\per\second}`	$\si{\kilo\meter\per\second}$
`\si{\kilogram\cubed\per\square\hertz}`	$\si{\kilogram\cubed\per\square\hertz}$
`\SI{4.543\pm.003e-8}{\kilogram\per\meter\tothe{5}}`	$\SI{4.543\pm.003e-8}{\kilogram\per\meter\tothe{5}}$

Also, a new column type for tables (see section 4.1) is provided, `S`. This will call `\num{...}` on each cell in that column, and will automatically align all data on the decimal separator (.). Note that in order to use normal text in that column, you should put it in between `{...}`.

In the `\si{}` and `\SI{}{}`-commands, you have to enter your units. For a full list of units, please refer to the package documentation, but using the autocomplete features in most editors you'll be able to find them as well. In short: all SI units and many derived units are available, from common ones such as `\meter`, `\second` and `\kilogram` (`\gram` does not exist!) to uncommon ones such as `\angstrom` and `\knot`.

4 Tables & Matrices

Tables are a complex matter in LaTeX. I will not treat all details here, but just the basics to get started. A table is, once again, a separate environment, which can be started using `\begin{tabular}{<conf>}` and `\begin{array}\end{array}{<conf>}`, among others. In here, the `conf` is the configuration of the columns, as explained below. The difference between both environments given is that `array` only works inside a maths environment and is therefore well-suited for e.g. matrices.

Note on `table` environment

The `tabular` environment is different from the `table` environment! The latter is a floating environment similar to `figure`, and will place your table anywhere on the page, which might be useful if you want to use it just like a figure with a number. This is not mandatory to use!

4.1 Columns

Columns of tables are defined in the `conf` part of the environment opening. In there you define the layout of your table. Each character you put in is part of that specification. For example, an `l` is a left-aligned column, a `c` is centered and `r` is right-aligned. Other options include `p<width>` which is a column with a fixed width, and the `array` package (not to be confused with the environment!) also defines `m<width>` and `b<width>` that are the same as `p` but then vertically centered and aligned at the bottom. Furthermore, `|` defines a single vertical line in between columns, `||` is a double line.

Note on professional tables

It is important to remember that professional tables in scientific articles and books almost never feature vertical lines. It is considered bad practice and cluttering to use them, because a good table layout should not require vertical lines. See section 4.2 for instructions on how to add proper horizontal lines to your table.

As an example, we get:

```
\begin{tabular}{r|c||lm{.3\linewidth}|}
    right & center & left & short \\
    r & c & l & short
\end{tabular}
```

right	center	left	short
r	c	l	short

Note that content in the paragraph-columns `p`, `m` and `b` will automatically be wrapped to be contained in the cell itself, other column types will simply put everything on one line:

```
\begin{tabular}{m{.4\linewidth}|l}
    Very long text that does not fit
    ↪ on one line so it will have
    ↪ to wrap & Very long text that
    ↪ does not fit on one line so
    ↪ it will have to wrap
\end{tabular}
```

Very long text that does not fit on one line so it will have to wrap	Very long text that does not fit on one line

The very useful package `siunitx` defines another useful column type: `S`. This will treat all content as the argument to a `\num{..}` call (see section 3.8 for more information) and will align the numbers at the decimal point. Note that in this column, all content that is not a number (e.g. column title) should be put in between `{}`.

```
\begin{tabular}{S}
    {\bf Values} \\
    2.3456 \\
    -54.23 \\
    1.2e3
\end{tabular}
```

Values
2.3456
−54.23
$1.2 \cdot 10^{3}$

4.2 Rows and lines

Inside a table, you put your content. As could already be seen in the example above, your separate separate columns with the `&` symbol and go to a new row with the `\\`. If you want multiple lines inside a cell, you therefore cannot use `\\`, but you'll have to use `\newline`. Note that this only works within columns of type `m`, `p` or `b`.

Horizontal lines can be added using `\hline` and `\cline{<i>-<j>}` where the first is a table-spanning line and the latter starts at column `i` and ends at columns `j`. However, using these you'll see that the table can become quite cramped, there will be little space left between the lines and the contents of the table:

```
\begin{tabular}{p{.6\linewidth}}
    \hline
    Example text \\\hline
    Text with some \newline newlines
    ↪  and such and also some line
    ↪  breaking \\\hline
\end{tabular}
```

Example text
Text with some newlines and such and also some line breaking

Of course, this is not desirable. To solve this, use the package `booktabs`. This package provides four useful commands inside your tables: `\toprule`, `\midrule`, `\bottomrule` and `\cmidrule{<i>-<j>}` where the first three are alternatives to `\hline` and the last one is comparable to `\cline{<i>-<j>}`. The advantage of these commands is that they provide nice spacing around the lines and make your table look a lot better.

```
\begin{tabular}{p{.4\linewidth}r}
    \toprule
    \bf Item & \bf Cost \\
    \midrule
    Pear & 500 \\
    Apple & 300 \\
    \cmidrule{2-2}
    & 800 \\
    \bottomrule
\end{tabular}
```

Item	**Cost**
Pear	500
Apple	300
	800

4.3 Spanning

Spanning is when one cell has the width of e.g. two columns. You can have cells spanning multiple columns, multiple rows, or both at the same time. Cells spanning multiple columns is integrated in LaTeX by default and is available using the `\multicolumn{<num>}{<pos>}{<contents>}`. In here, `num` is the number of columns spanned, `pos` is the formatting of the cell (using the same specifiers as for columns, `l`, `c`, `r`, etc.) and `contents` is the content of that cell. This is often used in headers of tables.

Cells spanning multiple rows is also possible, using the package `multirow`. It provides a command `\multirow{<num>}{<width>}{<contents>}` which is identical in arguments to its column counterpart, except for `width` which requires a width or `*` to just take over the width of over cells in that column. The call of this function should happen in the top row of the ones to be spanned, in the other rows this column can be left empty. As an example:

```
\begin{tabular}{llr}
    \toprule
    \multicolumn{2}{c}{\bf Product} & \\
    \cmidrule{1-2}
    \bf Category & \bf Item & \bf Cost \\
    \midrule
    \multirow{2}{*}{Fruit} & Pear & 50 \\
    & Apple & 30 \\
    \cmidrule{1-2}
    \multicolumn{2}{c}{\multirow{2}{*}{Others}} & 50 \\
    & & 100 \\
    \bottomrule
\end{tabular}
```

Product		
Category	**Item**	**Cost**
Fruit	Pear	50
	Apple	30
Others		50
		100

4.4 Matrices

Matrices are often much easier and require much less styling than normal tables. To have the most control over your matrices, simply use an `array` environment within your mathematical environment. You'll have to put in the brackets yourself, using `\left\right`. However, in the `amsmath` package, there are also some useful predefined environments available to simplify this process.

The base matrix environment available in `amsmath` is `matrix`. This is very similar to a `tabular` or `array`, except that it doesn't require column specifications. You just enter your content separated by an `&` and newlines `\\`. There are some other environments available that add brackets as well:

Environment	Brackets
`pmatrix`	Normal: ( )
`bmatrix`	Square: [ ]
`Bmatrix`	Curly: { }
`vmatrix`	Lines: | |
`Vmatrix`	Double lines: ‖ ‖

As an example matrix, we have:

```
X =
    \begin{pmatrix}
        1 & 2 & 3 & \cdots & 104 \\
        3 & 4 & 5 & \cdots & 106 \\
        \vdots & \vdots & \vdots & \ddots & \vdots
        ↪  \\
        43 & 44 & 45 & \cdots & 146
    \end{pmatrix}
```

$$X = \begin{pmatrix} 1 & 2 & 3 & \cdots & 104 \\ 3 & 4 & 5 & \cdots & 106 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 43 & 44 & 45 & \cdots & 146 \end{pmatrix}$$

5 Figures

In LaTeX, with *figures* we mean objects (e.g. graphs, pictures) with a caption and a number, that can be referenced to in the text. How to reference was already explained in section 2.6. Here I will discuss how to add figures, position them and put some content in them.

5.1 Floating environments

All figures in LaTeX are so-called floating environments. They are blocks of content that can move through the document independently of the text. The two most important floating environments are `figure` and `table`, the first for content such as pictures and graphs, the latter for tables. By default, such a floating environment will be placed on seemingly random position in the document. This position is chosen by the compiler and is often either the top or bottom of a page, in such a way that the flow of the rest of the text will not be affected too much. Floating blocks will also always be put together on one page.

By default, this works relatively well. However, at times when you have many figures this can cause some strange lay-out and you might want to take some control in your own hands. This can be done with the placement modifiers you can specify for your floating environment. The available modifiers are:

- `h` Place the float *approximately* here
- `t` Place the float at the top of the page
- `b` Place the float at the bottom of the page
- `p` Place the float at a special page for floats only
- `!` Overrides default LaTeX behaviour
- `H` Place the float at precisely this location, requires the `float` package

It can sometimes be difficult to correctly place your figures, and with multiple figures it is often a process of trial-and-error.

```
\begin{figure}[t]
    \centering
    \includegraphics[width=.9\linewidth]{bird.jpg}
    \caption{A bird}
    \label{fig:bird}
\end{figure}
```

Figure 1: A bird

5.2 Wrapping text

By default, figures in LaTeX are placed between two lines of text, they take up the entire width of the page. However, sometimes it might be useful to have your text wrap around the figure on the side, especially when you have smaller tables or pictures. For this, use the package `wrapfig`. To initiate a wrapped environment, use `\begin{wrapfigure}[<lines>]{<pos>}{<width>}` where `lines` is the number of lines you want the figure to span (optional), `pos` is the position of the figure (either `r` right or `l` left) and `width` is, of course, the width of the figure.

Example wrapped figure

```
\begin{wrapfigure}{r}{0.3\textwidth}
    \begin{center}
        \includegraphics[width=.9\linewidth]{wrapbird.jpg}
    \end{center}
    \caption{A wrapped bird}
    \label{fig:wrapbird}
\end{wrapfigure}
```

Figure 2: A wrapped bird

The placement of wrapped figures can be more difficult than normal figures. It can be the case that there is a lot of whitespace at the top or bottom of the wrapped figure, which causes text to look weird. This can be solved by inserting several `\vspace{<length>}` commands in your figure with negative lengths. As for normal figures, a lot of trial-and-error is involved.

5.3 Captions

As you have already seen in the figures before, a figures and tables can be given a caption using `\caption{<text>}`. This command will automatically recognize the current environment (e.g. figure and table), and will automatically but the correct text and number in place for you. Each different type of caption uses a separate counter.

By default, the caption system can be somewhat limiting. You might want to give captions to figures/images outside a `figure` environment or style the appearance of your captions. This can be done using the `caption` package, which provides the command `\captionof{<type>}{<text>}` to insert a caption of the given `type`. This can be for example `table` and `figure`.

The `caption` package also enables you to customize the styling of your captions. This can be set in the options of the package, or using `\captionsetup{<options>}`. The package provides many options, such as `width`, `font`, `labelfont` and `textfont`. The option `position` is also interesting, since it specifies where the caption is with respect to the figure/table. This allows the package to correctly set the whitespace.

```
\captionsetup{labelfont=bf,
↪  font=small, width=.8\linewidth}
\captionof{figure}{This is a smaller
↪  caption that is limited in width
↪  and automatically overflows}
You can see the difference when
↪  putting normal, long text next to
↪  it and comparing the width.
```

Figure 3: This is a smaller caption that is limited in width and automatically overflows

You can see the difference when putting normal, long text next to it and comparing the width.

5.4 Subfigures

Sometimes, it is desirable to have multiple items in one figure, such that you have figures 1a, 1b, etc. For this, the package `subcaption` can be used. This package provides the `subfigure` and `subtable` environments that can be nested within an existing `figure` and `table`. Using `\caption{<text>}` you can then give titles to them. As an example:

Subfigures example

```
\begin{figure}[!h]
    \centering
    \begin{subfigure}[t]{0.3155\textwidth}
        \includegraphics[width=\linewidth]{leftkitten.jpg}
        \caption{Left kitten}
    \end{subfigure}
    \begin{subfigure}[t]{0.3741\textwidth}
        \includegraphics[width=\linewidth]{middlekitten.jpg}
        \caption{Middle kitten}
    \end{subfigure}
    \begin{subfigure}[t]{0.2104\textwidth}
        \includegraphics[width=\linewidth]{rightkitten.jpg}
        \caption{Right kitten}
    \end{subfigure}
    \caption{Some kittens}
\end{figure}
```

(a) Left kitten **(b)** Middle kitten **(c)** Right kitten

Figure 4: Some kittens

You see that this works nicely, but the widths of the `subfigure` are set to specific values as to make the pictures align. This is usually not a major problem, but in this case it would be nice if the pictures align properly. To do this automatically, there is some extra code necessary. I won't explain it, but the example is below.

As an alternative to this, the `floatrow` package can also be used. This package is more flexible and allows to do this with any content, not just figures from a file. For instructions on how to make this alignment work, see the documentation, paragraph 11.1.7.

Subfigures with automatic sizing

```
\begin{figure}[!h]
    \centering
    \newsavebox{\rowbox}
    \sbox\rowbox{\resizebox{.9\textwidth}{!}{
        \includegraphics[height=10cm]{leftkitten.jpg}
        \includegraphics[height=10cm]{middlekitten.jpg}
        \includegraphics[height=10cm]{rightkitten.jpg}
    }}
    \newlength{\rowheight}
    \setlength{\rowheight}{\ht\rowbox}\hspace*{\fill}
    \subcaptionbox{Left kitten}{\includegraphics[height=\rowheight]{leftkitten.jpg}}\hfill
    \subcaptionbox{Middle kitten}{\includegraphics[height=\rowheight]{middlekitten.jpg}}\hfill
    \subcaptionbox{Right kitten}{\includegraphics[height=\rowheight]{rightkitten.jpg}}\hspace*{\fill}
    \caption{Some kittens}
\end{figure}
```

(a) Left kitten **(b)** Middle kitten **(c)** Right kitten

Figure 5: Some kittens

5.5 Including pictures

By default, it is not possible to include external pictures in your LaTeX document. The package `graphicx` enables this using the command `\includegraphics[<options>]{<filename>}`. Here, in options you generally put the desired width or height of the picture, and in file you put the filename of the picture **without spaces**. Spaces in filenames tend to break LaTeX.

The file types supported differ for each compiler, but the most commonly used ones (PdfLaTeX, LuaLaTeX) support `jpg`, `png` and `pdf`. The package `epstopdf` also enable you to include `eps` files. For example usage, see the figures above.

5.6 Float barriers

When including many figures in your document, they might get spread further and further down the document and might even 'pollute' next chapters or sections. A built-in way to solve this is using `\clearpage`, a command that puts all floats that do not yet have a position at this place, then goes on to the next page with the rest of the content. This way, several figures might end up at the pages before this command was called.

In some even more specific cases, it might not be desirable to insert a new page to clear all your floats. For example: that was the case with the last subfigure example, it kept drifting off into this section, something I did not want. I also did not want to include a page break here. The solution for this situation is provided by the package `placeins` with `\FloatBarrier` that enables you to place all 'pending' floats at the current spot and then continue with the rest of the document directly.

6 Graphs

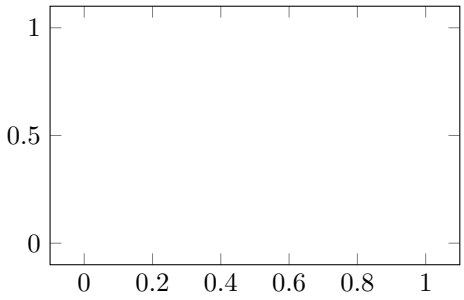
In many scientific reports and articles, graphs are an integral part. There are multiple ways to make graphs and include them in your document. Often, graphs are generated using some external tool (Excel, Matlab, Mathematica), exported as JPG/PNG/PDF and then included as pictures in the document. This is a perfectly fine method, with one major disadvantage: they don't look in place. They generally don't share the LaTeX fonts, its colors, etc. (The one exception to this is DataGraph (OSX only, paid) that generates stunning graphs that look just like native LaTeX ones)

An alternative to using an external tool is to use some graphing library inside LaTeX. The one I use the most is called `pgfplots` which heavily relies on `tikz` to make a range of graphs. In this chapter, I will be explaining the basics of this package. The possibilities are almost endless, as is clear from the enormous examples page. For more details on the options, refer to the 566 page documentation.

6.1 Basics

All fun is happening inside the `axis` environment, which lives inside a `tikzpicture` environment. Therefore, the basic syntax is:

```
\begin{tikzpicture}
    \begin{axis}[options]
        % Add plots
    \end{axis}
\end{tikzpicture}
```



Here, `options` is the place for many options of your graph. These options include things like `title`, axis labels, grid settings and size.

Name	Description
`title`	The title of the graph
`xlabel`	The title of the x-axis (also for `y, z`). This is evaluated inside a maths environment, so use `\text{...}` to get normal text
`width`	The width of the graph
`height`	The height of the graph
`grid`	Whether you want a grid on the background: `major`, `minor`, `none`, `both`
`minor x tick num`	(also for `y, z`) Number of ticks in the minor grid per step in major grid
`xmin, xmax`	(also for `y, z`) The range of x-values to plot
`axis x line`	(also for `y, z`) Where the axis should show. Default both sides of the plot, options include `left`, `right`, `top`, `bottom`, `middle`
`xmode`	(also for `y, z`) Whether the axis should be `log`arithmic or `linear` (default linear)

6.2 Function plots

The easiest plots to do are function plots, so things like plotting x^2, $\sin(x)$ or even more complicated things like $\theta \to \left(\cos\left(\frac{3}{2}\theta\right)\cos\theta, \cos\left(\frac{3}{2}\theta\right)\sin\theta\right)$ and $z = (x+2)^2 + 4y^2$.

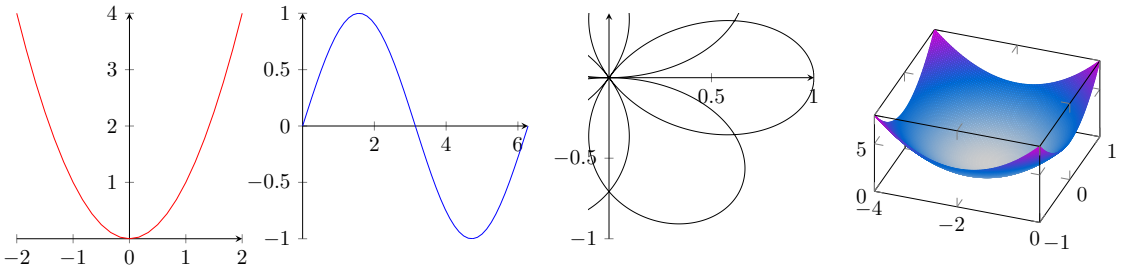


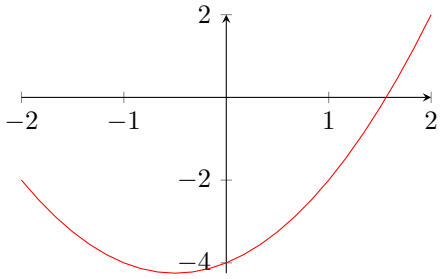
Figure 6: Some example plots that are possible with `pgfplots`

As you can already see, it is possible to plot both 2D- and 3D-functions. Also, both explicit ($y = f(x)$) and parametrized functions ($t \to (x(t), y(t))$) are supported. Unfortunately, implicit functions like the unit circle ($x^2 + y^2 = 1$) are not supported and need to be either parametrized or plotted in multiple parts (one part of each $x_\pm = \pm\sqrt{1-y^2}$ in this case). However, within those conditions, the library is quite flexible and can plot almost anything.

6.2.1 2D plots

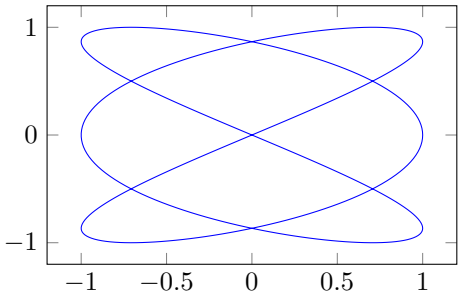
To plot a 2D-function, we use the command `\addplot [<options>] {<function>};` (note the semicolon!). In the options it is possible to define the color of the line, define some domain to be plotted or (useful for parametrized functions) set how many samples should be taken within your plotted domain. Within the curly brackets `{}` the function should be specified in terms of `x`. This gives the following code for our $y = x^2 + x - 4$.

```
\begin{tikzpicture}
    \begin{axis}[axis y line=middle,
    ↪  axis x line=middle]
        \addplot[domain=-2:2, red]
        ↪  {x^2+x-4};
    \end{axis}
\end{tikzpicture}
```



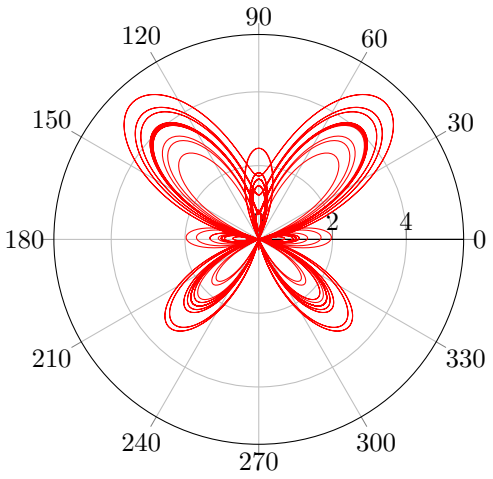
For parametrized functions, replace the function inside the curly brackets with two sets of them wrapped inside normal brackets as follows: `({a(x)},{b(x)})`. The first function describes `x`, the other `y`. Both functions should be defined in terms of `x` (confusing, I know). It is recommended to define the number of samples that should be taken, to avoid sharp edges. As an example, for the famous Lissajous curve:

```
\begin{axis}
    \addplot[domain=0:2*pi, blue,
    ↪  samples=400]
        ({cos(3*deg(x))},
        ↪  {sin(2*deg(x))});
\end{axis}
```



Note that all trigonometric functions in `pgfplots` are based on degrees and not radians, so in order to use them with radians, the input should be converted to degrees using `deg(x)`, as done in the example above.
It is also possible to work with a polar coordinate system, which might be more useful for some functions. For this, the pgfplots library `polar` should be included and all code should be in the `polaraxis` environment. Besides that, the function specified in `\addplot;` should give the radius ($r(\theta)$) instead of y. For a more advanced example, with a butterfly curve, we get:

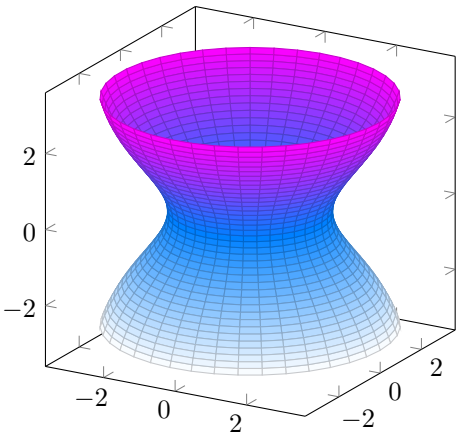
```
% In your preamble
\usepackage{pgfplots}
\usepgfplotslibrary{polar}

% ...
% In your document somewhere
\begin{tikzpicture}
    \begin{polaraxis}
        \addplot[red, domain=0:7200,
        ↪  samples=5000] {...};
    \end{polaraxis}
\end{tikzpicture}
```



6.2.2 3D plots

It is also possible to generate 3D plots in `pgfplots`. However, this will get you into trouble for more complex graphs. Plotting more than 1 function will get you into trouble, and also more advanced functions might give problems with the memory limit or something else. For the simpler functions `pgfplots` is fine, but for more complex ones, it might be best to use an external tool.
The syntax of 3D plots is quite similar to 2D plots, except that the function to call is now called `\addplot3 [<options>] {<function>};` where the 3 indicates that this is a 3D function. The options are again quite similar, with now an added `y domain` for the second variable. The function to be specified between the `{}` is now a function of $z = f(x, y)$, or again parametrized with $t \to (x(t), y(t), z(t))$ (so you have to specify 3 variables!).
The `axis` environment also has some extra options that are useful for 3D plots. With `view={a}{b}` it is possible to define the angle from which you view the graph and with `3d box=complete` it is possible to get lines on all edges of the 'box' that is plotted. In the `addplot3` command, options have been added for the domain of `y` (`y domain`) and the number of samples of `y` (`samples y`). For 3D surfaces there are multiple plot types, `mesh` is for contours only (like a net), `surf` is to fill the surface completely (and show contour lines).
It is also possible to color your surface, to make it clearer what its z-value is. This can be done using color maps, that have to be set in the options of your plot command. The syntax for this is `[colormap/name, <options>]` where name is the name of the color map you want to use. Some of the predefined color maps are: `hot`, `jet`, `blackwhite`, `cool` and `bluered`.
As an example, below the parametrized function $(t, u) \to \left(2+2\tanh^2\left(\frac{1}{2}t\right)\cdot\cos(u);\ 2+2\tanh^2\left(\frac{1}{2}t\right)\cdot\sin(u);\ t\right)$ is plotted, with a color map and 40 samples in both variables. Note that the option `z buffer=sort` is defined, this is to make sure that the elements in the back don't overlap the elements in the front.

```
\begin{axis}[colormap/cool, view={30}{20}]
    \addplot3[surf, z buffer=sort, samples=40,
    ↪  samples y=40, domain=-3:3, y
    ↪  domain=0:2*pi]
        (
            {(2 + 2 * tanh(.5*x)^2)
                * cos(deg(y))},
            {(2 + 2 * tanh(.5*x)^2)
                * sin(deg(y))},
            {x}
        );
\end{axis}
```



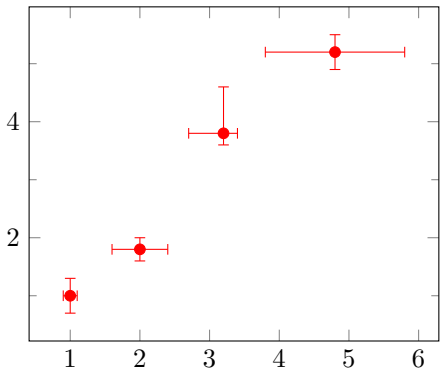
6.3 Data plots

A very important and often-used plot is of course a plot of measurement data. Various types of plots are available, such as scatter plots, bar plots, stacked plots, filled plots (area under the line is colored), boxplots and many more. In this document, only scatter plots will be discussed. For the other types of plots, refer to the documentation of `pgfplots`. All other types of plots are very similar to scatter plots and require only one or two options to be set.
In this part we will focus on getting your measurement data in your graph. There are two main ways of plotting your data points. The first is by defining them all in the `tikzpicture` code as coordinates. The other is by using a separate data file. The latter is the easiest if you export your data from another program such as Matlab or Mathematica, and the only option if you want to fit a function through the data later on.

6.3.1 Simple coordinates

To plot simple coordinates you can use `\addplot [<options>] coordinates {<coords>};` where again `options` is for your options and in this case you put a list of coordinates inside the curly brackets `coords`. It is also possible to specify error bars for your data points, see the example below. More information on error bars can be found in section 6.4. A useful option for these plots is `only marks` that hides the line that (by default) connects the data points. A basic example showcasing the possibilities:

```
\begin{axis}[minor y tick num=1]
    \addplot[red, only marks,
            error bars/.cd,
            x dir=both, x explicit,
            y dir=both, y explicit
        ] coordinates {
            (1,1) +- (0.1, 0.3)
            (2,1.8) +- (0.4, 0.2)
            (3.2,3.8) += (0.2, 0.8)
            ↪  -= (0.5, 0.2)
            (4.8,5.2) +- (1, 0.3)
        };
\end{axis}
```



6.3.2 External source

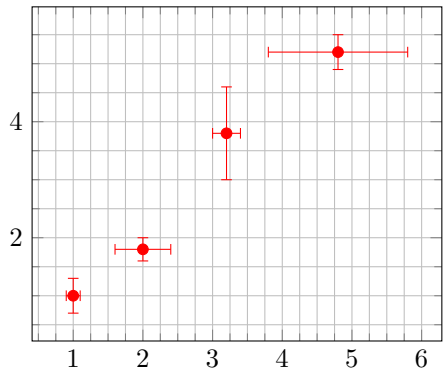
The basic coordinates method demonstrated above is easy to start with, but with larger data sets it becomes cumbersome to have all data in your latex code all the time. Also it might require a lot of manual labour to convert all your data to the correct format, especially if the data is extracted from another program such as Matlab or Mathematica. The solution for this is to save your data in any external file, which should be formatted as simple text with columns separated by white space (at least one tab or space) and rows separated by newlines. Other options are possible using the `pgfplotstable`.

Note on including your data in your LaTeX file

Sometimes it might be inconvenient to use separate files for your measurement data and if you're like me you'd like to keep all the data in one file. Even when using external files as a data source for your graphs, it is still possible to define your data inside latex. Simply include the package `filecontents` and put your data inside the `filecontents` environment using `\begin{filecontents}{<filename>} data \end{filecontents}`. The contents of the environment will then be stored in the file with your filename.

To use an external file as data source for your graph, put `table` in your `\addplot;` command: `\addplot [<options>] table [<data specification>] {<filename>};`. Inside, there is the usual `options` part for the usual options. New is the `data specification`, here you define what column is the data for what axis. The `filename` is quite straightforward, but keep in mind that **file names in LaTeX cannot have spaces in them**! This might work in some places but usually yields very unpredictable behaviour.
In this case, an example is again the easiest way to show how this works:

```
% Data format:
% X   Y   X_err   Y_err
% 1   1   0.1     0.3
% ...

\begin{axis}[
        minor y tick num=3,
        minor x tick num=3,
        grid=both
    ]
    \addplot[red, only marks,
            error bars/.cd,
            x dir=both, x explicit,
            y dir=both, y explicit
        ] table [
            x=X, y=Y,
            x error=X_err, y error=Y_err
        ] {<filename>};
\end{axis}
```

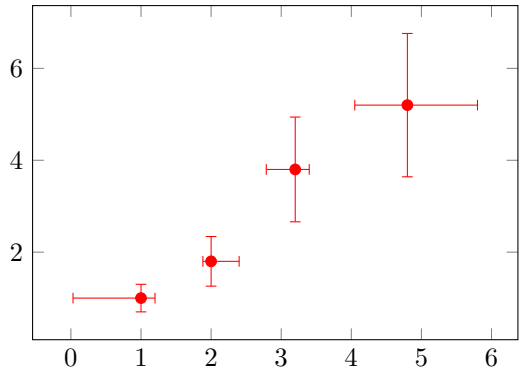


6.4 Error bars

Another useful and important feature in `pgfplots` is the ability to show error bars. The error bars are quite flexible, it is possible to define absolute and relative values, but also expressions/functions such as $\sqrt{x}$. Furthermore, it is possible to define separate error values for both the positive and negative directions (so-called asymmetric errors). To enable error bars on any (data) plot, include the option `error bars/.cd` in the `addplot` command. Include this as the last option, all other options you define after this will be about the errors.
First, you have to define what directions you want your error bars to appear in. For this, use `x dir` (or `y` or `z`) and specify what type of bars you want. The options are `none` (no error bars),

`plus` (only in the positive direction), `minus` (only in the negative direction) and `both` (error bars in both directions). The actual value has to be specified separately. In case of `both` it is also possible to specify different values for the positive and negative direction.
The easiest way to add error bars to your measurements is to include them in your data set, as done in section 6.3. Then in your options, define that the error is given explicitly, by adding `x explicit` (or `y`, `z`). It is also possible to make the error constant for all your measurement values, then add the options `x fixed = value` where you replace `value` by the error value. For both cases there is also the alternative `x explicit relative` (and `x fixed relative`) that states that the given value is a relative error (a factor, 1 = 100%).
For the `table` plot type it is also possible to specify separate positive and negative errors, put this information in separate columns and include them in the options using `x error plus` and `x error minus` and omit the default `x error`. More advanced is the ability to make each of these possible errors a mathematical expression rather than a value. This is done using `x error expr` (similar for other axes). The expression given can make use of the data of other columns in the same row using `\thisrow{<column>}`.
The example below showcases all functionality. The absolute error in positive x direction is given in the data file, the absolute error in negative x direction is given by the expression $\bar{x} = \sin^2(100x)$ (remember that sin works in degrees), the relative error in both directions for y is 30%.

```
\begin{axis}
    \addplot[red, only marks,
            error bars=./cd,
            x dir=both, x explicit,
            y dir=both, y fixed relative=.3
        ] table [
            x=X, y=Y,
            x error plus=X_err+,
            x error minus
            ↪  expr={sin(\thisrow{X}*100)^2}
        ] {<filename>};
\end{axis}
```



6.5 Fitting a function

When you have plotted your data, the next logical thing to do is to fit some function through your data. Often, you'll have some simple linear relationship $y = ax + b$ where you want to determine your a and b. In other cases you might want to fit more complex functions such as $y = a\sin(bx + c)$ through your data. Luckily, both are possible, but unfortunately not with `pgfplots` only. To properly fit a function, we need the help of the external tool `gnuplot`.

6.5.1 Setting up `gnuplot`

When working online in either Overleaf or ShareLatex, everything is already set up correctly and you can start working with `gnuplot` directly, so you can skip this part. When working offline, first install `gnuplot` from the download site. Make sure to add `gnuplot` to your `PATH` (option in the setup). After that, your LaTeX program needs to be set up correctly: the parameter `--shell-escape` needs to be added to the compile command. For some programs I added instructions below, for your specific program please search online.

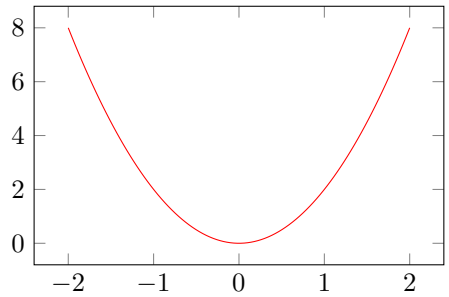
TeXstudio To modify the compile command, go to Options -> Configure TeXstudio -> Commands. Then add `-shell-escape` just after the part with `-synctex=1` in the top 4 entries (LaTeX, PdfLaTeX, XeLaTeX and LuaLaTeX).

TeXpad TeXpad has functionality built-in that automatically enables this option. If, for some reason, this doesn't work, you can manually enable it by switching the option 'Shell Escape' in the typeset configuration (click on the arrow next to 'Typeset' in the toolbar, choose 'Manual' instead of 'Auto-sense').

6.5.2 Calling `gnuplot`

`gnuplot` is a powerful plotting tool that can plot many different types of graphs. The program is script-based, so you need to put in a script to get out your graphs. It can be called using the command `\addplot [raw gnuplot, <options>] gnuplot {<code>};` where again, `options` are the usual options for the plot and `code` is the code to be executed. I will not discuss the full functionality of `gnuplot` here, since that is beyond the scope of this document. For more information, refer to the documentation. As a quick example:

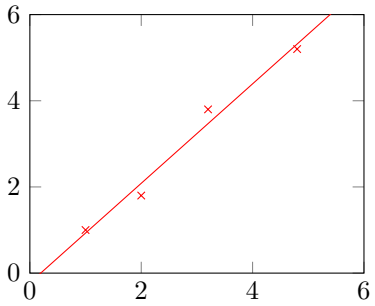
```
\begin{axis}
\addplot[raw gnuplot, red] gnuplot {
    f(x) = a*x^2;
    a = 2;
    plot [x=-2:2] f(x);
};
\end{axis}
```



6.5.3 Making a fit

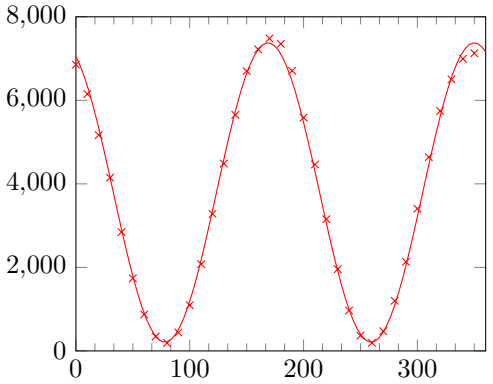
To do a fit through some test data, it is easiest to have your data in an external file, as outlined in section 6.3.2. Below, I will provide an example with the basic script to do a linear fit on some data. It is easy to adapt this to your situation, by changing `f(x)`, adding more variables, or changing the indexes of your columns.

```
\addplot[red, only marks, mark=x] table [x=X, y=Y]
↪ {<filename>};
\addplot[raw gnuplot, red] gnuplot {
    # Function to fit
    f(x) = a*x + b;
    # Initial guesses for all variables
    a = 1; b = 1;
    # Fit command.
    fit f(x) '<filename>' using 1:2 via a,b;
    # Plot the function for the given domain
    plot [x=0:6] f(x);
};
```



The `fit` command is used to calculate the best fit for the variables through the data and has the following syntax: `fit <function> <filename> using <x>:<y> via <vars>;` where `function` is the function to fit, `filename` is the file with the data, `x` and `y` are the indices of the columns for X and Y (starting at 1) and `vars` are the variables in the function that have to be fitted. A more complex example, based on a real experiment I did some time ago:

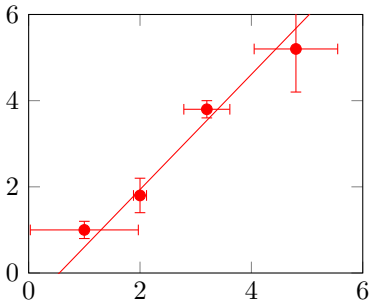
```
\addplot[red, mark=x, only marks] table [x=X, y=Y]
↪ {<filename>};
\addplot[raw gnuplot, red, mark=none] gnuplot {
    f(x) = a*sin(b*x + c) + d;
    a = 3000; b=-2; c=100; d=4000;
    # Marks that sin should work with degrees
    set angles degrees;
    fit f(x) '<filename>' using 1:2 via a,b,c,d;
    plot [x=0:360] f(x);
};
```



6.5.4 Incorporating error bars

It is also possible to fit data sets while incorporating errors in the measurement data. These errors should be stored in separate columns in your data file. The commands for incorporating errors are:
`fit <function> <filename> using <x>:<y>:<xerror> xerror via a,b;`
`fit <function> <filename> using <x>:<y>:<yerror> yerror via a,b;`
`fit <function> <filename> using <x>:<y>:<xerror>:<yerror> xyerrors via a,b;`
for errors in x-direction only, errors in y-direction only and errors in both directions, respectively. Here `<xerror>` and `<yerror>` are the column indices of the columns where the error value is stored. Each of the column index markers can be replaced with an expression as well. Put this expression in between normal brackets `()`. Within such an expression it is possible to refer to the other columns using `$<i>` where `i` is the index of the other column. As an example:

```
\addplot[
    red, only marks, error bars=./cd,
    x dir=both, x explicit,
    y dir=both, y explicit
] table [
    x=X, y=Y,
    x error expr={sin(\thisrow{X}*100)^2},
    y error=Y_err
] {<filename>};
\addplot[raw gnuplot, red, mark=none] gnuplot {
    f(x) = a*x + b;
    a = 1; b = 1;
    set angles degrees;
    fit f(x) '<filename>' using
    ↪ 1:2:(sin($1*100)^2):3 xyerrors via a,b;
    plot [x=0:6] f(x);
};
```

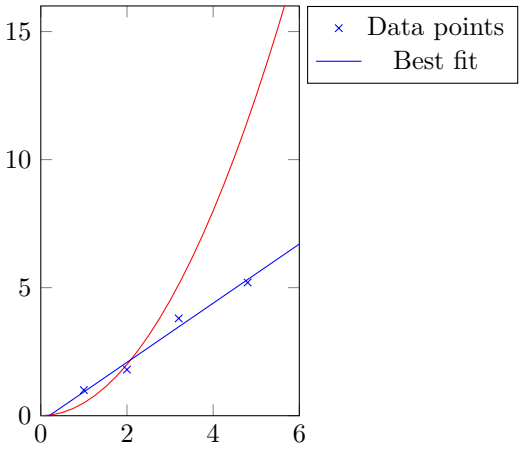


6.6 Legend entries

Another important aspect of a graph is its legend, especially if multiple lines or data sets are plotted. Luckily, generating the legend is relatively easy. Simply put the command `\addlegendentry{<name>}` (without semicolon!) after the `\addplot;`. Note that this will add all plots in order, so if you do two `\addplot;`'s and after that an `\addlegendentry`, the entry added will be for the first plot. In order to skip a plot, add the option `forget plot` to the `\addplot;` command.
To determine the legend position, use the `legend pos` option for the `axis` environment. Possible values are each diagonal wind direction (`north east`, `south west`, etc) that will place the legend

in one of these corners of the graph. Only `north east` can also be combined with `outer` (i.e. `outer north east`), to place the legend outside the graph, next to it.
It is also possible to label your plots using `\label{<name>}` and refer to them using `\ref{<name>}`, the reference will then be the line style of the plot you refer to. Note that this does not work when using the `external` library, as explained in 6.7.2, you'll find a workaround there as well. As an example to combine all options:

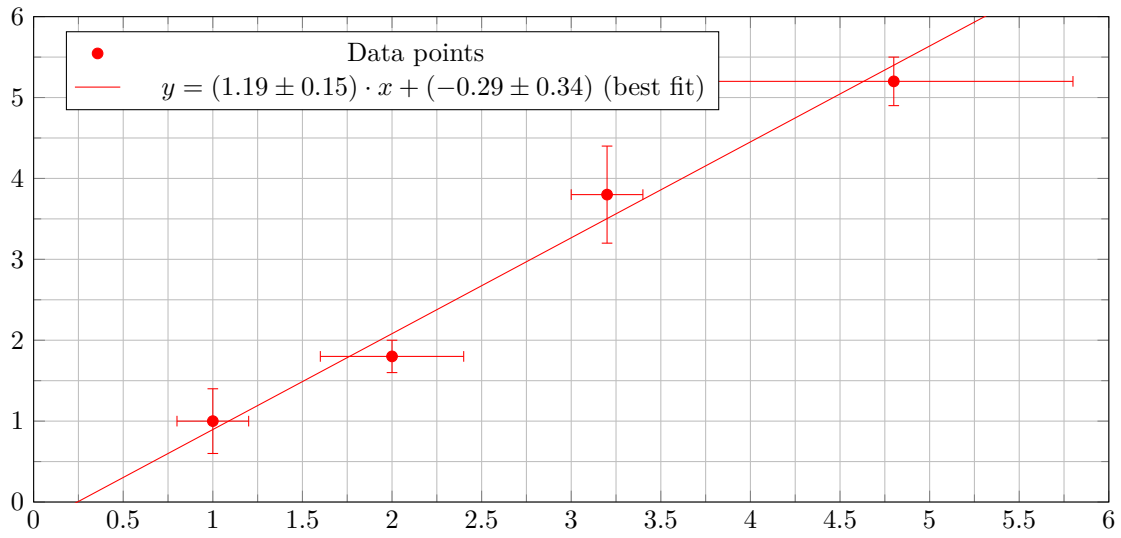
```
\begin{tikzpicture}
\begin{axis}[legend pos=outer north east]
    \addplot [red, domain=0:6, forget plot]
    ↪ {.5*x^2};
    \addplot [blue, only marks, mark=x] table
    ↪  [x=X, y=Y] {<filename>};
    \label{plot:data}
    \addlegendentry{Data points}
    \addplot [raw gnuplot, blue] gnuplot {
       % Code for linear fit
    };
    \label{plot:fit}
    \addlegendentry{Best fit}
\end{axis}
\end{tikzpicture}

Here, \ref{plot:data} is the measured data and
↪  \ref{plot:fit} is the best linear fit.
```



Here, × is the measured data and × is the best linear fit.

Something extra that might be interesting to do is to include your fit-parameters from `gnuplot` into your legend. This way you immediately show the fitted parameters, even including their error! For this, you'll have to extend your `gnuplot` script and do some trickery in the `\addlegendentry`. An example:

```
    f(x) = a*x + b;
    a = 1; b = 1;
    # Indicates that we want to know the error in the fit, creates for each fitted variable
    # another variable with name <>_err where <> is the name of the variable.
    set fit errorvariables;
    fit f(x) '<filename>' using 1:2:3:4 xyerrors via a,b;
    plot [x=0:6] f(x);

    # Output variables to a file
    set print '<other_filename>';
};
\addlegendentry{
    % Read the data in the specified file and store it in the command \out
    \pgfplotstableread{<other_filename>}\out

    % Get the elem on row 0 and column 0 of data in \out and store it in \outA
    % same for column 1-3 and \outB-D
    \pgfplotstablegetelem{0}{0}\of\out \pgfmathsetmacro\outA{\pgfplotsretval}
    \pgfplotstablegetelem{0}{1}\of\out \pgfmathsetmacro\outB{\pgfplotsretval}
    \pgfplotstablegetelem{0}{2}\of\out \pgfmathsetmacro\outC{\pgfplotsretval}
    \pgfplotstablegetelem{0}{3}\of\out \pgfmathsetmacro\outD{\pgfplotsretval}

    % Print numbers in \outA-D in the correct place and format
    $y = (\pgfmathprintnumber{\outA} \pm \pgfmathprintnumber{\outB}) \cdot x +
    ↪  (\pgfmathprintnumber{\outC} \pm \pgfmathprintnumber{\outD})$ (best fit)
}
```



6.7 Improving performance

This might all work fine with a few figures, but as soon as there are a couple of them, with some of the more advanced features (fitting, 3D), the compilation time can quickly increase dramatically and memory usage can become a problem. There are two main ways to deal with this: use LuaLaTeX, or cache your figures.

6.7.1 Using LuaLaTeX

LuaLaTeX is a new, alternative compiler that has many advantages over the 'traditional' compilers such as PdfLaTeX. The main advantage is that LuaLaTeX dynamically allocates memory so doesn't run into memory issues very quickly, and it is a lot quicker and more suitable to compile these graphics. There are still some minor incompatibilities, but for normal usage you should not encounter any difficulties. If you have a modern LaTeX installation, chances are that you already have it installed. In your editor, simply pick the right compiler. This works differently for each editor.

TeXstudio Go to Options -> Configure TeXstudio -> Build. For the item 'Default Compiler', choose `txs:///lualatex`.

TeXworks In your toolbar in the top left, choose 'LuaLaTeX' in the list.

TeXpad Click on the arrow next to 'Typeset' in the toolbar, choose on 'Manual' and enter LuaLaTeX at 'Format'.

Overleaf With your document open, click on the cog in the top-right to open the settings menu, for 'LaTeX Engine' choose LuaLaTeX.

6.7.2 Cache your figures

By default, all your graphs would be re-generated each time you compile your document. Especially if you have lots of graphs, such as this document, this can take ages. The solution for this is to cache the figures: generate them once and simply use the stored version in your final document. For this, `tikz` provides the library `external`. To start using this library, put `\usetikzlibrary{external}` and `\tikzexternalize` somewhere in your preamble. Keep in mind

that this requires the `-shell-escape` option to be set, for instructions on how to do this, see section 6.5.1.
Once you included the package, you should be all set and LaTeX will do its job. However, by default all output files will be stored in the same folder as your `.tex` file and it will quickly become a mess. I advise to create a directory specifically for storing these cache files, and specifying that folder as output folder with `\tikzexternalize[prefix=<folder name>/]` (pay attention to the `/` after the folder name!).
By default, when figures have errors, the compiler will not show them easily anymore and will simply crash without good reason. For this, it is easiest to disable the caching for this one figure only, by setting `\tikzset{external/export next=false}` just before the figure so it will skip this figure for the caching. Possible errors will then pop up.

Note on the use of `external` with plot references

As explained in section 6.6 it is possible to refer to specific plots with `\ref{<name>}`. This call will draw the icon of the plot itself, which itself is a figure, hence it will be externalized by this library. This will lead to some conflicts that are described in the manual of `tikz`. The easiest workaround is to skip the externalization for these references using `\tikzset{external/export next=false}`, put that just before the `\ref{}` call and everything should be fine

Note on the use of `external` with the custom vectors

The improved vector, defined in section 3.6, makes use of a tikz figure. Externalizing these will give errors, so uncomment the line that sets that the next line should be skipped (remove the %) to prevent any trouble.