Example Document

A modern LaTeX cookbook

Alex Povel

May 11, 2022

1st Examiner	Prof. Jane Doe
2nd Examiner	Prof. Foo Bar
Supervisor	John Doe, M.Sc.

University of Greatness
Institute of Big Bang

Name	**`cookbook` `(CENSORED VERSION)`**
Compiled on	**May 11, 2022 5:51pm Z**
	`013973e` `(HEAD)`
Engine	LuaHBTeX, Version 1.15.0 (TeX Live 2022)
LaTeX Version	LaTeX 2_ε (2021-11-15)
Glossary	`glossaries-extra` + `bib2gls`
Bibliography	biblatex + biber
Generator	`latexmk`
Class	2021/11/13 v3.35 KOMA-Script

Task

For:	**Alex Povel** [666]
Topic:	**A modern LaTeX cookbook** (Example Document)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

- First you are doing this,
- then another thing,
- lastly that.

You will be working on this a lot.

Alex Povel *Place & Date*

Declaration of Authorship

I, Alex Povel, hereby declare to be the sole, independent author of the Example Document submitted here. No other than the cited references have been used. Any content directly or indirectly obtained from external sources has been marked up as such. This thesis has neither been submitted to a second examination authority nor been published.

Alex Povel	*Place & Date*

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

- We found this out,
- and also that!

Contents

Glossary

Marked (*) symbols possess a mass-specific variant in which the symbol is written in lowercase and the unit is extended by kg^{-1}. Alternative symbols are specified following the / mark. International symbols are specified in brackets.

Symbols

Roman	Description	Page List	Unit
A	Area	17, 50	m^2
C	Heat capacity*	43	J/K
c	Velocity/Speed	16, 17	m/s
H	Enthalpy*	16, 17	J
H_i	(specific) Heating Value*	20, 38	J/kg
$J_\mathrm{el.}$	Electric current	49	A
k	Count/Quantity	16	–
m	Mass	17, 18, 49, 50	kg
M	Mach number	46	–
n	Component	16	–
p	Pressure	16, 43, 46	Pa
R	(specific) Gas Constant*	16	J/(kg K)
$R_\mathrm{el.}$	Electrical resistance	49	Ω
Re	Reynolds number	48	–
r	Radius	46	m
T	(absolute) Temperature	16, 17, 46	K
t	Time	17	s
$U_\mathrm{el.}$	Voltage	17, 49	V

V	Volume*	16–18, 50	m^3
x	First Cartesian coordinate (length)	16, 17	m
x	Moisture Content	44	$g_w/kg_{da.}$
y	Second Cartesian coordinate (width)	17	m
z	Third Cartesian coordinate (height)		m
ρ	Density	16–18, 38, 46	kg/m^3

Greek	**Description**	**Page List**	**Unit**
α	Angle	16, 46	rad
γ	Mass fraction	20, 38, 46	–
ζ	Drag Coefficient	48	–
κ	Ratio of specific heats	16, 43	–
φ	Relative Humidity		°C
ϑ	(relative) Temperature	16, 43	°C

Other	**Description**	**Page List**	**Unit**
$\lvert\square\rvert$	Absolute of $\square$	17	$\square$
$\overline{\square}$	Arithmetic mean of $\square$	17	$\square$
$\dot{\square}$	$\square$ as a flow quantity	17, 50	$\square/s$
$\Delta\square$	Difference in $\square$	16, 17	$\square$
$\mathrm{d}\square$	infinitesimal change in $\square$	15, 17	$\square$
$\tilde{\square}$	Logarithmic mean of $\square$	17	$\square$
$\partial\square$	partial, infinitesimal change in $\square$	16, 17	$\square$
$\nabla\square$	Vector of partial derivatives (Nabla operator) of $\square$	17	–
$\mathbf{x}$	Vector	17	–

Numbers

Symbol	**Description**	**Phys. Qtt.**	**Page List**
e	Euler's number	2.718 28...	16
i	Imaginary unit	$\sqrt{-1}$	16
π	Pi	3.141 59...	16, 30

Subscripts

a.	air	**i**	inferior
d	dry	**th**	thermal
el.	electric	**w**	Water

Acronyms

Notation	**Description**	**Page List**
CD	Continuous Delivery	5
CI	Continuous Integration	5
COP	Coefficient of Performance	50
CTAN	Comprehensive TEX Archive Network	
GUI	Graphical User Interface	
HFO	Heavy Fuel Oil	38
IDE	Integrated Development Environment	8, 9
IMO	International Maritime Organization	38

Notation	**Description**	**Page List**
ISO	International Organization for Standardization	21, 44, 45
JSON	JavaScript Object Notation	9
MDO	Marine Diesel Oil	38
PDF	Portable Document Format	4, 6, 10, 13, 33, 34
SSOT	Single Source of Truth	28, 33
SVG	Scalable Vectors Graphics	33, 34
URL	Uniform Resource Locator	21
WYSIWYM	What You See Is What You Mean	28

Glossary without page lists

The following styles do not contain page lists of the entries' occurrences, leading to a cleaner, more concise look. Refer to the source code on how to achieve this (which options and styles to use).

Symbols

Roman	**Description**	**Unit**
A	Area	m^2
C	Heat capacity*	J/K
c	Velocity/Speed	m/s
H	Enthalpy*	J
H_i	(specific) Heating Value*	J/kg
$J_\mathrm{el.}$	Electric current	A
k	Count/Quantity	–
m	Mass	kg
M	MACH number	–
n	Component	–
p	Pressure	Pa
R	(specific) Gas Constant*	J/(kg K)
$R_\mathrm{el.}$	Electrical resistance	Ω
Re	REYNOLDS number	–
r	Radius	m
T	(absolute) Temperature	K
t	Time	s
$U_\mathrm{el.}$	Voltage	V
V	Volume*	m^3
x	First Cartesian coordinate (length)	m

x	Moisture Content	$g_w/kg_{da.}$
y	Second Cartesian coordinate (width)	m
z	Third Cartesian coordinate (height)	m
ρ	Density	kg/m^3

Greek	**Description**	**Unit**
α	Angle	rad
γ	Mass fraction	–
ζ	Drag Coefficient	–
κ	Ratio of specific heats	–
φ	Relative Humidity	°C
ϑ	(relative) Temperature	°C

Other	**Description**	**Unit**
$\lvert\square\rvert$	Absolute of $\square$	$\square$
$\overline{\square}$	Arithmetic mean of $\square$	$\square$
$\dot{\square}$	$\square$ as a flow quantity	$\square/s$
$\Delta\square$	Difference in $\square$	$\square$
$d\square$	infinitesimal change in $\square$	$\square$
$\tilde{\square}$	Logarithmic mean of $\square$	$\square$
$\partial\square$	partial, infinitesimal change in $\square$	$\square$
$\nabla\square$	Vector of partial derivatives (Nabla operator) of $\square$	–
$\mathbf{x}$	Vector	–

Numbers

Symbol	**Description**	**Phys. Qtt.**
e	Euler's number	2.718 28...
i	Imaginary unit	$\sqrt{-1}$
π	Pi	3.141 59...

Subscripts

a.	air
d	dry
el.	electric
i	inferior
th	thermal
w	Water

Acronyms

Notation	**Description**
CD	Continuous Delivery
CI	Continuous Integration
COP	Coefficient of Performance
CTAN	Comprehensive TEX Archive Network
GUI	Graphical User Interface
HFO	Heavy Fuel Oil
IDE	Integrated Development Environment
IMO	International Maritime Organization
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MDO	Marine Diesel Oil
PDF	Portable Document Format
SSOT	Single Source of Truth
SVG	Scalable Vectors Graphics

Notation	Description
URL	Uniform Resource Locator
WYSIWYM	What You See Is What You Mean

List of Figures

List of Tables

List of Examples

List of Code

List of Reactions

Preface

This document is not a beginner guide. There is a wide choice of those out there already, both free and paid for. However, what is lacking is a collection of modern, or even at least current, best practices. If you scouted through package documentations and StackExchange long enough, you would eventually get an idea of what is current, idiomatic LaTeX and what is not. This is how I learned LaTeX too, so I cannot recommend any useful beginner books. You might want to check out Overleaf though, an online LaTeX editor[1] with a large knowledge database.

This document is an attempt at collecting best practices — or at least, useful approaches — and pointing out the old ones they could, often should, and sometimes absolutely have to replace. See also here for a second opinion on the topic. More than most other languages, the LaTeX code in circulation world-wide is quite aged. While that code does not necessarily get *worse*, it also does not exactly age like cheese and wine would. To that end, notice how the packages integral to this document are actively maintained and kept modern.

Usage as a Cookbook This document is supposed to be used in a *cookbook* style. That is, it is not meant to be read cover to cover (declaring it a cookbook is also a useful excuse for the mess I made).

The document contains various "recipes", most of which exist in parallel and are independent of one another. The goal is to answer questions of the form *How can this and that be done?* A quick search in the (printed) output or the source code is supposed to deliver answers.

Usage as a Template Considerable effort went into the class file, aka the template. It pulls all settings together, producing an output that is very different from vanilla LaTeX. Therefore, feel free to rip out all the cookbook content, keeping just the settings files to be used as a template.

Source Code This document is meant to be read side-by-side with its source code. That is why there is almost no source code in the printed output itself. If you are curious how a certain output is achieved, navigate to the source code itself. This approach was

[1]I do not recommend using online editors. You are putting your hard work onto some remote, foreign server, relying on their ongoing availability and forfeiting the chance of understanding LaTeX, in case you have to continue locally. Theses containing confidential material should also not be hosted externally.

chosen since, plainly, code does not lie, while annotations and comments might. So while the printed output will always remain true to its actual source code, duplicating that source code so it can be read in the printed output directly is just another vector for errors to creep in.

Source Repository The source repository for this document is at

https://github.com/alexpovel/latex-cookbook .

Any references to the "source" or "repository" refer to that project.

Chapter 1

Usage

This document "requires"[1] Docker. On a high level, Docker allows to prepare specific software bundles, tailor-made for whatever application the bundle author desires. Other users can then use these software bundles for their own projects. The bundles are well-defined and can be distributed and run very easily. The more complex and demanding the required software, the higher the benefit of using such bundles. The driving design principle and primary use case for these bundles is *isolation*. Whatever you do with the bundles, it happens in isolation from your host system. This allows a user to have, for example, arbitrary Python versions at their disposal, whereas a local system installation can only ever offer a single version.

For a LaTeX document, the one at hand here is pretty complex. This is owed to the many used packages and outside tooling. Outside tooling (programs other than LaTeX called from within LaTeX) is quite prevalent in LaTeX, since it itself is so limited.[2] Let us look at the outside tooling used for this document.

1.1 Outside tools and special packages

This section highlights the pain points of *not* using Docker. If you are already familiar and need no convincing, skip to Section 1.2. For this document, outside tools are required for:

1. **`glossaries-extra`**: requires the outside tool **`bib2gls`**, see Section 2.7.1, which in turn requires a Java Runtime Environment.
2. **`biblatex`**: requires the outside tool **`biber`**, see Section 2.4.1.
3. **`svg`**: requires the outside program InkScape. Examples for that package's main command, `\includesvg`, are Figures 3.1a to 3.3 and 3.13.

[1]It does not *technically* require Docker, but I hope to convince you that the non-Docker, manual way is the way of the dodo and a big no-no.

[2]For example, when writing Python, you would not call Perl or JavaScript from within it, because whatever they can do, Python can. The same analogy does *not* hold for LaTeX: base LaTeX can do surprisingly little, even though TeX is technically Turing-complete.

4. Using `contour gnuplot` commands with **`pgfplots`**, see Figure 3.7, requires gnuplot.
5. Syntax highlighting for source code, see Chapter 4, is done through **`minted`**. It requires Python (built into virtually all Linux distributions but needs to be installed on Windows) with its pygments package installed. That package does the heavy lifting and **`minted`** inserts the result into your LaTeX document.

If you have a proper LaTeX distribution, `bib2gls` and `biber` will already be available as commands and ready to be invoked. The latter means that they are on your `$PATH`, *i.e.* the paths to the respective binaries are part of the `PATH` environment variable. *All other stuff, you have to install manually*.

You will have to install and keep everything else updated by hand. What if one of the dependencies of this document conflicts with something you have already installed on your system? What if that conflict is not resolvable? You would be unable to use this document. This is where Docker comes into play. You outsource all this setting-up to someone who already bundled all that stuff together. Docker calls these "bundles" *images*. There is a Docker image *tailor-made* for this document:

alexpovel/latex.

It is guaranteed to function correctly for this document, since the author maintains both in parallel. There will never be a mismatch. If maintenance ceases, it will cease for both components at the same time, hence it will still continue to work, just not get updated anymore.

1.2 Using Docker

Instead of all of the above, **only one installation is required**: Docker. You can get it from

`https://docs.docker.com/get-docker/`.

You don't even need a LaTeX distribution. All you need is an editor to edit the files in, see Section 1.2.3. Once you want to compile, open a terminal, navigate to your directory and run

```
docker run --rm --volume ${PWD}:/tex alexpovel/latex
```

for PowerShell or

```
docker run --rm --volume $(pwd):/tex alexpovel/latex
```

for bash. That's it!

The command consists of:

- The `--rm` option, which removes the run container after you are done (containers are "instances" of images). This is generally desired, since containers are ephemeral and should be treated as such. Do not keep containers around, simply create new ones! If you need adjustments, adjust the image, not the container, then create new containers from that adjusted image.
- The `--volume` option makes the current directory (`pwd`) available *inside* the running container, at the `/tex` destination path. The LaTeX process needs your files to work with. Without this option, the container would be "naked", with no way of accessing your files.
- The final argument to `docker run` is the image to be run, in this case `alexpovel/latex`, which will look for that name on DockerHub.

Ideally, the command can be registered as the "compilation" command of your editor. That way, you just hit the compile button and will be using Docker in the background, with no changes to your workflow.

1.2.1 Compilation steps

You are probably used to running `pdflatex` or similar on your source files, as many times as needed. So where does that step happen in the above `docker` command?

The Docker approach uses the **`latexmk`** tool to ease all the painful labour of running chains of `pdflatex`, `biber` *etc*. manually. `latexmk` automates LaTeX compilation by detecting all the required steps and then running them as often as required. It requires **Perl**. Linux users will already have it available, Windows users may grab Strawberry Perl. As such, this document's processing pipeline *as a whole* requires Perl, although it is technically not required for document compilation only.

Once Perl is installed (of course, the Docker image already contains it), the entire document can be compiled by **simply calling `latexmk`**. You do not even have to provide a `*.tex` file argument. By default, `latexmk` will simply compile all found `*.tex` files. **The core ingredient to this magic process is the `.latexmkrc` configuration file**. You can find it in the repository root directory. It is tailored to this document and does not need to be touched if the compilation process itself has not changed. It also contains some more insights to the entire process.

`latexmk` is great because it figures out most things by itself and enjoys wide-spread acceptance and adoption. If it does not figure out everything from the get-go, it is easily customized, like for this document.

Having walked through all this manually, hopefully using the prepared Docker image instead makes more sense now. It is guaranteed to work for everyone, because the Docker container (that is, the virtual build environment) will be identical for all users. It is independent of local LaTeX installations and all their quirks. As such, it simply and forever does away with the entire, huge class of

> *But it works on my machine!* (*Everyone, at some point*)

Good riddance to that.

If all of this is embedded into a pipeline on GitLab, GitHub Actions, or similar, your documents are built whenever you `git push` to the remote server (or whenever you configure it to). It does not get simpler; the downside is of course the lengthier setup. Also, the repository itself is a live demonstration where everything is set up already!

Slow compilation

The most prevalent downside to `latexmk` are longer compilation times. The tool examines changes in auxiliary and source files. This is made possible by the way LaTeX works with its kind of unique way of writing out auxiliary files (a system of multiple passes). Consider a simplified, manual example chain:

1. You have a `test.tex` file (and nothing else!), with `cref` or equivalent cross-referencing commands in it.
2. You compile the file *once*, *e.g.* using `lualatex`.
3. A `test.aux` file (and probably others, *e.g.* `.toc`) will have been generated, as well as the Portable Document Format (PDF) file itself.
   The PDF has literal **??** markers where there should be references. This is because to resolve those references, and print *e.g.* the correct page number, LaTeX needs to look into the `.aux` file. But that file was just generated and was not available yet for the first compilation run.
4. Hence, you compile again.
   The **??** markers are now replaced by their properly spelled-out reference. However, what if, hypothetically, one of these **??** substitutions now reads *very super long reference to page 37 in the document*. It is so long that subsequent content has shifted again. That content is now on different pages. This triggered another change in the auxiliary files.
5. You compile *again*, the third time. Things have settled, no changes happened this time anymore. You are done.

`latexmk` works by detecting this convergence towards a steady-state (which may, through bugs, never be reached). However, it would compile *an additional* time after Item 5, to make sure or because it cannot really be certain (just talking from experience here, not knowledge). So while you gain deterministic, automated builds, they will take much longer.

What if you do not care if there are **??** markers left, undefined entries, shifted pages and so forth, maybe because you are only drafting things up? You can simply compile yourself once, manually, *e.g.* using `lualatex`. This will speed things up by a long shot, but is considered advanced usage. It does, as opposed to `latexmk`, not "just work" anymore, and beginners might wonder what those **??** are. Now that you know, you may handle them.

Faster compilation The base docker commands were shown at the beginning of Section 1.2. An alternative is to run

```
docker run --rm --volume $(pwd):/tex alexpovel/latex -e '$max_repeat=1'
```

This limits latexmk to only run commands once (-e is for additional Perl code), with the following effects:

- latexmk might complain that it did not succeed for lack of repeats (including a non-zero exit code, meaning failure when it was only partial failure).
- Results are compiled quickly.
- latexmk and its tailor-made config is still used under the hood, so you do not have to adjust anything.
- Other commands like biber for bibliography generation might still be run.

This is probably your best bet to get faster drafting cycles while retaining high likelihood of it working.

Fastest compilation The following method is the most advanced (but not hard at all!). To overwrite the Docker image's default, so-called **ENTRYPOINT**, which is latexmk, run the command as:

```
docker run --rm --volume $(pwd):/tex --entrypoint="lualatex" alexpovel/latex
 ↪  --shell-escape $FILENAME
```

Notice that you now have to give a filename, like cookbook.tex, since lualatex expects that. The above command is exactly like running *only* lualatex, just in the context of that image (with your files made available at /tex). The effects of this command are:

- No latexmk involved at all:
  - it is therefore the fastest approach,
  - but you have to pass all required options yourself (like --shell-escape).
- No commands like biber run; you have to do that yourself (or use latexmk).

This is the approach I personally use, while outsourcing the full latexmk-based compilations to a server using Continuous Integration (CI)/Continuous Delivery (CD). It affords us the fastest iteration speeds while having a full version available "in the background": the best of both worlds and arguably the fastest workflow.

Fastest still slow We cannot compile faster than a naked lualatex run (docker overhead is negligible). For the current document you see here, even such a single run takes about one minute. Remember that you need upwards of three runs from a "cold" (no auxiliary files, empty caches, ...) start. Single, article-size, pdflatex-based documents compile in only a few seconds. The reasons for the slowness are, among others:

- `lualatex` is simply considerably slower than `pdflatex` (the additional capabilities make up for that many times over). One reason is the time spent dealing with (system) fonts.
- This document loads a metric crapton of packages, some of which are very expensive to load, like **`tikz`**. This is in the spirit of providing a useful document with numerous examples of various functionalities and packages. You should definitely **comment out the packages you do not need**.
- Compiling the entire document takes long, so **make use of the `include` and `subinclude` commands and comment out unneeded parts**. The more fine-grained you do this, the more control you get. If you cut it down to short (only a handful of PDF pages) files, you can work on and compile only those. This method has the highest time saving potential.
- If you perform **`pgfplots`** calculations within LaTeX, *c.f.* Figures 3.6a to 3.7, expect significant slowness. For simple plots, this is still much better than having to use an external program, export data, import to LaTeX, repeat on data change (instead of adjusting the equation in LaTeX directly) *etc.*
- LaTeX is not compiled in parallel.

This document is like an aircraft carrier: modern, feature-rich and high-impact, but very sluggish.

1.2.2 More on Docker

You do not need to know of the entire chain of how Docker images are created and run. Only consuming the final image has all the benefits with little effort. However, the process is not complex:

1. A **`Dockerfile` text document** is created, containing instructions on how the image should look like (like what stuff to install, what to copy where, ...).
   As a baseline, these instructions often rely on a Debian distribution. As such, all the usual Debian/Linux tools can be accessed, like `bash`.
   An (unrelated) example Dockerfile can look like:

```
# Get the latest Debian Slim with Python installed
FROM python:slim

# Update the Debian package repositories and install a Debian package.
# Agree to installation automatically (`-y`)!
# This is required because Dockerfiles need to run without user interaction.
RUN apt-get update && apt-get install -y ffmpeg

# Copy a file from the building host into the image
COPY requirements.txt .

# Run some shell command, as you would in a normal sh/bash environment.
# This is a Python-specific command to install Python packages according to some
# requirements.
RUN pip install -r requirements.txt

# Copy more stuff!
COPY music-converter/ music-converter/

# This will be the command the image executes if run.
# It runs this command as a process and terminates as soon as the process ends
# (successfully or otherwise).
# Docker is not like a virtual machine: it is intended to run *one* process, then
# die. If you need to run it again, just create a new container (instance of a
# Docker image). Treat containers as *cattle*, not as a *pet*. The
# container-recreation process is light-weight, fast and the way to go.
#
# Of course, this does not stop anyone from running one *long-running* process
# (as in infinity, `while True`-style). This is still a good use-case for Docker
# (as are most things!). An example for this is a webserver.
ENTRYPOINT [ "python", "-m", "music-converter", "/in", "--destination", "/out" ]
```

The Dockerfile this project uses for LaTeX stuff is here:

https://github.com/alexpovel/latex-extras-docker/blob/master/Dockerfile

It is not as simple, so not as suited for an example. Its length gives you an idea of the setup required to compile this LaTeX document. All of that complexity is of no concern to you when using Docker! Of course, such an image also works for much simpler documents.
If you require custom additions, you can always inherit from existing base images:

```
FROM alexpovel/latex

# ... Your stuff goes here ...
```

2. The **image** is then built according to the `Dockerfile` instructions, resulting in a large-ish file that contains an executable environment. For example, if we install a comprehensive TeXLive distribution, the image can be more than 2 GB in size. Note that you will never interact with that "file" directly. Docker manages it for you, and all interaction occurs through the `docker` command.
The Docker image can be distributed. If you just instruct to run an image called *e.g.* `alexpovel/latex`, without specifying a full URL to somewhere, Docker will look on its Hub for an image of that name (and find it here). Anyone can pull (public) images from there, and everyone will be on the same page (alternatively, you can build the image from the Dockerfile).
For example, as stated, the LaTeX environment for this project requires a whole bunch of setting-up. This can take more than an afternoon to read up upon, understand, implement and getting to run. In some cases, it will be impossible if some required part of a project conflicts with a pre-existing condition on your machine. For example, suppose project *A* requires Perl in version `6.9.0`, but project

B requires version `4.2.0`. This is what Docker is all about: isolation. Whatever is present on your system does not matter, only the Docker image/container contents are relevant.
Further, if you for example specify `FROM python:3.8.6` as your base image, aka provided a so-called tag of `3.8.6`, it will be that tag in ten years' time still. As such, you nailed the version your process takes place in and requires. Once set up, this will run on virtually any computer running Docker, be it your laptop now or whatever your machine is in ten years. This is especially important for the reproducibility of research.
3. Once the image is created, it can be run, **creating a container**. We can then enter the container and use it like a pretty normal (usually Linux) machine, for example to compile our LaTeX files. Other, single commands can also be executed. The proper way is to run one container *per process*. If that process (*e.g.* `latexmk`) finishes, the container exits. A new process then requires a new container.

1.2.3 Editor

You are free to do whatever you want. However, a garbage editor can substantially hamstring your work. For example, please do not use Notepad++. It is a fantastic little program but unsuitable for any serious, longer work.

The author uses and dearly recommends Visual Studio Code, using its LaTeX Workshop extension, which provides syntax highlighting, shortcuts and many other useful things. Visual Studio Code is among the most state-of-the-art editors currently available. Being usable for LaTeX is just a nice side-effect we can take advantage of. It is open-source and therefore also has a privacy-respecting alternative fork, *VSCodium*.

For a more conventional, complete Integrated Development Environment (IDE), try TeXStudio. Like VSCode, it is also open source. TeXStudio will cater to 90 % of your LaTeX needs, but nothing else (Markdown, ...).

Visual Studio Code

The repository of this document comes with a `.devcontainer` directory. In it is all the configuration necessary to run your development environment *inside* the Docker container entirely. To make the magic work, install the *Remote – Containers* extension. Visual Studio Code automatically prompts you to install it when you first open this repository, as configured in `.vscode/extensions.json`. Follow the instructions there on how to download all prerequisites, like of course Docker. Visual Studio Code will detect the remote container configuration of this repository automatically. If not, run the steps manually:

1. Open the command palette (Ctrl + ⇧ + P — the command palette is one of the defining, core features of Visual Studio Code, it's great!).
2. Run Remote-Containers: Reopen in container.

Your development environment is now *inside* the container. Hence, if you ran for example `latexmk` in the console, it executes the container version. This is exactly what we want and previously did using the `docker run` command.

Extensions There is a small problem. On its own, Visual Studio Code has no concept of LaTeX, since out of the box, it is much closer to being an editor than an IDE. You increase the program's capabilities by choosing and installing extensions. Using the *LaTeX Workshop* extension for Visual Studio Code is recommended. For the container environment, it is installed automatically from the settings found in `devcontainer.json`. The *LaTeX Workshop* extension turns Visual Studio Code into a LaTeX IDE. It works using *recipes*, which in turn use *tools*. The extension comes with a pre-defined `latexmk` tool:

1. Open the command palette.
2. Run LaTeX Workshop: Build with recipe > latexmk.

This just runs `latexmk` within the container. As usual using Docker, your host machine does not need a LaTeX installation for this to work. If you want, you can stop here — running `latexmk` is all that is ever needed. Hit Ctrl + Alt + B to build the project again, using the last used recipe.

Settings You can configure *everything* in Visual Studio Code, including its extensions, in the settings. Open them by either:

- The command palette at Preferences: Open Settings (JSON), or
- navigating to File > Preferences > Settings, then opening as JavaScript Object Notation (JSON).

For example, as discussed at the end of Section 1.2.1, you might want to have a recipe that only runs `lualatex`, once. This is **already taken care of** for this repository:

```json
{
    "latex-workshop.latex.recipes": [
        {
            "name": "lualatex",
            "tools": [
                "lualatex"
            ]
        },
        {
            "name": "latexmk",
            "tools": [
                "latexmk"
            ]
        }
    ],
    "latex-workshop.latex.tools": [
        {
            "name": "lualatex",
            "command": "lualatex",
            "args": [
                "--shell-escape",
                "--synctex=1",
                "--recorder",
                "%DOC%"
            ]
        },
        {
            "name": "latexmk",
            "command": "latexmk",
            "args": [
                // latexmk doesn't need a file argument, but provide it to avoid runnin
 ↪  g
                // on *all* found tex files
                "%DOC%"
            ]
        }
    ],
    "latex-workshop.latex.recipe.default": "lastUsed",
    "latex-workshop.chktex.enabled": true,
    "latex-workshop.latex.autoBuild.run": "never",
}
```

These are *Workspace Settings*, which take precedence over (global) user settings. As such, they also override all the recipes *LaTeX Workshop* usually comes with. You will have to define more recipes yourself.

With the new recipes, you can now run

LaTeX Workshop: Build with recipe ⟩ lualatex

from the command palette. Notice the freedom and flexibility you have for defining many more recipes. In fact, alongside `latexmk` and classic `make`, you now have more potential flexibility and automation than one could possibly need.

Using the above *Remote – Containers* approach, the entire development chain happens inside the container, with your working directory mounted into it. This enables:

- the *LaTeX Workshop* extension to work fully:
  - SyncTeX works:
    - Ctrl + Click into the PDF preview goes to the source code
    - LaTex Workshop: SyncTeX from cursor (command palette) goes the other way.
  - Highlighting infos, warnings and errors (*Problems* pane) works
  - IntelliSense works, providing autocompletion for:

    * citations (`autocite` *etc.*) — check out the *Citation Browser* in the command palette!
    * references (`cref` *etc.*)
    * and more, like commands
  - Auto-Formatting (⇧+Alt+F) works (*LaTeX Workshop* uses `latexindent`)
- Your files are persisted normally, despite the container being ephemeral.
- Tools like `make` work from within the Visual Studio Code terminal. For example, run `make clean` to clear everything not tracked by git. This is very convenient for a cold start, because LaTeX sometimes gets stuck if auxiliary files are corrupted.

More Settings For more control, you might want to keep the extension from compiling on every document save:

```
"latex-workshop.latex.autoBuild.run": "never"
```

See the official documentation for more.

Other Editors

You will have to incorporate the `docker` commands shown in Section 1.2 manually. Most editors do not support container-native developing, so you will go the old-school route.

Chapter 2

Base Features

Let us first go through some of the base features offered by LaTeX and used in this template. Before anything happens, the TeX engine is chosen. This is the binary responsible for compiling the LaTeX source code. If you have used LaTeX before but never heard those terms before, you are most likely using pdfLaTeX as your engine. This is the go-to default for generating PDF output. Its most popular modern alternatives are XeLaTeX and LuaLaTeX. These two offer various new, shiny features, chief among which is Unicode support through the **`unicode-math`** package, which builds onto **`fontspec`** and extends it with math fonts capabilities. Note that these packages *require* XeLaTeX or LuaLaTeX use: pdfLaTeX is *old* and will no longer work. More on that in Section 2.1.

LuaLaTeX is preferred over XeLaTeX for the following reasons:

1. **`contour`**, a package that can do things like this, works. For an application of that, see Figure 3.4 on Page 41. Getting **`contour`** to work on XeLaTeX is somewhat impossible, see also here, here, here and here.
2. LuaLaTeX allocates as much memory as it happens to need. The ancient limitations of TeX are easily reached by packages like **`tikz`** and **`pgfplots`**. Circumventing that either feels hacky, or actually is hacky. `tikzexternalize` is a great module, but has some caveats, like breakage of clickable references. It solves a problem that should no longer exist in the first place.
3. The fantastic **`microtype`** package has a number of unavailable functionalities when using XeLaTeX.

Using LuaLaTeX, you do not have to worry about any of that: hit compile, wait, done[1].

2.1 Fonts and Text

Using high-quality fonts is one goal. This includes the fantastic TeXGyre fonts, of which the *Palatino* (by Hermann Zapf) clone *Pagella* was chosen for this document. It comes

[1]The *waiting* part might be a bit longer than what you might be used to from pdfLaTeX. So far, this is the only disadvantage I could find.

Table 2.1a / Examples for font features offered by the main roman font. Notice the small vertical white space after the fourth row, nicely separating content that is slightly dissimilar.

Feature	Sample Text
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
Bold	**The quick brown Fox jumps over the lazy Dog 13 times!**
Italics	*The quick brown Fox jumps over the lazy Dog 13 times!*
Bold Italics	***The quick brown Fox jumps over the lazy Dog 13 times!***
Small Capitals	The quick brown Fox jumps over the lazy Dog 13 times!
Bold SC	**The quick brown Fox jumps over the lazy Dog 13 times!**
Italics SC	*The quick brown Fox jumps over the lazy Dog 13 times!*

with an accompanying math font of the same name[2]. There are not very many high-quality free fonts with a math companion available, see also this compilation. A similar compilation of "nice" fonts is found here, however that is specifically for pdfLaTeX, not LuaLaTeX. The font choice can be changed with relative ease in the options for the `unicode-math` package. If a new font does not provide at least the same array of features, parts of the document might break! For reference, there is a list of symbols defined by `unicode-math`.

Using both normal and maths fonts in conjunction is made possible through the `unicode-math` package. As such, an exact match of the text and math fonts is achieved. This is important but often overlooked. However, such a match is often plain unavailable in the typesetting tool at hand. LaTeX is no exception here if there simply exist no matching fonts. We took care to ensure a complete match here.

Not only do the fonts look great on their own and paired up, they (just as importantly) also feature very broad support for all sorts of symbols and characters, as well as font shapes and weights and combinations thereof. The latter is demonstrated in Table 2.1a.

Combine all that with `microtype` (a package taking care of typesetting details), and we get very beautiful typesetting. For example:

> *Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut*

[2]Both fonts are vector fonts; if LaTeX yields any warnings about *font size substitutions*, that is bogus and can be ignored.

metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Notice the balanced line endings and low number of hyphenation; easy on the eyes and very readable.

Old approach The overwhelming majority of LaTeX documents relies on the two lines

```
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
```

This approach is **outdated** and only required when using pdfLaTeX. Since this is what most people still do, the two packages are still required. However, these two packages should at least not be used for *new* work anymore! Using LuaLaTeX, they are not required anymore:

- input encoding is automatically UTF-8 (as it should be in `$CURRENTYEAR`),
- font encoding is also not required, since fully capable fonts are used, which come with all the glyphs required.

The same is true for XeLaTeX.

2.1.1 Math

What often does not occur at first glance is that many documents use text and math fonts that are very different from one another. As far as I know, Microsoft's *Word* has usable math typesetting and sensible default fonts for that. Yet, it cannot do what dedicated, fully-fleshed text and math fonts — different, but matched — can achieve. They provide a seamless transition, especially when using actual text in the math environment or vice versa, like when we go for $x \to \infty$ and then also do $\int_1^2 y^2 \mathrm{d}y$ or maybe $a^2 + b^2 = c^2$. All these inline math elements look perfectly natural. If instead the fonts did not match, inline math would stick out like a sore thumb. Toggle the colors in the class file (`*.cls`) to highlight each different font family to see all the differences (see the `Color` option for **`unicode-math`**). Some display-style math examples follow.

$$pv = RT \tag*{[2.1]}$$

$$e^{i\pi} + 1 = 0 \tag*{[2.2]}$$

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln(x)} = 1 \tag*{[2.3]}$$

$$\boldsymbol{\sum_{k=0}^{n} \binom{k}{n} = 2^{k}} \tag*{[2.4]}$$

$$\left[T \cancelto{0}{\frac{\partial}{\partial T}} + \kappa(p)\frac{\partial}{\partial \rho} + \alpha\right] \vartheta^{k}(x_1, x_2, \ldots, x_k) = 0 \tag*{[2.5]}$$

$$\Delta h_{\text{change}} = {}^{1}\!/\!{}_{2}\left(c^{2}_{\text{exit}} - c^{2}_{\text{entry}}\right) \tag*{[2.6]}$$

If these symbols are colored, it is because they are links and link coloring is on. This can be turned off (globally or for certain elements) in the options to **`hyperref`**. If color is turned off, they are still hyperlinks. If even that is undesired, that can also be turned off in that package.

Predefined Macros

The **`physics`** package has some issues, and as such, this document relies on a couple custom macros in place of it, see Table 2.2.

Symbols

Note how the symbols in equations are hyperlinks (leading to their definition in the glossary), courtesy of packages **`hyperref`** and the powerful **`glossaries-extra`**. Having not specified anything else, they take on whatever hyperlink color was specified in the preamble. In the final document, all links should be hidden, aka black. This is a given for print output, but probably also sensible for digital output. The visual noise introduced by colored links is quite immense. Their only point is to let users know there is something clickable — there is a good chance that is figured out anyway. If at all, use a dark green or blue tone.

Math Highlighting

In a very unobtrusive yet also unambiguous way, we can highlight important results, as shown in Equation 2.3. This feature is a natural part of **`tcolorbox`**, a very powerful

Table 2.2 / Predefined math macros. Refer to the source code for help on the syntax. For derivatives, the starred macros yield partial derivatives.

Name	Examples
Mean	$\bar{\rho}, \overline{A}$
Logarithmic Mean	$\tilde{\rho}, \tilde{A}$
Absolute[a]	$\lvert\rho\rvert, \lvert A\rvert, \left\lvert\frac{A^2}{A^3}\right\rvert$
Flow	$\dot{m}, \dot{H}$
Delta	$\Delta V, \Delta h$
Nabla Operator	$\nabla x, \nabla^3 x$
Vectors	$\mathbf{x}, \mathbf{A}$
Derivatives	$\mathrm{d}m, \mathrm{d}^2x, \partial m, \partial^2 x$
Fractional Deriv.	$\frac{\mathrm{d}U_{\mathrm{el.}}}{\mathrm{d}t}, \frac{\mathrm{d}^2y}{\mathrm{d}x^2}, \frac{\partial U_{\mathrm{el.}}}{\partial t}, \frac{\partial^2 y}{\partial x^2}, \frac{\mathrm{d}h}{\mathrm{d}T}, \frac{\mathrm{d}^2h}{\mathrm{d}T^2}, \frac{\partial h}{\partial T}, \frac{\partial^2 h}{\partial T^2}$
Time Deriv.[b]	$\frac{\mathrm{d}U_{\mathrm{el.}}}{\mathrm{d}t}, \frac{\mathrm{d}^2y}{\mathrm{d}t^2}, \frac{\partial U_{\mathrm{el.}}}{\partial t}, \frac{\partial^2 x}{\partial t^2}$
Positional Deriv.[b]	$\frac{\mathrm{d}V}{\mathrm{d}x}, \frac{\mathrm{d}^2c}{\mathrm{d}x^2}, \frac{\partial V}{\partial x}, \frac{\partial^2 c}{\partial x^2}$
Parentheses[a]	$(x), (x_2), (x^2), \left(\frac{x}{y}\right), \left(\frac{x^2}{y^3}\right)$
Brackets[a]	$[x], [x_2], [x^2], \left[\frac{x}{y}\right], \left[\frac{x^2}{y^3}\right]$
Braces[a]	$\{x\}, \{x_2\}, \{x^2\}, \left\{\frac{x}{y}\right\}, \left\{\frac{x^2}{y^3}\right\}$

[a]These scale automatically according to their content, using `mathtools`.
[b]As a shortcut version of fractional derivatives.

package for anything color and boxes. Note that for print, the gray tones should probably be darkened.

Indices and Operators

Indices are typeset upright! They are text. Math output like x_{upper} is one of the most common mistakes. If text occurs in math mode, that is also actual text. Consider

$$MEAN_{sample} \neq 2\,, \tag{2.7}$$

which looks stupid. What Equation 2.7 really says is "$M \cdot E \cdot A \cdot N$": slanted characters are variables, and not specifying any operator implies multiplication. This example is pedantic, but it is very easy to imagine such an issue evolving into real ambiguity, which would then be a serious error. Since multiplication is implied by doing nothing, leaving out `*` aka `\cdot` altogether often looks best.

In the preamble, use `\DeclareMathOperator{\examplemean}{MEAN}`, then use `\text` for the subscript:

$$\operatorname{MEAN}_{\text{sample}} \neq 2\,. \tag{2.8}$$

Text-Flow

Equation 2.8 is part of the surrounding sentence. Math is just another written "language" (the only one understood around the globe!) which can and will be read as a natural part of the surrounding text. As such, it should contain punctuation marks. So that now, having considered that we have the amazing result of

$$1 \neq 2\,, \tag{2.9}$$

we have achieved better overall reading flow, with commas where there would naturally be separations when reading the sentence out loud as a whole. Notice the small spaces `\,` before the commas and dots in equations. They are convenient to avoid ambiguities. Implementing these as `\eqcomma` and `\eqend` ensures consistency. Additionally, these punctuations can then also be switched off globally if your requirements demand so.

A second example often occurs when symbols are defined. Notice the difference between these:

1. The density ρ is defined in Equation 2.10.

$$\rho := m/V \tag{2.10}$$

2. We can therefore define the density as

$$\rho := m/V\,. \tag{2.11}$$

The example in Item 2[3] reads more naturally, is less clunky, there is less duplication of code and information and the reader does not have to jump around references. Nevertheless, embedding equations into the surrounding text is unfortunately somewhat subjective.

Physical Units

In the name of all that is holy, *never* manually type out units, numbers or quantities yourself: use **siunitx**. The package is an absolute must-have if there are any units to be typeset. It is also worth using if there are no units, but numerals. The latter can be typeset using the `\num` command. It is capable of parsing number constructs:

`\num{1.0}`	$\rightarrow 1.0$	
`\num{1,0}`	$\rightarrow 1.0$	(Localisation)
`\num{1.0}`	$\rightarrow$ 1,0 (German)	
`\num{1.2e3}`	$\rightarrow 1.2 \times 10^{3}$	
`\num{1.2e-7}`	$\rightarrow 1.2 \times 10^{-7}$	
`\num{-e7}`	$\rightarrow -10^{7}$	
`\num{3.2(9)}`	$\rightarrow 3.2(9)$	(Uncertainty)
`\num{300000000}`	$\rightarrow 300\,000\,000$	(Group separation)

If the roman text font uses hanging numerals (like: 1234567890), but a stand-alone number needs to be typeset, `\num` is also required. In this document, it will use the math font: $1\,234\,567\,890$.

Physical units are output using `\unit`. It also has parsing capabilities:

```
\unit{\meter\cubed\kelvin\per\kilogram\squared\per\giga\watt\per\degreeCelsius}
```

will print $\mathrm{m^3\,K/(kg^2\,GW\,^{\circ}C)}$. The mode in which units with negative exponents are displayed can be changed, *e.g.* fractions can be used: $\frac{\mathrm{m^3\,K}}{\mathrm{kg^2\,GW\,^{\circ}C}}$. You can also use `\DeclareSIQualifier` to create textual subscripts for units, further specifying them, or create entirely custom units using `\DeclareSIUnit`. An example for both combined is: $\frac{\mathrm{MWh_{th}}}{\mathrm{m^3}}$.

The `\qty{<quantity>}{<unit>}` command essentially combines `\num` and `\unit`, inserting a small space in between, for correct typesetting. No or full spaces are wrong and painful to read.

There are also commands to print ranges: 20 to 50 m. Refer to the package documentation for more.

[3]Items in lists can be labeled and referenced using **enumitem** and **cleveref**.

Chemistry

Closely related to purely mathematical equations are chemical ones, as shown in Reaction {2.1}.

$$\mathrm{C} + \mathrm{O_2} \longrightarrow \mathrm{CO_2} \tag*{\{2.1\}}$$

Aligned arrays of multiple chemical reaction equations are also possible, as demonstrated in Reactions {2.2} and {2.3}.

$$\mathrm{N_2} + \mathrm{O} \xleftrightarrow{k_1} \mathrm{NO} + \mathrm{N} \tag*{\{2.2\}}$$

$$\mathrm{N} + \mathrm{O_2} \xleftrightarrow{k_2} \mathrm{NO} + \mathrm{O} \tag*{\{2.3\}}$$

Single chemical compounds are invoked using `\chcpd`, as demonstrated in Equation 2.12.

$$\boxed{\frac{H_\mathrm{i}}{1 \times 10^6} = 35\gamma_\mathrm{C} + 94.3\gamma_\mathrm{H} + 10.4\gamma_\mathrm{S} + 6.3\gamma_\mathrm{N} - 10.8\gamma_\mathrm{O} - 2.44\gamma_\mathrm{w}} \tag*{[2.12]}$$

These commands are provided by **chemmacros** and its **chemformula** module. Those packages are very capable, so more complex chemistry can also be typeset:

$$\mathrm{{}^{227}_{90}Th^{+}}$$

$$\mathrm{[Cu(NH_3)_4]^{2+}}$$

$$\mathrm{RNO_2^{-\bullet}}$$

$$\mathrm{CH{\equiv}CH}$$

$$\mathrm{\{[CH_2{=}CH{-}CH_2]^{-} \longleftrightarrow [CH_2{-}CH{=}CH_2]^{-}\}}$$

$$\mathrm{H_2C{-}C{\equiv}C{-}CH{=}CH^{+}} + \mathrm{CrO_4^{2-}} \xrightleftharpoons[\mathrm{y}]{\mathrm{x}} 2.5\,\mathrm{Cl^{-\bullet}} + 3\frac{1}{2}\,\mathrm{Na \cdot OH_{(aq)}} + \underset{\mathrm{name}}{\mathrm{A^{n}}}$$

All these examples are courtesy of the documentations of these two packages.

2.1.2 Sans-Serif

Having taken care of the main, aka roman, font, the sans-serif font is the logical next step. Fontin Sans is a decent such font. Importantly, and in contrast to most other free sans-serif fonts, it comes with all the bells and whistles required for stunts, see Table 2.2a. This even includes small-capitals support.

Table 2.2a / Examples for font features offered by the sans-serif font.

Feature	Sample Text
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
Bold	**The quick brown Fox jumps over the lazy Dog 13 times!**
Italics	*The quick brown Fox jumps over the lazy Dog 13 times!*
Bold Italics	***The quick brown Fox jumps over the lazy Dog 13 times!***
SC	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!

Table 2.2b / Examples for font features offered by the monospaced font. The italics are indeed faked and not real glyphs, just slanted regular shapes.

Feature	Sample Text
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
Bold	**The quick brown Fox jumps over the lazy Dog 13 times!**
Italics	*The quick brown Fox jumps over the lazy Dog 13 times!*

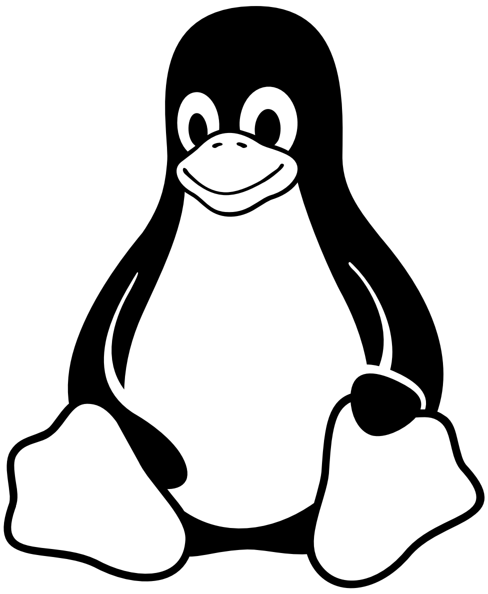
2.1.3 Mono-Spaced

Wanting to display any sort of code, or maybe a Uniform Resource Locator (URL), in the document will have you looking for a mono-spaced aka typewriter font. Inconsolata, published by Google, is the pick here. See Chapter 4 for a more in-depth demonstration. At the time of writing (April 15, 2020)[4], Inconsolata has native regular and bold faces, but no italics. However, such a feature can be faked using **`unicode-math`**/**`fontspec`**, where the regular font is *slanted* to give an italics-like result. Italics are a nice but uncritical feature for code listings, so this should be okay. See for yourself in Table 2.2b.

2.1.4 Symbols

`fontawesome5` is a package providing hundreds of freely usable vector symbols. They scale just like normal vector graphics and can be used alongside text, like this little friend here: ◕. They can also be colored (◉) or scaled:

[4]Date typeset as `\DTMdate{YYYY-MM-DD}` using **`datetime2`**. This gets us **language-independent** International Organization for Standardization (ISO) formatting.



All of these should also be select- and then copyable, yielding *e.g. LINUX*. Refer to the package documentation for a list of all the available symbols.

2.2 Language

Closely related to the previous section are the language features. Currently, there are two supported languages: English and German. See for yourself in the file responsible for the translations, `translations.tex`. It leverages the `\providecaptionname{}` command from KOMA-Script. Given the name of that command, it is probably a misuse and not its purpose, but it worked better than its alternatives.

The **`polyglossia`** package does all the heavy lifting for this document. It is an alternative to the widespread **`babel`** package. Note that **`polyglossia`** requires the language to be passed to its `\setdefaultlanguage{}` command. It will normally not work by just specifying a global document language. However, you can pass a global language to the `language=` keyword in the options to `\documentclass{acp}`, and it will be picked up correctly.

The chosen language option will also be forwarded to various other packages, like **`enquote`** for quotations and **`siunitx`** for localization of numerals typesetting. **`polyglossia`** will take care of all the hyphenation rules, so there should be no or very minimal need for manual `\hyphenation{}` or `\-` commands.

The language can be switched dynamically, and all content will be adjusted, see Table 2.3. More languages can be added easily in the translations file. Note that the `listing` type of **`cleveref`** will only work with the main document language and is currently not able to be switched dynamically.

Table 2.3 / Available languages and dynamic switching. Note how the commands are language-dependent; no manual adjustments necessary.

Language	Quotes	Numerals	Headers *etc.*	Hyphenation
English	“Quote!”	100.2 m	Glossary, Figure 2.1, Table 2.1a, Chapter 2	This is a random text intended to show of hyphenation and line endings in the respective language. It is therefore a bit longer, so hopefully some words will break at the word boundaries. This cannot be guaranteed however. So there is to hoping that this test shows off the hyphenation rules and capabilities put into place by `polyglossia`.
German	„Zitat!“	100,2 m	Glossar, Figure 2.1, Table 2.1a, Chapter 2	Dies ist ein zufälliger Text, der die Worttrennungsregeln und Satzenden der jeweiligen Sprache demonstrieren soll. Er ist daher etwas länger, sodass manche Wörter hoffentlich an den Seitenrändern gebrochen werden. Dies kann allerdings nicht garantiert werden. Wir können also nur hoffen, dass dieser Test die Silbentrennungsregeln und -fähigkeiten des `polyglossia` Pakets.

2.3 Sectioning

2.3.1 Explanation

Between any two sectional commands (from `\part` down), there always should be be at least *some* content. Anything else looks off, *cf.* the direct transition between Sections 2.3 and 2.3.1. At least, tell the reader what the following hierarchical level contains for them.

Further, three numbered levels of hierarchy are enough. The numbering and even availability of sectioning commands depends on the used `documentclass`. In **`koma-script`** articles (`scrartcl`), these are sections, subsections and subsubsections. **`koma-script`** reports (`scrreprt`) and books (`scrreprt`) extend this by parts and chapters, both of which numbered. Subsubsections will no longer be numbered. Trying to create a `\part` or `\chapter` in article classes will result in errors.

Deeper!

Deeper sectioning is fine, but it should not be numbered or occur in the table of contents. Notice how, without specifying anything, **`koma-script`** abides by that automatically. This is despite using `\subsubsection{Deeper!}` as opposed to `\subsubsection*`, its explicitly unnumbered counterpart.

Also notice how between `\section{Sectioning}`, which produced Section 2.3, and `\subsubsection{Deeper!}`, there was no subsection. Usually, you would want a clean cascade with no jump in the hierarchy levels. If you catch yourself jumping, it is also a sign there might be something off with your actual content structure or thought process. That being said, it is absolutely okay to do; just be aware.

LaTeX has many ways where it does not let you do something easily, or where commands look off. *Never fight LaTeX* in that; you will lose, it will win. When something is awkward to do or requires a lot of manual labor, you are probably doing it wrong and there will be an easier way. Most often, that comes in the form of packages. Being a fully-fleshed programming language under the hood, *any* repetition in LaTeX should make you stop for a second and wonder about another way. That may lead to you discovering suboptimal approaches and code. Just as often, it will not lead anywhere and trying to force minimum code and maximum optimization will not be economical. After all, LaTeX is a markup language.

Paragraphs They are a nice touch, introducing a new, distinct paragraph, idea or thought without being too loud about it.

That being said, leave a blank line in the source code for regular paragraphs. Use either vertical spacing between paragraphs or indentation (like here). Do not use both! Each new train of thought should go into its own paragraph. This paragraph is not indented, as was intended by manually calling `\noindent`. In regular text and documents, there are very few reasons that should ever occur. Again, let LaTeX do its thing; if you find yourself typing `\noindent` all the time, there will be something wrong. For example, paragraph styles (vertical spacing between them, indentation *etc.*) is configured globally.

2.4 References

This section is about referencing. Notice `\lcnamecref` in the previous sentence: it references (in lower-case) the name of the passed label. If this section ever changes to something else (chapter, ...), it is updated automatically.

Note that using package **cleveref**, we only ever issue `\cref{<label>}` commands. That command also has many useful cousins, like `\crefname{<label>}`. The package does the heavy lifting and inserts the reference *type*, also with correct plural forms if required: Table 2.1a and Figures 2.1 and 3.4 ← all of that was done automagically. Doing it any other way is just way too laborious and error-prone.

For added convenience, insert the *type* directly into the label, *e.g.* `\label{fig:hello}`. This aids auto-completion and readability.

2.4.1 Bibliography

The second place where references occur are bibliographical contexts. Examples are spread throughout this document. At their basis, they rely on **biblatex** and its back-end **biber**. Forget about the outdated **bibtex**: that package urges users to use **biblatex** instead, especially for UTF-8 support, which is crucial for citing authors with arbitrary, international names.

Using `\autocite{bibid}`, sources can be cited reliably, while having a whole range of features delivered for free. We do not have to worry about the specific citation styles (in parentheses, as a superscript, ...) — `\autocite{bibid}` takes care of that, we can then manage its behavior globally.

As such, we can have citations looking like: Einstein 1905, p. 8; Goossens, Mittelbach, and Samarin 1993, pp. 29 sqq.; Donald Ervin Knuth 1986, pp. 2–9; Donald E. Knuth 1973, pp. 89 sq.; Dirac 1981 . A note may be added to each citation; if this note is an integer number, it is automatically taken to be the page number. No need to write "*p.*" and similar manually. If a following page is to be included in the citation, append `\psq`. Otherwise, `\psqq` for "this page and the following ones". Using `\autocites{}` *etc.*, we can then chain together as many as we want.

Dirac (1981, p. 3) does not claim anything, this is just an example for a `\textcite`, used to implement a citation to be a readable part of the sentence. There is also a plural form available. Explore the documentation for a taste of the vast selection of automatic citation commands.

Language support All of this is incredibly convenient, for there is minimal manual work and a lot of abstraction into high-level commands. Importantly, this is language-agnostic, and **`biblatex`** will make use of **`polyglossia`** (the LuaLaTeX replacement for **`babel`**) and **`csquotes`**. Therefore, through changing the desired document language for just these packages, all others including **`biblatex`** will quickly adjust automatically.

backref-feature Taking a look at the bibliography (Page 77) in the backmatter of this document reveals the `backref` feature: the pages where a reference was cited occur after its entry. This is helpful in print, but amazing in digital format, for these page numbers are also links. This allows readers to very swiftly navigate and jump within the document.

Try it out by following this reference: Dirac 1981, leading you to the bibliography. It will have this page's number (Page 26) at the end of its entry. Clicking it will land you back here exactly.

Citation Manager To generate the `*.bib` file from which LaTeX pulls the info in the first place, a citation manager is a must-have. Zotero (especially with its excellent *Better Bibtex* addon) is recommendable, but it ultimately does not matter much. Just use *some* manager to keep your actual documents (consisting of bibliographical info and the attachments themselves, like e-books) and the automatically derived `*.bib` files in one place, named and structured consistently.

2.5 Lists

In technical publications, using lists (either bullet points or enumerations) is highly encouraged. They should almost always be preferred over doing the same thing in a block of text. Just consider this example list:

- The demonstrated approach is more complex than the previous one:
    1. more time was spent doing computations,
    2. less was spent fooling around,
    3. features were added.
- At the same time, the following simplifications were made:
    1. went from continuous- to discrete-time simulations,
    2. threw out some superfluous stuff.

The very same information in form of a text block is suddenly much more inaccessible to readers. In technical writing, precision and conciseness matter — prose, synonyms and filler words do not.

[redacted]

Figure 2.1 / Example for a censoring box.

Note the block-like `itemize` symbol (■)[5] and the fact that enumerate numbers are part of the sans-serif family and **bold**. That is totally revolutionary and stuff, because it's... different? If it is not to your liking, lists may be changed, configured and created using the **`enumitem`** package. For example, using that package, this document has a list for numbered descriptions, in essence a combination of the `enumerate` and `description` environments:

1. This item has this description.
2. This other item has a different description.

2.6 Censoring

There is a censoring feature, provided by **`censor`**. It allows you to publish documents containing sensitive information, *e.g.* for proofreading. So, this is now a huge secret: [redacted].

I am a TODO note.

Float contents can also be censored, as illustrated in Figure 2.1.

2.7 Glossaries

`glossaries-extra` is an absolute unit of a package. For this document, it is used with its back-end **`bib2gls`**. This Java tool comes with installations of TeXLive and MiKTeX. You should already have it. This section is only a brief, opinionated overview. For a more appropriate introduction, there is a suitable Beginner's Guide, written by the package author. Before you jump further into the relevant, proper documentations (these are

[5]This is also a regular Unicode character. When copy-pasting old or outdated LaTeX documents, ever noticed all the funny business going on? A good example is copying German *umlauts* like Ä, Ö, Ü. They might turn into ¨ *A*, since LaTeX only ever put two random dots over the regular ASCII letter A. With **`unicode-math`** (building onto **`fontspec`**) and LuaLaTeX, this document is fully Unicode-compatible. Copy-paste anything correctly: $\delta\sigma \int_1^2 y$. Googling the ■ block will reveal that it is `U+25AA`.

three: the **`glossaries`** base package, the comprehensive extension **`glossaries-extra`** and the helper tool **`bib2gls`**), you might want to give the beginner's guide a shot.

In a format similar to normal bibliographies, all glossary entries are now managed using `*.bib` files. Each entry is processed by **`bib2gls`** and put into an auxiliary file. From it, **`glossaries-extra`** reads and inserts all contents when `\gls` and its many sibling commands are used. The framework for all of that, also printing of the glossary and nomenclature, is already taken care of for this document. In addition to printing the glossaries, they are also added to the table of contents. This behavior can be toggled using the `toc` package option flag, see the source code for more info.

Traditionally, glossary packages are used for abbreviations. However, using **`glossaries-extra`** this can be kicked into top gear. It is used for:

- abbreviations,
- (physical) constants,
- symbols (greek, roman and all other),
- subscripts,
- the book index, consisting of names and the normal index.

In the case of symbols, this means the source now relies on `\sym{<label>}` commands, see also Table 2.4. For example, equations no longer read `E = mc^{2}`, but instead

```
\sym{energy} = \sym{mass}\sym{velocity}^{2}
```

This initially unintuitive approach has several critical advantages:

1. absolute **consistency** following the Single Source of Truth (SSOT) principle: there is exactly *one* place where the symbol itself is defined. All other usages are just *references* to this central definition. If suddenly, the symbol for velocity has to change from c to v throughout the entire document, it can be done with ease. Replacing single letters like that using tools like `sed` would be a nightmare, if not impossible.
   There is also absolutely no danger that *both* symbols occur (unless it is explicitly set up like that), with both referring to velocity, as can happen when returning to a document after abandoning it for many months or if multiple authors work on one document.
2. following the **What You See Is What You Mean (WYSIWYM)** principle: this is LaTeX's big strength. In the source code, it no longer states what you *want to see*, but instead *what you mean*. When you write `E = mc^2`, you are writing what you want to see: the letter `E` is somehow equal to the product of letter `m` times `c` squared. But the *meaning* is that *energy* equals *mass* times *velocity* squared. This can now be expressed directly in the source code.
   The WYSIWYM principle is at the core of LaTeX and is what sets it apart. It is the reason you also do not say

```
{\Huge\textsf{\textbf{<Section Title>}}}
```

but instead simply

```
\section{<Section Title>}
```

In the first, it was stated what the author wants to *see*, but only in the second one does it say what was *meant*. It is painfully obvious why the latter approach is the correct one. Another example is emphasized text. Do *emphasized* (produced from `\emph`) and *emphasized* (`\textit`) text look the same? They certainly do... usually.
Yet, what is *meant* is *emphasis*; italic text is just what it happens to look like now, but it is not the *meaning*. For example, we could later decide to redefine emphasized text to bold, or colored. If you previously did not differentiate strictly enough between `\emph` and `\textit`, you are in for a bad time. This is a trap beginners unfortunately often fall into. Using **`glossaries-extra`**, abstracted markup-commands can be taken to a whole next level, leveraging this core LaTeX strength.

3. **abolishing ambiguity**: especially when the source code is read by other people, or even worked on, "naked" symbols can become ambiguous. What American authors refer to with a capital P, European authors interpret as *power*, when really *pressure* was meant. This is a non-issue if in the source it says `\sym{pressure}`. For internationalization, authors would then only adjust their *style-sheets* (in this case the `*.bib` files) and be done.
4. arguably improving **readability**. The source code can almost be read like a normal sentence consisting of full words, albeit with braces and backslashes in the way.
5. printing the **nomenclature** becomes trivial. A single command simply prints the `*.bib` file contents. Being an external Java tool, the customization, sorting and filtering capabilities of **`bib2gls`** for printing those lists are likely more than will ever be needed.
6. lastly, some more gimmicky features are enabled, like printing all page numbers of occurrences of a symbol. Alternatively, only the first occurrence can be printed.

Available Commands **`glossaries-extra`** is already heavily leveraged for this document. Take a look around the source code for all the details. For starters, there are a couple of predefined `*.bib` files, showcased in Table 2.4. Note that using entries with commands like `\sym{<entry name>}` is independent of the entry type used in the `*.bib` file. For example, subscripts are invoked using `\sub{<entry name>}` despite being defined as `@symbol{<entry name> ...` in `subscripts.bib`.

Table 2.4 / Predefined glossaries with their respective `*.bib` files, invoking commands and list occurrence in the document.

Name (`.bib`)	Command	Listed in / Description
`terms`	`\idx`	Index → Terms
`names`	`\name`	Index → Names
`roman`	`\sym`	Glossary → Symbols → Roman
`greek`	`\sym`	Glossary → Symbols → Greek
`other`	`\sym`	Glossary → Symbols → Other
`subscripts`	`\sub`	Glossary → Subscripts
`constants`[a]	`\cons`	Glossary → Numbers
`abbreviations`	`\abb`	Glossary → Acronyms

[a] Constants like π are toy examples for this document. However, this glossary section is very convenient to share assumptions and used constants in a clear and concise way in one central place, aiding reproducibility and overall document integrity.

Modifying Print Output To use glossary entries but not print them in a glossary, comment out the entire relevant `\printunsrtglossary` macro where appropriate. For the index, the relevant command is `\printunsrtindex`. Some glossary categories, like *Symbols*, have sub-categories, like *Roman*, *Greek* and *Other*. To omit printing a single sub-category in the glossaries (but still allow their use in the document[6]), refer to the `notprinted` type and the instructions found in the `*.cls` file.

2.7.1 bib2gls

`bib2gls` is an external tool (of the same name as the package) that **`glossaries-extra`** employs to convert external `*.bib` files with definitions for, for example, acronyms, into a TEX-compatible format. During conversion, it also processes all the entries, like sorting them in whatever way you request. The sorting and filtering capabilities are very strong, and of course also Unicode compatible. A minimal `bib` file for acronyms can be as simple as:

```
@abbreviation{cont_int,
    short={CI},
    long={Continuous Integration},
}
```

For concrete examples, see the files for this document at `bib/glossaries/`. There can be as many keys as required, with custom ones being easily created. Effectively, this

[6] The *Other* symbol entries are required for the provided built-in math macros, see Table 2.2, to work!

is analogous to how bibliography entries are created. Thus, users of LaTeX who are already familiar with that concept and format should have an easy time getting started with **glossaries-extra**. Using it for mathematical symbols can look like:

```
@symbol{abs_temperature,
    name={\ensuremath{T}},
    description={absolute temperature},
    group={roman},
    unit={\unit{\kelvin}},
}
```

2.8 Landscape

Pages in landscape format are rather straightforward to implement. Note that not only are these in landscape orientation; they are also recognized as such by supporting document viewers, rotating the page for you and keeping it legible.

Chapter 3

Float Features

Examples for floats were already shown in Figure 2.1 and Table 2.1a. These are mainly handled by the packages:

caption The base package providing customizations for the caption text, for example automatic ending periods, which can be toggled globally.

svg Including Scalable Vectors Graphics (SVG) files using LaTeX, or in this case LuaLaTeX, is not very straightforward. In fact, it is anything but: SVG files cannot be included directly at all and intermediate steps are needed.
Using the **svg** package, the workflow is somewhat automated. Only the original SVG files are kept as the SSOT, and the generation of the PDF and accompanying `.pdf_tex` files are left to the package. It calls InkScape for converting the SVG to PDF (or another format of choice), and if the SVG contains text to be included as LaTeX, a sidecar `.pdf_tex` file is generated (the default behavior). To call InkScape, two requirements have to be met:

1. elevation aka `--shell-escape` is required to write external files, and
2. InkScape has to be on the `$PATH`, that means installed and subsequently registered. This is automatically done for Linux. If it is not done automatically on Windows, navigate to *Edit the system environment variables* and add the directory containing `inkscape.exe` to the variable.

Once the files are generated, they can be treated as temporary junk and are always easily regenerated from the source SVG.
After years of experimentation, this seems like the best workflow. The only laborious manual task left is placement of annotations onto the generated PDF files. This seems like the best deal: no text is left in the SVG files themselves. Placing and debugging text in SVG files using the InkScape → `PDF+PDF_TEX` route is *very, very* annoying. This is because while InkScape offers text alignment operations (left, center, right) that translate into the embedded `PDF_TEX`, the font cannot be known a priori while working on the SVG. Neither font size (most importantly its height), nor any other font property can be assumed. This also

makes functions like *Resize page to drawing or selection* futile if text is part of the outer elements of a drawing.
Wanting to change any text later on results in having to start InkScape instead of just doing it conveniently in the LaTeX source. The alternative is to place macros (`\newcommand*{}`) everywhere inside the original SVG, where content should later be placed. These macros serve as labels, but are ugly, annoying, and remove the usability of the plain, original SVG file (since we would first need to know what each macro stands for).
Using the **`svg`** package to generate plain, text-less PDF and only later adding any text or annotation in the LaTeX source itself seems the best of both worlds. **It certainly allows both tools to do what they're good at, and no more**: draw free-flowing vectors graphics with InkScape, then add text in LaTeX (which can be done in `\foreach` loops as well). An example for annotations (with loops) is shown in Figure 3.4. All other graphics that are neither bitmaps nor Ti*k*Z graphics are handled using the **`svg`** approach.
All of this is very convenient indeed, since we can now do everything in LaTeX and **`tikz`** and basically never have to revisit the base SVG file, unless the graphic itself changes. All the labels stay in the LaTeX source and are therefore also manageable through `git`.

`floatrow` A notable feature is the capability for captions the same width as the float they are attached to. This tends to look much tighter and tidier, see Figures 3.1a and 3.1b for a comparison.
Other features are automatic centering of floats, automatic positioning of captions (tables on top, figures below, independent of where the `\caption` command occurs in the source), multiple floats and captions on the side. Depending on the aspect ratio of the image, positioning the caption besides the figure can look (subjectively) pretty, see Figure 3.3.

Multiple floats

Other possibilities are rather arbitrary arrangements of sub-figures and -captions. For this, see Figure 3.2, which contains the two sub-figures Figures 3.2a and 3.2b. Multiple sub-tables are also possible, see Table 3.1 with its four sub-tables Tables 3.1a to 3.1d.

Side-Captions

Occasionally, figures and their captions can look disproportionate in combination. In these cases, placing a side-caption might relieve the situation, as shown in Figure 3.3.

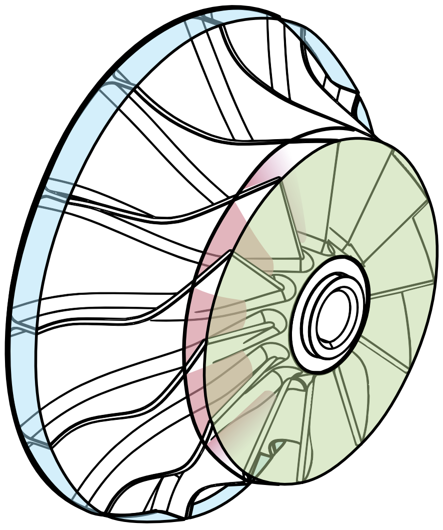


Figure 3.1a / Example for a regular caption, spanning the whole width since it is so long. This can quickly look strange, especially if the figure itself is narrow.

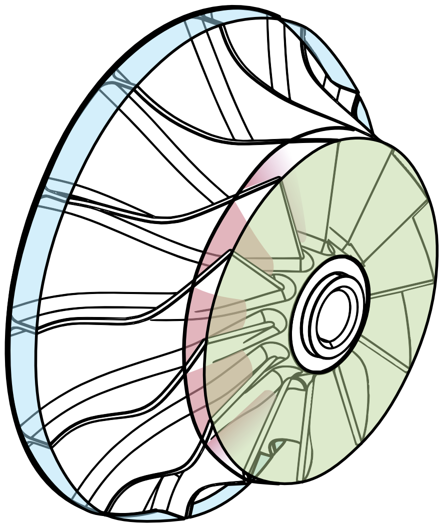
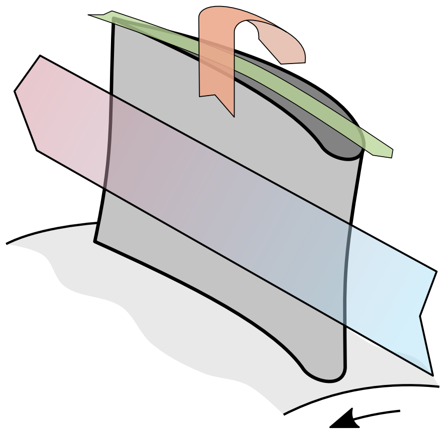
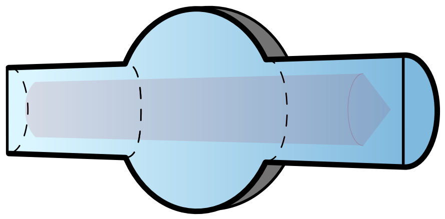


Figure 3.1b / Example for a new caption, spanning just the width of the float it is attached to, despite the caption being much longer than the float is wide.



a / An example.



b / Another example.

Figure 3.2 / An example for sub-figures.

This field can be used for a reference to a source: Adapted from wherever.

Table 3.1 / Example for multiple tables in one float.

a / Table One.

Column One	Column Two
Just	a
normal	table
Nothing	special.

b / Table Two.

Column One	Column Two
Just	a
normal	table
Nothing	special.

c / Table Three.

Column One	Column Two
Just	a
normal	table
Nothing	special.

d / Table Four.

Column One	Column Two
Just	a
normal	table
Nothing	special.

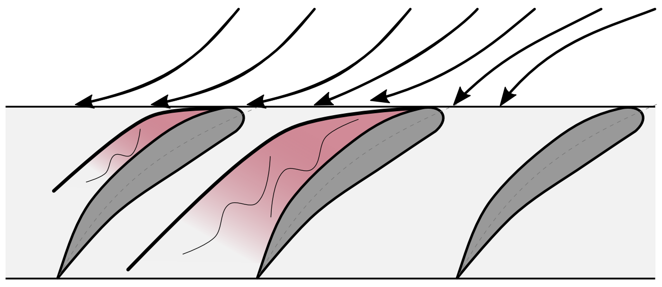


Figure 3.3 / A side caption, which may also span multiple lines like demonstrated in this rather long caption right here.

Large Floats

An example for a larger table is shown in Table 3.2. One key aspect there: the `S` column-type, provided by **`siunitx`**. It automatically applies `\num` to each cell, which in turn allows easy printing of things like: -3.23×10^{-5}. Further, decimal places are accounted for and aligned by automatically. Use `*x{y}` to print column-type `y` (*i.e.* `c`) `x` times. No need for tedious repetition.

Table 3.2 / Compositions and properties of fuels, as used in Eq. 3.1.

Name	Mass fraction γ [–]							ρ [kg/m^3]	H_i [MJ/kg]
	C	H	S	O	N	H_2O	Ash		
Diesel[a]	0.8600	0.1320	0.0060	0.0020[b]		n.a.	n.a.	840	42.7
Oil EL[a]	0.8570	0.1310	0.0100	0.0020[b]		n.a.	n.a.	840	42.7
Oil H[a]	0.8490	0.1060	0.0350	0.0100[b]		n.a.	n.a.	980	40.0
MDO[c]	n.a.	n.a.	0.0150	n.a.	n.a.	0.0030[d]	0.0001	900	n.a.
HFO[e]	n.a.	n.a.	0.0350[f]	n.a.	n.a.	0.0050[d]	0.0015	1010	n.a.
Light[g]	0.8600	0.1320	0.0060	0.0010	0.0010	0	0	840	Eq. 2.12
Medium[g]	0.8530	0.1269	0.0150	0.0010	0.0010	0.0030	0.0001	900	Eq. 2.12
Heavy[g]	0.8460	0.1025	0.0350	0.0050	0.0050	0.0050	0.0015	1010	Eq. 2.12

[a]Baehr and Kabelac 2016, p. 634; see also Dubbel, Grote, and Feldhusen 2007, p. L70 and Mollenhauer and Tschöke 2007, p. 97.
[b]Given as a sum $\gamma_\mathrm{O} + \gamma_\mathrm{N}$.
[c]Max. values of specification *DMB*, International Organization for Standardization 2017.
[d]Given as a volume fraction, assumed equal to mass fraction.
[e]Max. values of specification *RMK*, International Organization for Standardization 2017.
[f]IMO level prior to 2020.
[g]Derived, *virtual* fuels.

Table 3.3 / Dreadful version of Table 2.1a.

Feature	*Sample Text*
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
Bold	**The quick brown Fox jumps over the lazy Dog 13 times!**
Italics	*The quick brown Fox jumps over the lazy Dog 13 times!*
Bold Italics	***The quick brown Fox jumps over the lazy Dog 13 times!***
Small Capitals	The quick brown Fox jumps over the lazy Dog 13 times!
Bold SC	**The quick brown Fox jumps over the lazy Dog 13 times!**
Italics SC	*The quick brown Fox jumps over the lazy Dog 13 times!*

Very long tables See Table A.1 in Appendix A.2 for an example of a very large table (which does not float).

Table Style

In general, use the least ink possible to get your point across. Any more is only noise. This is especially true for tables. For this, compare Table 2.1a to its not-so-blessed twin, Table 3.3:

- do not use vertical lines in tables,
- avoid double lines,
- if in doubt, left-align. If there is no actual reason to center or right-align, refrain from it. Of course, if your language flows right-to-left, like Arabic or Hebrew, this advice is reversed.

For more info, see the package `booktabs`.

Caption Positioning

Note that table captions (see for example Table 3.1) occur *above* the table no matter the `\caption` command's position in the source code. Per convention, figure captions should appear below, table captions above their bodies. This is handled automatically by `floatrow`. Also note that there is neither a period nor *any* character (no space, no empty line) behind the last caption line in the source code, since those periods are managed globally by the `caption` package. They can therefore be toggled easily.

Float Footer There is also a `\floatfoot` command for all floats. This is used to place additional info underneath the caption, for example for references, *cf.* Figure 3.2.

3.1 TikZ and pgfplots

Packages **tikz** and **pgfplots** offer a vast array of features. A select few are presented here.

3.1.1 Drawing over Bitmaps

When having to rely on bitmaps, one might still want to add additional info. This can be done directly in LaTeX, profiting from all the usual features. In the example here, this is of course the retaining of the text font, but also employing the useful **contour** package to draw legible black-on-white (or vice-versa) text. An example is shown in Figure 3.4. There is a debugging grid functionality to allow for easier positioning of the labels on the graphic. Figure 3.4 also showcases a vertical alignment of sub-figures.

3.1.2 Trees

Drawing trees is possible with **forest**, see Figure 3.5. It is a very easy package to get started with. Take a look at the source code. It is deceivingly simple, but also highly customizable. Note that the package is stand-alone, but uses Ti*k*Z under the hood (like a lot of packages do).

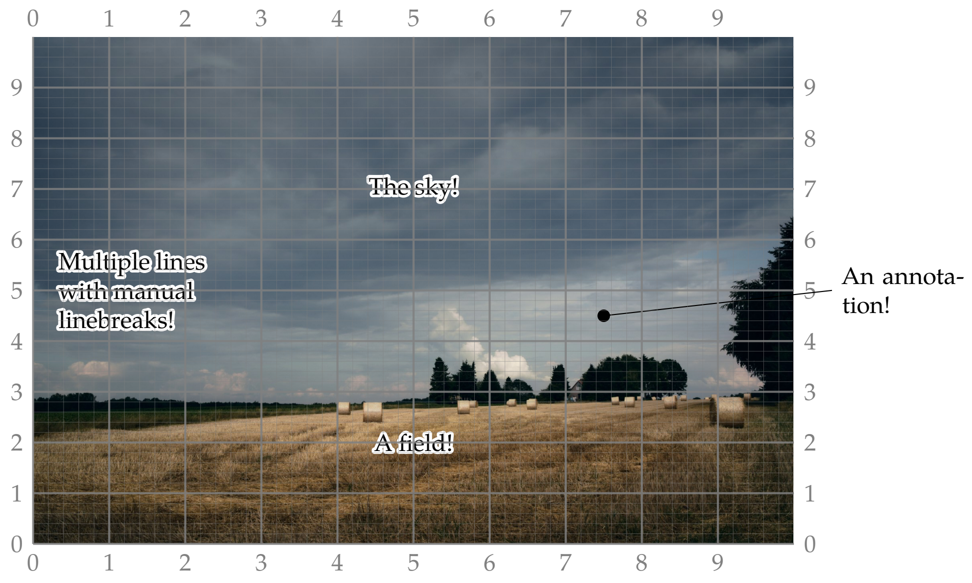
3.1.3 Plotting

If you rely on tools like `matlab2tikz`, this is for you. Instead of these outside tools, one can plot *directly* in LaTeX, either through importing external `*.csv` data (for example from experiments) or through computing the plots using **pgfplots** directly from within LaTeX.

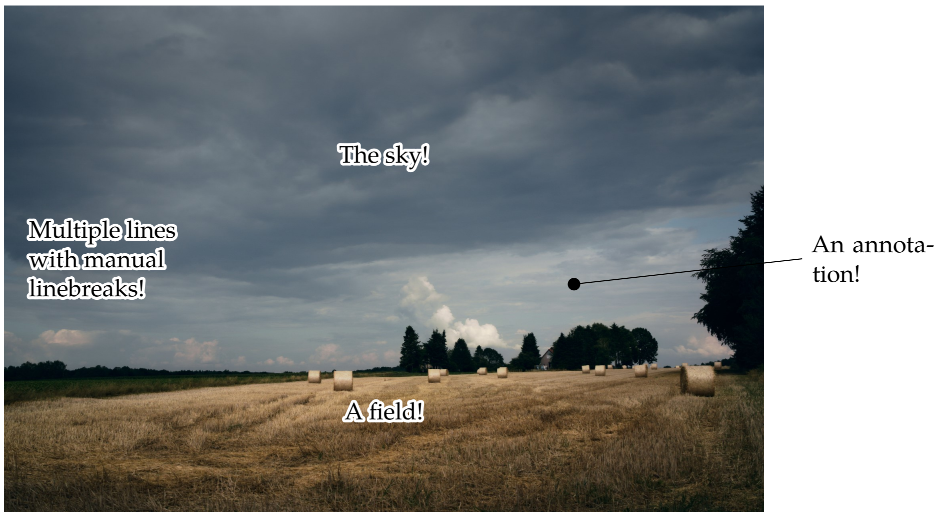
Directly in LaTeX

Plotting and computing the return values directly in LaTeX is achieved through the `declare function` command. While the functionality is limited (owed to the limitations of TeX, which is of course not meant for this sort of thing), it may still save a lot of time and headaches. Finding and specifying the correct settings for **pgfplots** can take a lot of time. This is already taken care of for this document. As such, a plot from existing `*.csv` data can be set up in a handful of lines using one of the built-in styles, like `regularplot`. Plotting from a Ti*k*Z function is demonstrated in Figures 3.6a and 3.6b.

Contour Plots Contour plots are too much to handle for **pgfplots** itself. This is owed to the limitations of the underlying TeX engine. What the math and computation engine of **pgfplots** does is amazing, but in the end, TeX is a typesetting system, not a general-purpose programming language. However, **pgfplots** has an option to delegate



a / Debugging/positioning grid.



b / Result.

Figure 3.4 / Drawing over a bitmap graphic using Ti*k*Z in a vertical sub-figure environment.

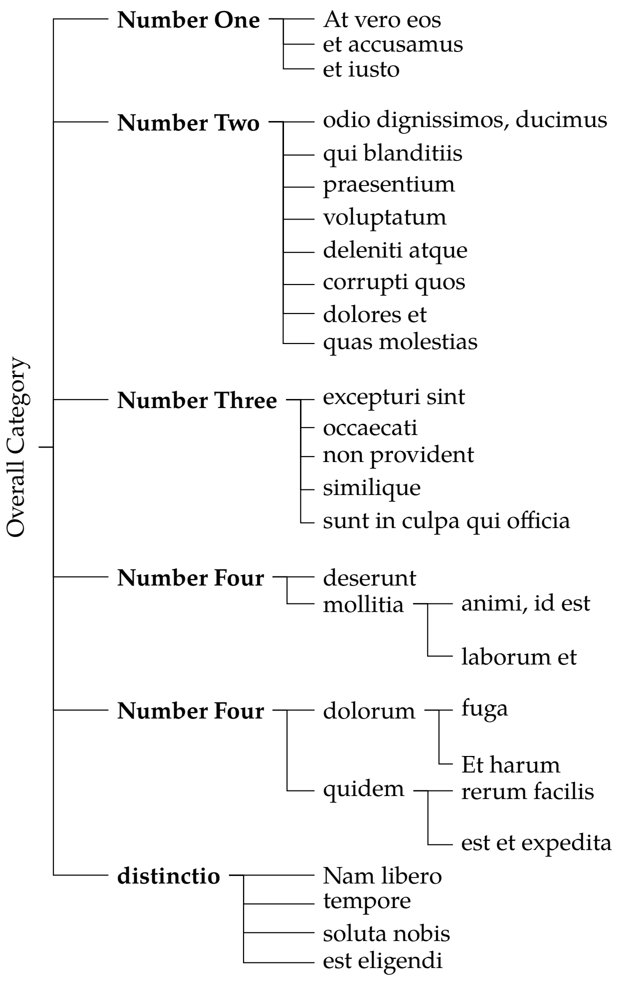


Figure 3.5 / Example for a tree.

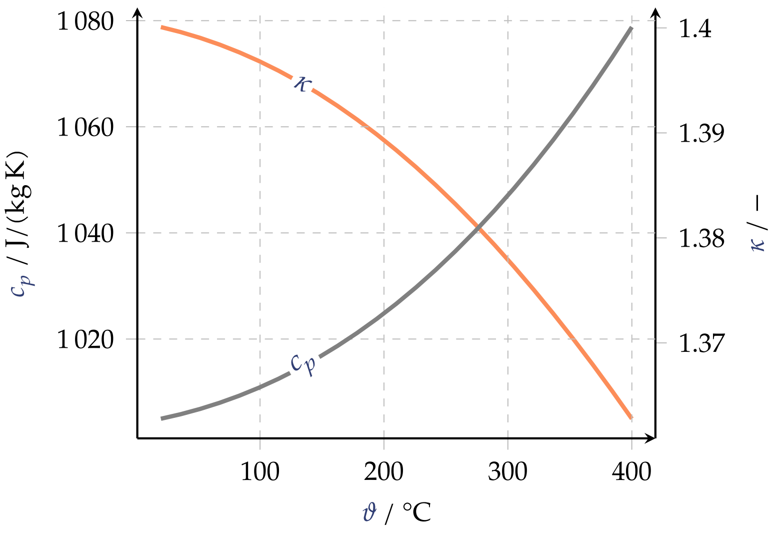


Figure 3.6a / Caloric parameters of air. **Avoid legends** and put info where it belongs, improving legibility (less back-and-forth for the eye). This plot is using the custom `regularplot` style.

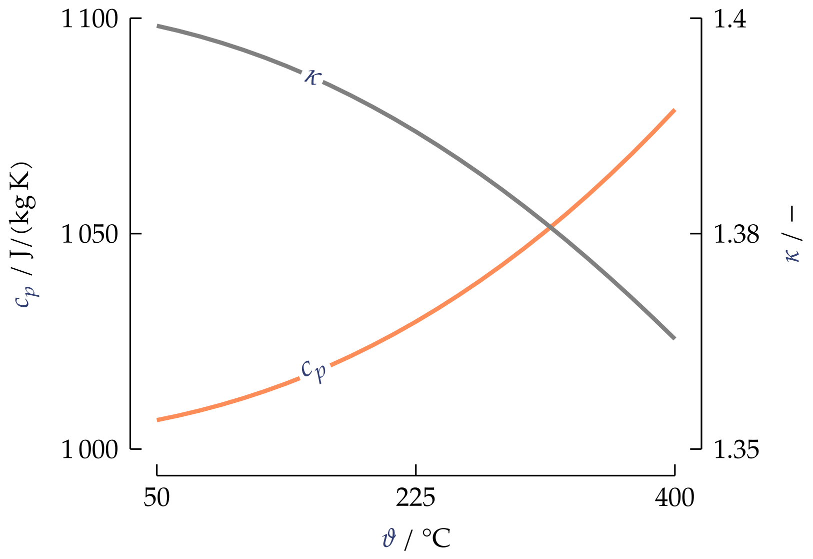


Figure 3.6b / Same content as Figure 3.6a in "*Tufte*-like" for a modern, minimalist look where precision counts less than the overall message. For more info, refer to its namesake, TUFTE.

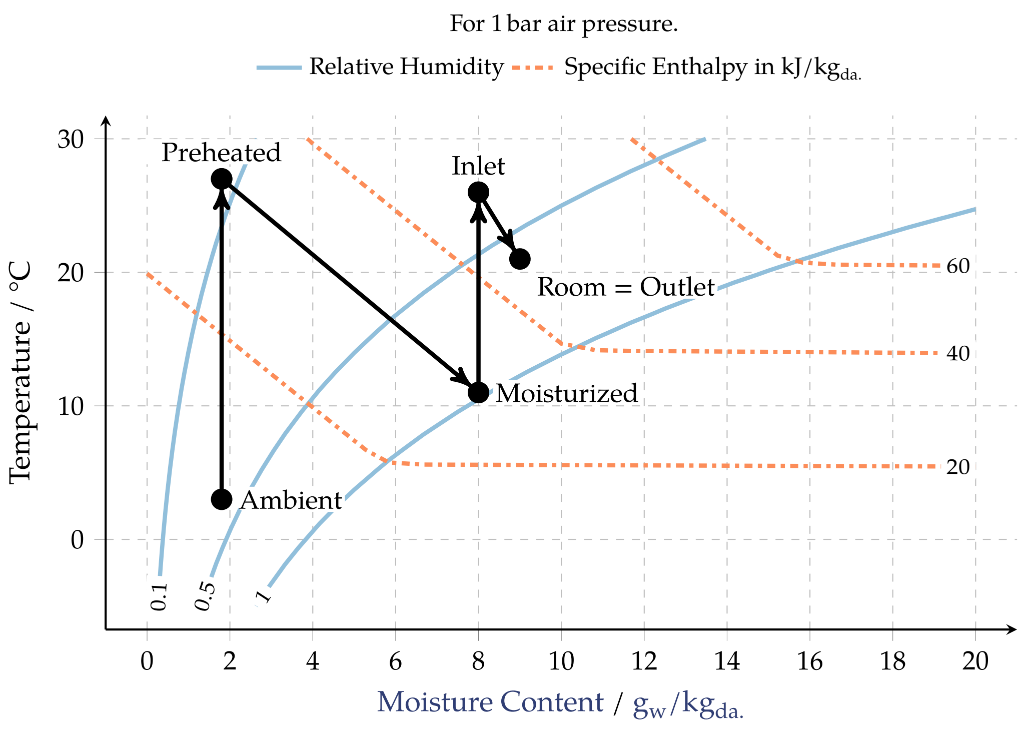


Figure 3.7 / MOLLIER diagram for humid air as an example for `gnuplot`. The entire source code required (functions *etc.*) is in the LaTeX source.

computations to the external `gnuplot` tool. In analogy to **`svg`**, using it requires `gnuplot` itself to be installed and on the `$PATH`. Figure 3.7 shows an example for a plot entirely specified in the LaTeX source. All the functions required are defined using **`pgfplots`** functionalities! "Only" the computation itself is outsourced to `gnuplot`. The resulting plot is two-dimensional, but the underlying logic is three-dimensional, where one dimension was reduced, forming contours. This is where the complexity resides. The entire plot is just 200 lines of code, with really only half that being lines that effectively do something useful.

Time-series Time-series plots are straightforward to implement. **`pgfplots`**, using its `dateplot` library, can automatically parse and plot dates. For this to work best and most reliably, dates and times should be in ISO 8601 format:

```
YYYY-MM-DDThh:mm:ss :
2020-05-19T12:15:31Z,
```

where `Z` is time-zone information and `T` is an arbitrary, but fixed separator. It can also be a simple space. Use just the date *or* time part when appropriate.

This format is unambiguous and understood worldwide as well as, most importantly here, by computers. No extra string and date parsing is required, it will just work in

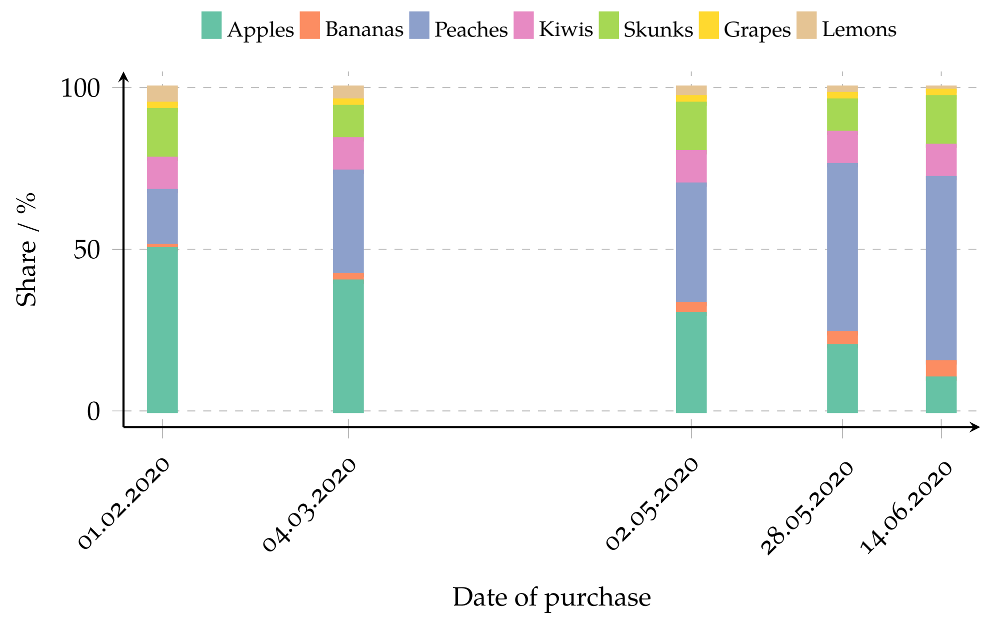


Figure 3.8 / Example automatic timeseries plot. Note the automatic spacing-out according to the actual time deltas, and the automatic conversion of timestamps to human-friendly versions, to whatever specification the author chooses. Also note the colors: here, *distinction* is important and a *qualitative* palette is chosen. A *sequential* or a *diverging* palette would have been less suited (some would say plain wrong).

a lot of cases, not only for `pgfplots` like here. Further, it is often the standard output format (or close to it) of measurement equipment anyway, so there is not even a need to modify it.

An example is shown in Figure 3.8. Note how the actually displayed date/time format can be modified and made human-readable freely. Only the underlying data format is best followed strictly in ISO 8601 format.

Importing CSV

Often, one wants to plot data from files. The better behaved the file is (meaningful headers, no junk rows), the easier that is. In Figure 3.9, `y=<column header>` is all that has to be specified for the column corresponding to that name to be automatically chosen, with no confusion about indices or column numbers.

Using MATLAB2Ti*k*Z

MATLAB2Ti*k*Z is a tool to convert MATLAB figures to LaTeX, see Figure 3.10a. Notice that by default, it exports using whitespace (tabs) as column separators, which might differ from what is set as a global option in `\pgfplotstableset`.

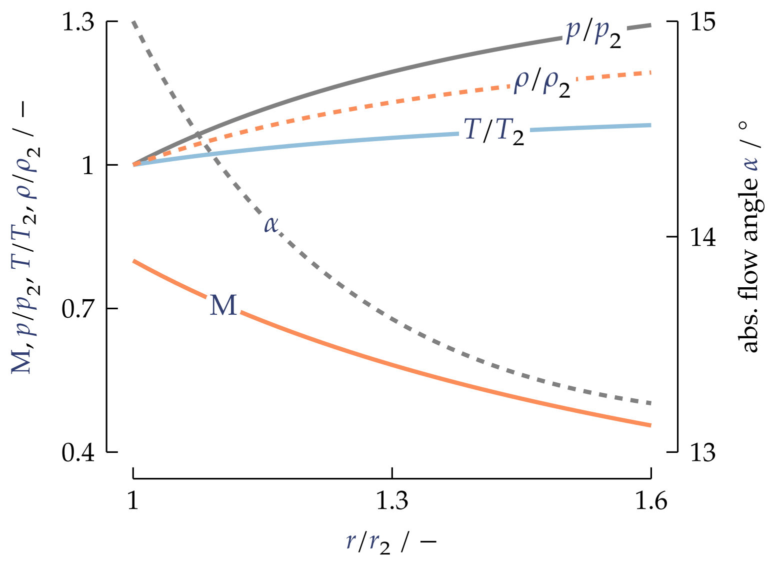


Figure 3.9 / Plotting from CSV data for a diffuser.

Figure 3.10b is the same plot, but generated directly in LaTeX, without relying on MATLAB2TikZ—using the latter, a plethora of things break since it hard-codes a bunch of stuff. Figure 3.10b is only around 30 lines of code (arrived at after only a couple of hours...)!

3.1.4 TikZ and Text

TikZ content can also be intertwined with text using `\tikzmark`. This is illustrated in Equation 3.1. Note that this procedure needs two compilation runs, since the label positions need to be written to an auxiliary file first.

$$\underset{\substack{\uparrow\\\text{Carbon}}}{\gamma_{\mathrm{C}}} + \underset{\substack{\uparrow\\\text{Hydrogen}}}{\gamma_{\mathrm{H}}} + \underset{\substack{\uparrow\\\text{Sulphur}}}{\gamma_{\mathrm{S}}} + \underset{\substack{\uparrow\\\text{Oxygen}}}{\gamma_{\mathrm{O}}} + \underset{\substack{\uparrow\\\text{Nitrogen}}}{\gamma_{\mathrm{N}}} + \underset{\substack{\uparrow\\\text{Water}}}{\gamma_{\mathrm{H_2O}}} + \underset{\substack{\uparrow\\\text{Ash}}}{\gamma_{\mathrm{ash}}} := 1. \tag{3.1}$$

3.1.5 Regular TikZ pictures

TikZ really is not meant for arbitrary graphics. The more free-form images shown in this document were created using InkScape. Still, "drawing" in TikZ is much preferred and easier when the images are somewhat programmatic, aka there are straight corners, edges and turns, equal distances, and everything is a bit "block-like", repetitive. For example, a small file structure diagram:

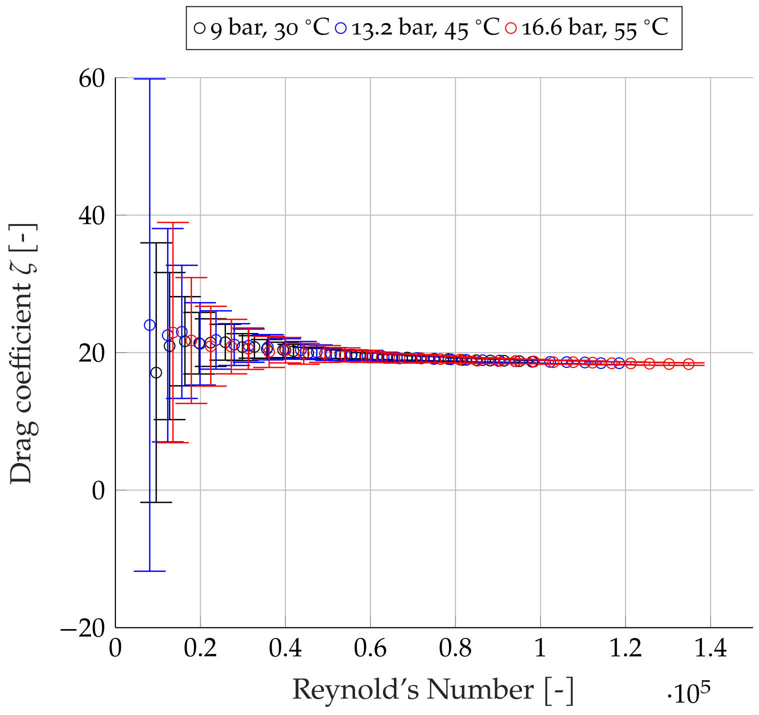


Figure 3.10a / A vanilla MATLAB2Ti*k*Z example, imported here without changes except those to allow successful compilation (see commit `56556a0`: `set table/col sep=space`). While it works, the style created in MATLAB conflicts with the local one, and you miss out on many useful features, like **`siunitx`** or **`glossaries-extra`**. A more frictionless approach is to export plain-text (CSV) data from MATLAB, and import it into LaTeX, see Figures 3.9 and 3.10b.

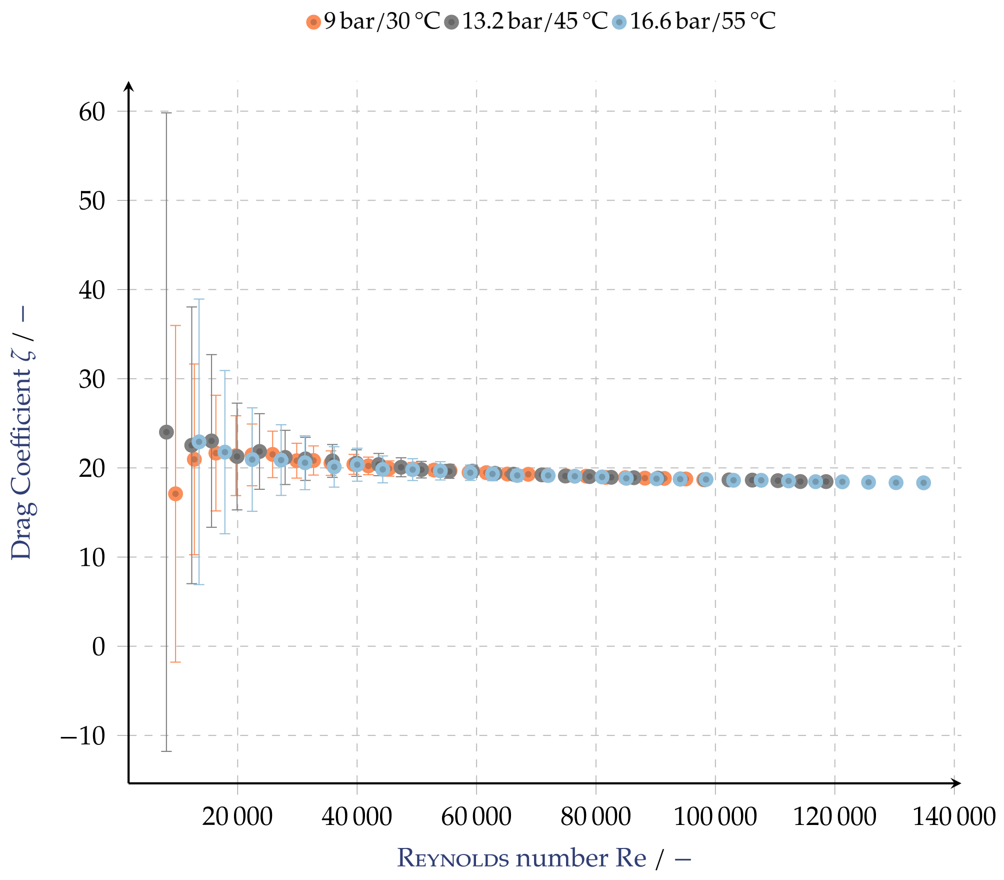


Figure 3.10b / Same data as Figure 3.10a but plotted in LaTeX directly, using only the provided raw data. Note how this allows for a consistent style and unlocks all other, usual features found in this template. Also take a look at the source data to see what *tidy data* looks like, a data format most suitable for data processing.

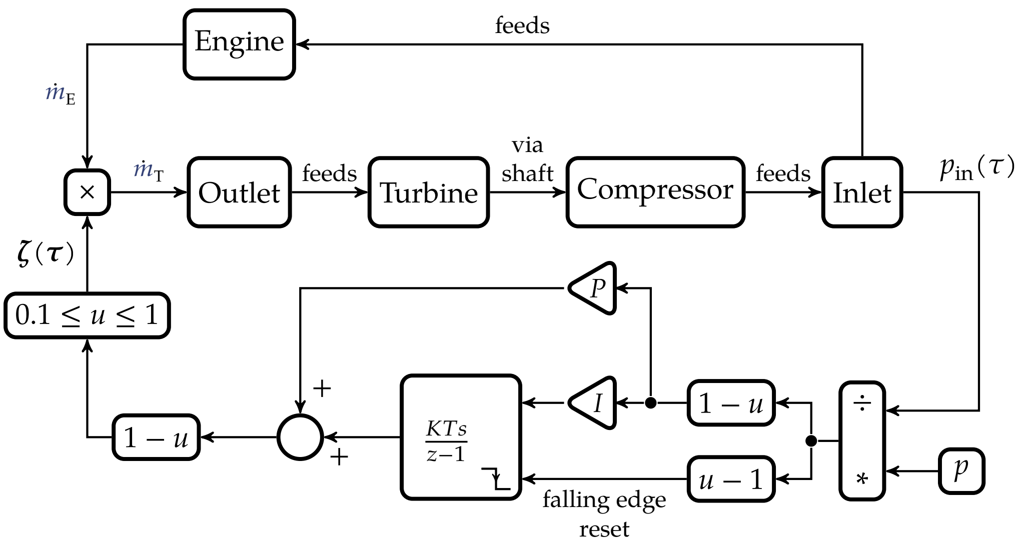


Figure 3.11a / Wastegate implementation in a feedback-loop in MATLAB/Simulink as an example for a TikZ diagram.

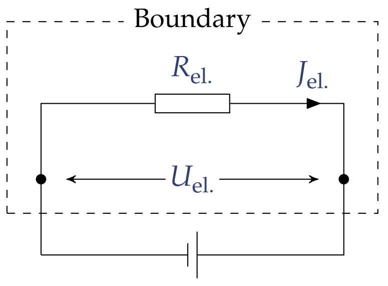
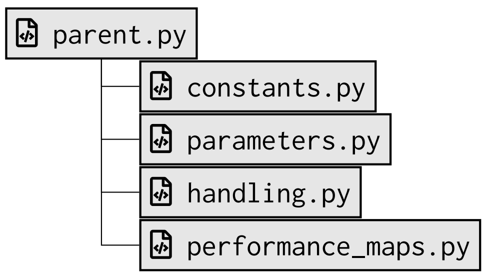


Figure 3.11b / Example for the `circuits.ee.IEC` TikZ library.



Note how `tikzpicture` environments do not have to be contained in floats.

More TikZ examples are shown in Figures 3.11a to 3.11c.

Included shapes This repository includes custom-made shapes for thermodynamic applications. These can be used like many other TikZ elements, for example by positioning them somewhere on the canvas, connecting them to other elements, rotating them *etc*. In that sense, they work like usual TikZ elements (just buggier…). There are a couple of advantages:

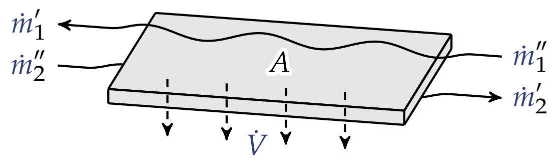


Figure 3.11c / Example for a three-dimensional TikZ drawing using the `3d` library.

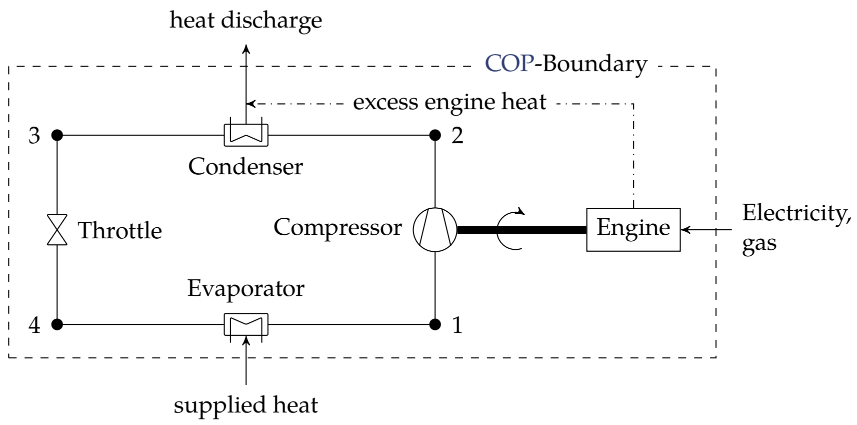


Figure 3.12a / Example for a thermodynamic device drawing using TikZ. It relies heavily on the custom-made library of shapes.

1. unified looks: no more drawing these in InkScape, where they come out slightly dissimilar every time,
2. tight integration with TikZ, allowing to use all its other features,
3. very fast generation of drawings once some familiarity is gained; with InkScape or other outside tools, one can also become fast, but it is hard to beat a LaTeX-internal approach.

Examples are shown in Figures 3.12a and 3.12b. Refer to their source code to see how more or less easily they are created.

3.1.6 InkScape

Having seen what TikZ is good at, Figures 3.1b to 3.3 are good examples for when InkScape might be the better choice: three-dimensional, curvy drawings.

Figure 3.1b is a vectorized bitmap. InkScape can detect edges and contrasts in bitmaps and replicate those lines in vector format (`Path > Trace Bitmap`).

3.2 Example Boxes

As a special gimmick, there is an environment for examples (can also be renamed). It may be useless now, but you can alter it to suit your needs; the skeleton is there for you. It even has its own list, like the list of figures. For an example box, see Example 3.1.

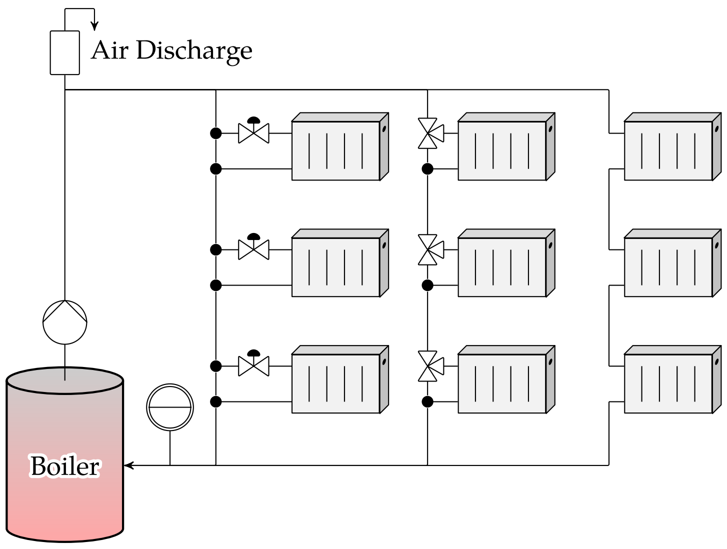


Figure 3.12b / Example Ti*k*Z shapes.

Example 3.1: I am a useless box

There can be pretty much any content in here. Math works — as we can see,

$$1 = 1 \tag{3.2}$$

still holds true after all these years!
If the `float` setting is set, this box also floats. Inserting other floats in here will then cause LaTeX to have a massive fit (floats inside floats do not make sense). This is circumvented setting the `[H]` flag, saying "this float is not really a float, pin it down *right* here".
In any other context, setting any such flag is code smell/poor style. These are often used wrong and just as often set prematurely and then reset countless times, all the while LaTeX complains (rightfully so) about the poor spacing introduced by forcing float positions, instead of letting LaTeX take care of it. For the love of God, let LaTeX do its job in placing the floats. Truth be told, they will on occasion not be placed where you need or want them. Keep working. At the very end, when all is done, go ahead and change the few *truly* misplaced floats manually, by shoving them about in the source code (still not using `htb!` flags). This ensures minimal pain and maximum usage of LaTeX's spacing and placing capabilities.
Anyway — here is such a float within an example box:

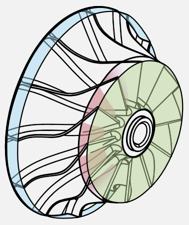


Figure 3.13 / Side-captions are still possible. So are labels.

Note how using `\linewidth` as a length, not the global constant `\textwidth`, figures can be scaled according to the current context.
This box even breaks across pages if so required. Should this turn out ugly, some manual action is certainly required.

Chapter 4

Code Syntax Highlighting

To properly typeset code in LaTeX, we use the `minted` package. It relies on Python, and calls in outside help for syntax highlighting using Python's `pygments` package. As such, it requires `--shell-escape` to compile, and of course Python. The latter can be a pain in the buttocks to set up; Docker usage is especially useful here. Since `pygments` has nothing to do with LaTeX (whose ecosystem is a sad mess), but is instead a regular old Python package, chances are the language of your choice is not only available but also well-supported![1]

Color Scheme Note how the color scheme is the same throughout languages. The idea is that keywords of similar importance, status or semantics are highlighted uniformly. For example, keywords for class and function definitions, like `class` for Python or `classdef` for MATLAB.[2] Another example are error-handling and -throwing keywords, which many languages offer. All these different types should be identified and treated equally. `minted` is a widely used package and knows about a lot of languages. You can check it out at

https://pygments.org/demo/.

If the current colors are not to your liking they can be changed easily using `minted`'s `style` option. Refer to the comments in the source code on how to see available styles.

References Individual code lines can also be referenced. For example, we find a `return` statement on line 24 in Code Listing 4.1. That line is also highlighted, using `minted`'s `highlightlines` option.

[1]For example, `listings`, the inferior alternative to `minted`, still had no Python 3 support (only basic 2) in 2020, at the time of originally writing this. At that point, Python 3 was over 10 years old already and Python 2 was end-of-life.

[2]Notice how these keywords were created using an *inline* listing, like: `y = [file_patterns[x] for x in ["send", "help"]]`.

```python
import json
import logging.config
from pathlib import Path

from resources.helpers import path_relative_to_caller_file

LaTeX can go in here: $\sum_{i=1}^{n} a + \frac{\pi}{2}$

def set_up_logging(logger_name: str) -> logging.Logger:
    """Set up a logging configuration."""
    config_filepath = path_relative_to_caller_file("logger.json")  # same directory

    try:
        with open(config_filepath) as config_file:
            config: dict = json.load(config_file)
        logging.config.dictConfig(config)
    except FileNotFoundError:
        logging.basicConfig(
            level=logging.INFO, format="[%(asctime)s: %(levelname)s] %(message)s"
        )
        logging.warning(f"Using fallback: no logging config found at {config_filepath}"
         ↪  )
        logger_name = __name__

    return logging.getLogger(logger_name)
```

Code Listing 4.1 / This is a caption. Listings cannot be overly long since floats do not page-break.

4.1 Python

The following examples are not always complete or functioning, they are only supposed to showcase the available syntax highlighting. The base style looks like:

```python
def get_nonempty_line(
    lines: Iterable[str],
    last: bool = True
) -> str:
    if last:
        lines = reversed(lines)
    return next(line for line in lines if line.rstrip())
```

It is intended for (small) samples of code that flow into the surrounding text. A second feature are code listings as regular floats, as demonstrated in Code Listing 4.1. As floats, they behave like any other figure, table *etc.*

The third example showcases breaking across pages, probably best suited for an appendix:

```python
def ansi_escaped_string( A random reference: Figure 2.1
    string: str,
    *,
    effects: Union[List[str], None] = None,
    foreground_color: Union[str, None] = None,
    background_color: Union[str, None] = None,
    bright_fg: bool = False,
    bright_bg: bool = False,
) -> str:
    """Provides a human-readable interface to escape strings for terminal output.

    Using ANSI escape characters, the appearance of terminal output can be changed. The
    escape chracters are numerical and impossible to remember. Also, they require
    special starting and ending sequences. This function makes accessing that easier.

    Args:
        string: The input string to be escaped and altered.
        effects: The different effects to apply, e.g. underlined.
        foreground_color: The foreground, that is text color.
        background_color: The background color (appears as a colored block).
        bright_fg: Toggle whatever color was given for the foreground to be bright.
        bright_bg: Toggle whatever color was given for the background to be bright.

    Returns:
        A string with requested ANSI escape characters inserted around the input string.
    """

    def pad_sgr_sequence(sgr_sequence: str = "") -> str:
        """Pads an SGR sequence with starting and end parts.

        To 'Select Graphic Rendition' (SGR) to set the appearance of the following
        terminal output, the CSI is called as:
        CSI n m
        So, 'm' is the character ending the sequence. 'n' is a string of parameters, see
        dict below.

        Args:
            sgr_sequence: Sequence of SGR codes to be padded.
        Returns:
            Padded SGR sequence.
        """
        control_sequence_introducer = "\x1B["  # hexadecimal '1B'
        select_graphic_rendition_end = "m"  # Ending character for SGR
        return control_sequence_introducer + sgr_sequence +
         ↪  select_graphic_rendition_end

    # Implement more as required, see
    # https://en.wikipedia.org/wiki/ANSI_escape_code#SGR_parameters.
    sgr_parameters = {
        "underlined": 4,
    }

    sgr_foregrounds = {  # Base hardcoded mapping, all others can be derived
        "black": 30,
        "red": 31,
        "green": 32,
        "yellow": 33,
        "blue": 34, 30 + 4
        "magenta": 35,
        "cyan": 36,
        "white": 37,
    }

    # These offsets convert foreground colors to background or bright color codes, see
    # https://en.wikipedia.org/wiki/ANSI_escape_code#Colors
    bright_offset = 60
    background_offset = 10

    if bright_fg:
        sgr_foregrounds = {
            color: value + bright_offset for color, value in sgr_foregrounds.items()
        }

    if bright_bg:
        background_offset += bright_offset

    sgr_backgrounds = {
        color: value + background_offset for color, value in sgr_foregrounds.items()
    }

    # Chain various parameters, e.g. 'ESC[30;47m' to get white on black, if 30 and 47
    # were requested. Note, no ending semicolon. Collect codes in a list first.
    sgr_sequence_elements: List[int] = []

    if effects is not None:
        for sgr_effect in effects:
            try:
                sgr_sequence_elements.append(sgr_parameters[sgr_effect])
            except KeyError:
                raise NotImplementedError(
                    f"Requested effect '{sgr_effect}' not available."
                )
    if foreground_color is not None:
        try:
            sgr_sequence_elements.append(sgr_foregrounds[foreground_color])
        except KeyError:
            raise NotImplementedError(
                f"Requested foreground color '{foreground_color}' not available."
            )
    if background_color is not None:
        try:
            sgr_sequence_elements.append(sgr_backgrounds[background_color])
        except KeyError:
            raise NotImplementedError(
classdef BasicClass
    properties
        Value {mustBeNumeric}
    end
    methods
        function r = roundOff(obj)
            r = round([obj.Value],2);
        end
        function r = multiplyBy(obj,n)
            r = [obj.Value] * n;
        end
    end
end
```

Code Listing 4.2 / A class definition in MATLAB, from Mathworks 2020.

```
                f"Requested background color '{background_color}' not available."
            )

    # To .join() list, all elements need to be strings
    sgr_sequence: str = ";".join(str(sgr_code) for sgr_code in sgr_sequence_elements)

    reset_all_sgr = pad_sgr_sequence()  # Without parameters: reset all attributes
    sgr_start = pad_sgr_sequence(sgr_sequence)
    return sgr_start + string + reset_all_sgr
```

4.2 MATLAB

This section contains example code for MATLAB, for example:

```
%{
    Universal Gas Constant for SIMULINK.
%}
R_m = Simulink.Parameter;
    R_m.Value = 8.3144598;
    R_m.Description = 'universal gas constant';
    R_m.DocUnits = 'J/(mol*K)';
```

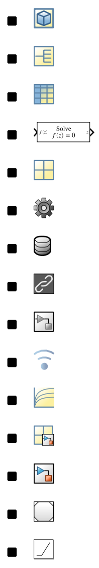
Of course, floats (see Code Listing 4.2) are available as well. So are longer sections, like the following.

```
fullfilepath = mfilename('fullpath');
[filepath, filename, ~] = fileparts(fullfilepath);

%% Begin Dialogue
%{
Extract Data from user-specified input. Can be either a File that is run
(an *.m-file), variables in the Base Workspace or existing data from a
previous run (a *.MAT-file). All variables are expected to be in Table
```

```
format, which is the data type best suited for this work. Therefore, we
force it. No funny business with dynamically named variables or naked
matrices allowed.
%}
pass_data = questdlg({'Would you like to pass existing machine data?', ...
    'Its data would be used to compute your request.', ...
    ['Regardless, note that data is expected to be in ',...
    'Tables named ''', dflt.comp, ''' and ''', dflt.turb, '''.'], '',...
    'If you choose no, existing values are used.'}, ...
    'Machine Data Prompt', btnFF, btnWS, btnNo, btnFF);

switch pass_data
    case btnFF
        prompt = {'File name containing the Tables:', ...
            'Compressor Table name in that file:',...
            'Turbine Table name in that file:'};
        title = 'Machine Data Input';
        dims = [1 50];
        definput = {dflt.filename.data, dflt.comp, dflt.turb};
        machine_data_file = inputdlg(prompt, title, dims, definput);
        if isempty(machine_data_file)
            fprintf('[%s] You left the dialogue and function.\n',...
                datestr(now));
            return
        end
        run(machine_data_file{1});%spills file contents into funcWS
    case btnWS
        prompt = {'Base Workspace Compressor Table name:',...
                'Base Workspace Turbine Table name:'};
        title = 'Machine Data Input';
        dims = [1 50];
        definput = {dflt.comp, dflt.turb};
        machine_data_ws = inputdlg(prompt, title, dims, definput);
        if isempty(machine_data_ws)
            fprintf('[%s] You left the dialogue and function.\n',...
                datestr(now));
            return
        end
    case btnNo
        boxNo = msgbox(['Looking for stats in ''', dflt.filename.stats, ...
            '''.'], 'Using Existing Data', 'help');
        waitfor(boxNo);
        try
            stats = load(dflt.filename.stats);
            stats = stats.stats;
        catch ME
            switch ME.identifier
                case 'MATLAB:load:couldNotReadFile'
                    warning(['File ''', dflt.filename.stats, ''' not ',...
                        'found in search path. Make sure it has been ',...
                        'generated in a previous run or created ',...
                        'manually. Resorting to hard-coded data ',...
                        'for now.']);
                        a_b = 200.89;
                        x_c = 0.0012;
                        f_g = 10.0;
                otherwise
                    rethrow(ME);
            end
        end
    case ''
        fprintf('[%s] You left the dialogue and function.\n',datestr(now));
        return
    otherwise%Only gets here if buttons are misconfigured
        error('This option is not coded, should not get here.');
end
```

4.2.1 MATLAB/Simulink icons

For an older project, MATLAB/Simulink vector icons were created. They are included here at the off-chance that someone else might find a use for these.



4.3 Modelica

Naturally, floating and all other environments and styles are also available for Modelica. The syntax highlighting for a few basic code samples[3] looks like:

```
x := 2 + y;
x + y = 3 * z;
```

[3]Wikipedia Contributors 2021.

```
model FirstOrder
    parameter Real c=1 "Time constant";
    Real x (start=10) "An unknown";
equation
    der(x) = -c*x "A first order differential equation";
end FirstOrder;

type Voltage = Real(quantity="ElectricalPotential", unit="V");
type Current = Real(quantity="ElectricalCurrent", unit="A");

connector Pin "Electrical pin"
    Voltage      v "Potential at the pin";
    flow Current i "Current flowing into the component";
end Pin;

model Capacitor
    parameter Capacitance C;
    Voltage u "Voltage drop between pin_p and pin_n";
    Pin pin_p, pin_n;
equation
    0 = pin_p.i + pin_n.i;
    u = pin_p.v - pin_n.v;
    C * der(u) = pin_p.i;
end Capacitor;

model SignalVoltage
    "Generic voltage source using the input signal as source voltage"
    Interfaces.PositivePin p;
    Interfaces.NegativePin n;
    Modelica.Blocks.Interfaces.RealInput v(unit="V")
        "Voltage between pin p and n (= p.v - n.v) as input signal";
    SI.Current i "Current flowing from pin p to pin n";
equation
    v = p.v - n.v;
    0 = p.i + n.i;
    i = p.i;
end SignalVoltage;

model Circuit
    Capacitor C1(C=1e-4) "A Capacitor instance from the model above";
    Capacitor C2(C=1e-5) "A Capacitor instance from the model above";
        ...
equation
    connect(C1.pin_p, C2.pin_n);
        ...
end Circuit;
```

4.4 Lua

Files can also be read into LaTeX directly. For example, the following is some current Lua code *used for this very document*:

```lua
--[[
    The following import is not required since in LuaTeX's `\directlua` environment,
    `token` etc. is already available. However, this is nice to keep linters from
    complaining about an undefined variable.
--]]
local token = require("token")
local texio = require("texio")
local status = require("status")

--[[
    Trying to incorporate dynamic values into certain newcommand macros. Their
    contents are set at build-time according to environment variables. This is useful
    for automatic workflows in CI environments. See also:
    https://tex.stackexchange.com/a/1739/120853.
    An alternative to using environment variables are command line arguments:
    https://tex.stackexchange.com/a/18813/120853
    However, this seems more error-prone and requires more steps, e.g. piping arguments
    to `lualatex` through `latexmk` first, etc.
    The previous approach was to `sed` for certain `newcommand` definitions in an
    additional CI job. This was much more error-prone (bash scripting) and less
    easily expanded than the below approach.
    LuaTeX provides excellent access to TeX, making this implementation much easier.
--]]

local function get_cmd_stdout(cmd)
    -- See: https://stackoverflow.com/a/326715/11477374
    local fh = assert(io.popen(cmd))
    local first_line = assert(fh:read())
    fh:close()
    return first_line
end

-- Environment variables as used e.g. in GitLab CI.
-- Otherwise, e.g. when developing locally, use commands as a fallback.
local macro_content_sources = {
    GitRefName = {
        env = "CI_COMMIT_REF_NAME",
        cmd = "git rev-parse --abbrev-ref HEAD",
    },
    GitShortSHA = {
        env = "CI_COMMIT_SHORT_SHA",
        cmd = "git rev-parse --short HEAD",
    },
}

for macro_name, content_sources in pairs(macro_content_sources) do
```

```lua
    -- Default: check for environment variable:
    local env = content_sources.env
    local cmd = content_sources.cmd
    local content = "n.a."  -- Default value
    local env_content = os.getenv(env)

    if env_content and env_content ~= "" then  -- Empty string evaluates to true
        texio.write_nl("Found and will be using environment variable '"..env.."'.")
        content = env_content
    else
        texio.write_nl("Environment variable '"..env.."
         ↪  ' undefined or empty, trying fallback command.")
        -- luatex reference for shell escape:
        -- "0 means disabled, 1 means anything is permitted, and 2 is restricted"
        if status.shell_escape == 1 then
            local cmd_success, cmd_stdout = pcall(get_cmd_stdout, cmd)
            if cmd_success then
                texio.write_nl("Fallback command '"..cmd.."' succeeded.")
                content = cmd_stdout
            else
                texio.write_nl("Fallback command '"..cmd.."' unsuccessful.")
            end
        else
            texio.write_nl("shell-escape is disabled, cannot use fallback command.")
        end
    end

    -- Shouldn't happen, would be programmer error, therefore assert Python-style
    assert(content, "Content not defined (neither success nor fallback present)")

    --[[
        The `content` can contain unprintable characters, like underscores in git branch
        names. Towards this end, use detokenize in the macro itself, which will make all
        characters printable (assigns category code 12). See also:
        https://www.overleaf.com/learn/latex/Articles/An_Introduction_to_LuaTeX_(Part_2):_Understanding
    --]]
    local escaped_content = "\\detokenize{"..content.."}"

    texio.write_nl("Providing new macro '"..macro_name.."' with contents: '"..
     ↪  escaped_content.."'.")

     ↪ --  Set a macro (`\newcommand`) see also: https://tex.stackexchange.com/a/450892/120853
    token.set_macro(macro_name, escaped_content)
end
```

Appendix A

An appendix chapter

Some appendix content can go here, for example detailed computations.

A.1 More appendix

The usual sectioning commands keep working, as does page numbering, showcased by the following blind text.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

A.2 Unprocessed data

This is also a good place to put unprocessed data, like Table A.1.

Table A.1: A `longtblr` from the new **`tabularray`** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column

Magnitude	Unit	Variable-width text	Math column
10.0	m	Some rather long text that will get broken up.	$x = y$

Continued on next page

Table A.1: A `longtblr` from the new **`tabularray`** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
37.13	G°C	Foo	$H^i = f$
496.111	kbar	Yeet	$a - P = d$
0.23	mm	Baz	$C^u = o$
3.30	nW	Baz	$I + Y = C$
0.6	kΩ	C	$p + n = t$
5.845	bar	B	$Y^H = x$
7.343	nN	Yeet	$Z/z = c$
403.0	GN	Foo	$a^Z = v$
312.81	ncd	Bar	$I - l = g$
0.457	Pa	B	$f + Q = i$
545.773	k°C	Bar	$i + j = W$
76.5	N	Hello World	$o/q = A$
67.20	A	Hello World	$T^e = M$
2.571	Gg	Foo	$O + o = j$
285.542	nPa	A	$o + T = Z$
12.458	V	A	$k/K = b$
668.0	V	B	$p + E = o$
799.923	km	Hello World	$p^j = X$
413.0	n°C	Foo	$j^s = l$
86.64	mg	Yeet	$F^k = C$
50.931	ncd	A	$T^j = M$
0.75	K	Yeet	$X/g = z$
0.184	mK	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$i + k = K$
24.0	GK	Baz	$R + H = a$
583.56	nA	B	$R + C = r$
4.92	GPa	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$P/O = z$

Continued on next page

Table A.1: A `longtblr` from the new **`tabularray`** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
196.104	GW	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$u^R = g$
11.2	GV	C	$q + P = R$
60.14	nbar	Bar	$E^B = B$
347.91	mA	Baz	$u^t = M$
563.70	Ω	Baz	$S/y = D$
0.43	mK	Yeet	$w/z = u$
3.851	mK	A	$m/Z = c$
0.74	g	Foo	$b^d = F$
0.98	nΩ	Yeet	$m/w = G$
942.0	Gcd	Hello World	$T^I = S$
426.661	nA	Hello World	$b/n = H$
663.729	GN	Yeet	$j + D = Z$
0.65	ncd	C	$J - s = E$
0.29	bar	B	$f + J = M$
433.72	W	Foo	$s^k = r$
0.96	ng	C	$H/P = m$
51.79	nV	Baz	$b/G = b$
0.744	kW	B	$G^C = X$
0.9	cd	B	$H + i = U$
416.71	GK	Foo	$H^j = I$
447.0	°C	Bar	$N + o = K$
0.65	°C	Yeet	$K - h = F$
0.954	nPa	Hello World	$z + Y = J$
874.234	nW	Baz	$M + i = w$
0.14	V	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Q + i = v$
4.427	mN	Baz	$B/B = p$

Continued on next page

Table A.1: A `longtblr` from the new **tabularray** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
5.4	kg	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$t/A = w$
0.60	cd	B	$n^P = K$
163.0	kA	Yeet	$M + w = b$
53.911	Ω	Yeet	$E - y = a$
61.0	Gcd	Yeet	$T - M = E$
0.686	kcd	Foo	$d^a = f$
6.354	bar	A	$F/L = G$
2.88	k°C	Bar	$M^r = T$
283.457	mg	Hello World	$D - E = u$
490.455	m	Hello World	$S/u = E$
960.1	A	Yeet	$r/M = q$
884.0	bar	B	$w^R = v$
824.9	kW	Baz	$Z - t = T$
91.6	V	Yeet	$E/o = s$
45.0	N	Baz	$Z - m = P$
75.0	nW	C	$Y - U = R$
778.0	mK	Foo	$x^j = L$
0.885	mW	C	$u^S = G$
70.49	mcd	Yeet	$z^R = r$
185.0	kg	Bar	$O + T = A$
4.880	GΩ	A	$F/U = q$
0.78	GK	Bar	$m + C = E$
712.36	nbar	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Z + D = p$
37.13	G°C	Foo	$H^i = f$
496.111	kbar	Yeet	$a - P = d$
0.23	mm	Baz	$C^u = o$

Continued on next page

Table A.1: A `longtblr` from the new **tabularray** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
3.30	nW	Baz	$I + Y = C$
0.6	kΩ	C	$p + n = t$
5.845	bar	B	$Y^H = x$
7.343	nN	Yeet	$Z/z = c$
403.0	GN	Foo	$a^Z = v$
312.81	ncd	Bar	$I - l = g$
0.457	Pa	B	$f + Q = i$
545.773	k°C	Bar	$i + j = W$
76.5	N	Hello World	$o/q = A$
67.20	A	Hello World	$T^e = M$
2.571	Gg	Foo	$O + o = j$
285.542	nPa	A	$o + T = Z$
12.458	V	A	$k/K = b$
668.0	V	B	$p + E = o$
799.923	km	Hello World	$p^j = X$
413.0	n°C	Foo	$j^s = l$
86.64	mg	Yeet	$F^k = C$
50.931	ncd	A	$T^j = M$
0.75	K	Yeet	$X/g = z$
0.184	mK	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$i + k = K$
24.0	GK	Baz	$R + H = a$
583.56	nA	B	$R + C = r$
4.92	GPa	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$P/O = z$
196.104	GW	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$u^R = g$
11.2	GV	C	$q + P = R$
60.14	nbar	Bar	$E^B = B$

Continued on next page

Table A.1: A `longtblr` from the new **`tabularray`** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
347.91	mA	Baz	$u^t = M$
563.70	Ω	Baz	$S/y = D$
0.43	mK	Yeet	$w/z = u$
3.851	mK	A	$m/Z = c$
0.74	g	Foo	$b^d = F$
0.98	nΩ	Yeet	$m/w = G$
942.0	Gcd	Hello World	$T^I = S$
426.661	nA	Hello World	$b/n = H$
663.729	GN	Yeet	$j + D = Z$
0.65	ncd	C	$J - s = E$
0.29	bar	B	$f + J = M$
433.72	W	Foo	$s^k = r$
0.96	ng	C	$H/P = m$
51.79	nV	Baz	$b/G = b$
0.744	kW	B	$G^C = X$
0.9	cd	B	$H + i = U$
416.71	GK	Foo	$H^j = I$
447.0	°C	Bar	$N + o = K$
0.65	°C	Yeet	$K - h = F$
0.954	nPa	Hello World	$z + Y = J$
874.234	nW	Baz	$M + i = w$
0.14	V	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Q + i = v$
4.427	mN	Baz	$B/B = p$
5.4	kg	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$t/A = w$
0.60	cd	B	$n^P = K$
163.0	kA	Yeet	$M + w = b$

Continued on next page

Table A.1: A `longtblr` from the new **`tabularray`** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
53.911	Ω	Yeet	$E - y = a$
61.0	Gcd	Yeet	$T - M = E$
0.686	kcd	Foo	$d^a = f$
6.354	bar	A	$F/L = G$
2.88	k°C	Bar	$M^r = T$
283.457	mg	Hello World	$D - E = u$
490.455	m	Hello World	$S/u = E$
960.1	A	Yeet	$r/M = q$
884.0	bar	B	$w^R = v$
824.9	kW	Baz	$Z - t = T$
91.6	V	Yeet	$E/o = s$
45.0	N	Baz	$Z - m = P$
75.0	nW	C	$Y - U = R$
778.0	mK	Foo	$x^j = L$
0.885	mW	C	$u^S = G$
70.49	mcd	Yeet	$z^R = r$
185.0	kg	Bar	$O + T = A$
4.880	GΩ	A	$F/U = q$
0.78	GK	Bar	$m + C = E$
712.36	nbar	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Z + D = p$
37.13	G°C	Foo	$H^i = f$
496.111	kbar	Yeet	$a - P = d$
0.23	mm	Baz	$C^u = o$
3.30	nW	Baz	$I + Y = C$
0.6	kΩ	C	$p + n = t$
5.845	bar	B	$Y^H = x$
7.343	nN	Yeet	$Z/z = c$

Continued on next page

Table A.1: A longtblr from the new **tabularray** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
403.0	GN	Foo	$a^Z = v$
312.81	ncd	Bar	$I - l = g$
0.457	Pa	B	$f + Q = i$
545.773	k°C	Bar	$i + j = W$
76.5	N	Hello World	$o/q = A$
67.20	A	Hello World	$T^e = M$
2.571	Gg	Foo	$O + o = j$
285.542	nPa	A	$o + T = Z$
12.458	V	A	$k/K = b$
668.0	V	B	$p + E = o$
799.923	km	Hello World	$p^j = X$
413.0	n°C	Foo	$j^s = l$
86.64	mg	Yeet	$F^k = C$
50.931	ncd	A	$T^j = M$
0.75	K	Yeet	$X/g = z$
0.184	mK	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$i + k = K$
24.0	GK	Baz	$R + H = a$
583.56	nA	B	$R + C = r$
4.92	GPa	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$P/O = z$
196.104	GW	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$u^R = g$
11.2	GV	C	$q + P = R$
60.14	nbar	Bar	$E^B = B$
347.91	mA	Baz	$u^t = M$
563.70	Ω	Baz	$S/y = D$
0.43	mK	Yeet	$w/z = u$
3.851	mK	A	$m/Z = c$

Continued on next page

Table A.1: A `longtblr` from the new `tabularray` package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
0.74	g	Foo	$b^d = F$
0.98	nΩ	Yeet	$m/w = G$
942.0	Gcd	Hello World	$T^I = S$
426.661	nA	Hello World	$b/n = H$
663.729	GN	Yeet	$j + D = Z$
0.65	ncd	C	$J - s = E$
0.29	bar	B	$f + J = M$
433.72	W	Foo	$s^k = r$
0.96	ng	C	$H/P = m$
51.79	nV	Baz	$b/G = b$
0.744	kW	B	$G^C = X$
0.9	cd	B	$H + i = U$
416.71	GK	Foo	$H^j = I$
447.0	°C	Bar	$N + o = K$
0.65	°C	Yeet	$K - h = F$
0.954	nPa	Hello World	$z + Y = J$
874.234	nW	Baz	$M + i = w$
0.14	V	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Q + i = v$
4.427	mN	Baz	$B/B = p$
5.4	kg	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$t/A = w$
0.60	cd	B	$n^P = K$
163.0	kA	Yeet	$M + w = b$
53.911	Ω	Yeet	$E - y = a$
61.0	Gcd	Yeet	$T - M = E$
0.686	kcd	Foo	$d^a = f$
6.354	bar	A	$F/L = G$

Continued on next page

Table A.1: A `longtblr` from the new **`tabularray`** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
2.88	k°C	Bar	$M^r = T$
283.457	mg	Hello World	$D - E = u$
490.455	m	Hello World	$S/u = E$
960.1	A	Yeet	$r/M = q$
884.0	bar	B	$w^R = v$
824.9	kW	Baz	$Z - t = T$
91.6	V	Yeet	$E/o = s$
45.0	N	Baz	$Z - m = P$
75.0	nW	C	$Y - U = R$
778.0	mK	Foo	$x^j = L$
0.885	mW	C	$u^S = G$
70.49	mcd	Yeet	$z^R = r$
185.0	kg	Bar	$O + T = A$
4.880	GΩ	A	$F/U = q$
0.78	GK	Bar	$m + C = E$
712.36	nbar	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Z + D = p$

[1] Some footnote.

Note: Some general hint regarding the table, which might span multiple lines.

Rest of this page intentionally left blank.

Appendix B

Another appendix chapter

More stuff can go into the next appendix chapter, for example algorithms and code.

Rest of this page intentionally left blank.

Bibliography

Baehr, Hans Dieter and Stephan Kabelac (2016). *Thermodynamik: Grundlagen und technische Anwendungen*. 16., aktualisierte Auflage. Lehrbuch. Berlin: Springer Vieweg. 671 pp. (cit. on p. 38).

Dirac, Paul Adrien Maurice (1981). *The Principles of Quantum Mechanics*. International Series of Monographs on Physics. Clarendon Press (cit. on pp. 25, 26).

Dubbel, Heinrich, Karl-Heinrich Grote, and J. Feldhusen (2007). *Taschenbuch Für Den Maschinenbau*. 22nd ed. Berlin Heidelberg New York: Springer. 1 p. (cit. on p. 38).

Einstein, Albert (1905). "Zur Elektrodynamik Bewegter Körper. (German) [On the Electrodynamics of Moving Bodies]." In: *Annalen der Physik* 322.10 (10), pp. 891–921 (cit. on p. 25).

Goossens, Michel, Frank Mittelbach, and Alexander Samarin (1993). *The \LaTeX\ Companion*. Reading, Massachusetts: Addison-Wesley (cit. on p. 25).

International Organization for Standardization (Mar. 2017). *ISO 8217:2017 - Petroleum Products - Fuels (Class F) - Specifications of Marine Fuels*. Standard 8217. ISO/TC 28/SC 4, p. 23 (cit. on p. 38).

Knuth, Donald E. (1973). *Fundamental Algorithms*. 3rd ed. Vol. 1. 4 vols. The Art of Computer Programming. Reading, Mass.: Addison-Wesley (cit. on p. 25).

Knuth, Donald Ervin (1986). *The TeXbook*. Computers & Typesetting A. Reading, Mass: Addison-Wesley. 483 pp. (cit. on p. 25).

Mathworks (2020). *Create a Simple Class - MATLAB & Simulink*. URL: https://www.mathworks.com/help/matlab/matlab_oop/create-a-simple-class.html (visited on 04/14/2020) (cit. on p. 57).

Mollenhauer, Klaus and Helmut Tschöke (2007). *Handbuch Dieselmotoren*. 3., neubearb. Aufl. VDI-Buch. Berlin: Springer. 702 pp. (cit. on p. 38).

Wikipedia Contributors (Jan. 5, 2021). *Modelica*. In: *Wikipedia*. Ed. by Wikipedia Contributors (cit. on p. 59).

Further Reading

The following references were used in this work but not cited in the text body; they are provided here as-is.

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural Language Processing with Python*. 1st ed. Beijing ; Cambridge [Mass.]: O'Reilly. 479 pp.

McKinney, Wes (2018). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. Second edition. Sebastopol, California: O'Reilly Media, Inc. 524 pp.

Ramalho, Luciano (2015). *Fluent Python*. First edition. Sebastopol, CA: O'Reilly. 743 pp.

Index

Terms

Names