Partial Solutions to Universal Problems

Alex A. Wiseguy

MASTERARBEIT

eingereicht am

Fachhochschul-Masterstudiengang

Universal Computing

in Hagenberg

im Juni 2024

Advisor:

Roger K. Putnik, M.Sc.

Declaration

Contents

Preface

This is version **2023/11/06** of the LaTeX document template for various theses at the School of Informatics, Communication and Media at the University of Applied Sciences Upper Austria in Hagenberg. We are pleased to learn that this document collection is meanwhile also used at various other institutions in Austria and abroad.

The document was initially created in response to requests from students after the 2000/01 academic year when an official LaTeX introductory course was offered in Hagenberg for the first time. The fundamental idea was to "simply" convert the already existing *Microsoft Word* template for diploma theses to LaTeX and possibly to add some unique features. This quickly turned out to be not very useful since LaTeX, especially concerning the handling of literature and graphics, requires a substantially different way of working. The result is— rewritten from scratch and much more extensive than the previous document—a manual for writing with LaTeX, supplemented with additional (meanwhile removed) hints for *Word* users. Technical details of the current version can be found in Appendix **??**.

While this document was initially intended exclusively for the preparation of diploma theses, it now also covers *master theses*, *bachelor theses*, and *internship reports*. The differences between these documents have been deliberately kept small.

When creating this template, an attempt was made to work with the basic functionality of LaTeX and—as far as possible—to achieve this without additional packages. This was only partially successful; several supplementary "packages" are necessary, but only common extensions have been used. Of course, there is a large number of additional packages which can be helpful for further improvements and refinements. Everyone is encouraged to experiment with these as soon as they have the necessary self-confidence and sufficient time to experiment. Many details and tricks are not explicitly mentioned in this document but can be explored in the underlying source code at any time.

Numerous colleagues have provided valuable support through careful proofreading and constructive suggestions for improvement. We thank Heinz Dobler for consistently improving our "computer slang" and Elisabeth Mitterbauer for her proven "orthographic eye".

Usage of this template is free without any restrictions and not bound to any mention. However, when used as a basis for one's work, one should not simply start working on it, but at least *read* the essential parts of the document and, if possible, take them to heart. Experience has shown that this improves the quality of the results significantly.

This document and the associated LaTeX classes have been available since November

2017 on CTAN[1] as package `hagenberg-thesis`,

https://ctan.org/pkg/hagenberg-thesis.

The current source code, as well as additional materials—such as a wiki with instructions for the integration of often requested functionalities and extensions—can be found at

https://github.com/Digital-Media/HagenbergThesis.[2]

Despite great efforts, a document like this always contains errors and shortcomings. Comments, suggestions, and helpful additions are welcome. Ideally, as comments or issues on GitHub.

By the way, here, in the preface (which is common in diploma and master theses but dispensable for bachelor's theses), you may briefly describe the genesis of the document. This is also the place for any acknowledgments (e.g., to the supervisor, the examiner, the family, the dog, etc.) as well as dedications and philosophical remarks. These should be balanced and limited to a maximum of two pages.

W. Burger (em.) and W. Hochleitner

University of Applied Sciences Upper Austria
Department of Digital Media, Hagenberg
https://www.fh-ooe.at/campus-hagenberg/

[1]Comprehensive TeX Archive Network

[2]https://github.com/Digital-Media/HagenbergThesis/blob/main/CHANGELOG.md contains a list of chronological changes (formerly included in the appendix of this document).

Abstract

Here goes an abstract of the work, with a maximum of 1 page. Unlike other chapters, the abstract is usually not divided into sections and subsections. Footnotes are also not used here.

By the way, abstracts are often included in literature databases with the author and title of the work. It is, therefore, essential to ensure that the information in the abstract is coherent and complete in itself (i.e., without other parts of the work). In particular, *no literature references* are typically used at this point (as is the case also in the *title* of the thesis and the German *Kurzfassung*)! If such is needed—for example, because the paper is a further development of a particular, earlier publication—then *full* references are necessary for the abstract itself, e.g., [Zobel J.: *Writing for Computer Science – The Art of Effective Communication.* Springer, Singapore, 1997].

It should also be noted that special characters or list items are usually lost when records are added to a database. The same applies, of course, to the German *Kurzfassung.*

In terms of content, the abstract should not be a list of the individual chapters (the introduction chapter is intended for this purpose). However, it should provide the reader with a concise summary of your thesis. Therefore, the structure used here is necessarily different from that used in the introduction.

Kurzfassung

Dies sollte eine maximal 1-seitige Zusammenfassung Ihrer Arbeit in deutscher Sprache sein.

The German “Kurzfassung” should contain the same content as the English abstract. Therefore, try to translate the abstract precisely but not word for word. When translating, remember that certain idioms from English have no counterpart in German or must be formulated differently. Also, word order in German is very different from English (more on this in Section **??**). Without knowledge of the German language, it is acceptable to resort to translators. Nevertheless, hiring a skillful person for proofreading is recommended even with the highest confidence in one’s German knowledge.

The correct translation for “diploma thesis” is *Diplomarbeit*, a “master thesis” is called *Masterarbeit*. For “bachelor’s thesis”, *Bachelorarbeit* is the appropriate translation.

By the way, for this section, the *language setting* in LaTeX should be switched from English to German to get the correct form of hyphenation. However, the correct quotation marks must be set manually (see Sections **??** and **??**).

Chapter 1

MMIX.

Chapter 2

Arithmetic.

Introduction

I wanted to write an article about Fixed Point Arithmetic aimed at high-school students. I had some doubts about the early development of computers and the use of fixed and floating point arithmetic. I thought that perhaps we should have strived for using integer arithmetic to get exact answers. I am sure there are limitations to using only integer arithmetic that I want to more fully understand.

I asked ChatGPT about it and it led me to read Volume 2 of Knuth's classics. From time to time, I have had an urge to read his books and solve at least some of the exercises. Like many others, however, I am an eager starter and slow (but deep) doer, reader, and problem-solver. Finally, I feel like I can devote some time to embark on my arduous-looking Knuth journey again.

As a result, I have postponed my plan to complete the article I was writing. Hopefully, I will return to it someday and it might then contain fewer mistakes.

I record my notes and answers to his exercises beginning with the fourth chapter. The section numbers have been removed but section names match those from the book. I am reading the book on Safari; thanks to O'Reilly and Addison-Wesley Professional. I believe the ebooks published by MSP are at least as faultless as the printed books which are painstakingly proofread and corrected by Knuth and many others.

Arithmetic is fun and it is crucial. In this chapter we analyze algorithms to perform arithmetic operations (addition, subtraction, multiplication, division, and, I guess, exponentiation) on integers, "floating point" numbers, extremely large numbers, rational numbers (Knuth identifies rationals as different from floating point numbers), polynomials, and power series. In addition, we discuss radix (base) conversion, factorization, polynomial evaluation.

Positional Number Systems

Numbers are abstract ideas; their *textual representations* are concrete. Textual representations of a number may differ depending upon the notation used.

In the beginning, there were only positive integers. Then came the rational numbers (fractions). The positional notation using base b (or *radix* b) is defined for the rational numbers by the rule

$$(\dots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \dots)_b = \dots + a_2 b^2 + a_1 b + a_0 + a_{-1} b^{-1} + a_{-2} b^{-2} + a_{-3} b^{-3} \dots \quad (2.1)$$

In equation (2.1), if $b \in \mathbb{N}, b > 1$, then there are b "digits": $0 \leq a_k < b$ that express a non negative rational number *positionally.* The '.' in the notation is variously called the 'point', 'decimal point', 'radix point', 'binary point', etc. The number to its left is the *integer part* and that to its right is the *fractional part.* The value (magnitude) of the fractional part is less than 1.

Babylonians used the base- or radix-60 (Sexagesimal) system as early as 1750 B.C. In fact, they had a form of floating point representation! The exponent of the radix (60) was to be *understood from context.* The numbers $2, 120, 7200, \frac{1}{30}$ can all be written as 2 because $2 = 2 \cdot 60^0$, $120 = 2 \cdot 60^1$, $7200 = 2 \cdot 60^2$, and $\frac{1}{30} = 2 \cdot 60^{-1}$ respectively. Here are some interesting features of positional number systems:

- In Babylonian system, the "square of 30 is 15" is same as saying "The square of $\frac{1}{2}$ is $\frac{1}{4}$" because 30 means $30 \cdot 60^{-1}$ which is $\frac{1}{2}$ and 15 means $15 \cdot 60^{-1}$ which is $\frac{1}{4}$ in this context.

  The reciprocal[1] of $81 = (1 \quad 21)_{60}$ (i.e. $\frac{1}{81}$) is $(44 \quad 26 \quad 40)_{60}$ and its square (i.e. $\frac{1}{6561}$) is $(32 \quad 55 \quad 18 \quad 31 \quad 6 \quad 40)_{60}$ (See A.1 for details). Here too one can see that *the point floats*: It is placed after the first leading zero in the former, whereas after the first two leading zeros in the latter; in both the cases, however, the leading zeros do *not* appear in the notation and we deduce the exponents of 60 from context.

- Knuth mentions in [7] on historical study of *algorithms* that 60 is a large enough integer for a Babylonian mathematician to estimate (and not requiring an explicit mention of) the correct exponent of 60 relatively easily. Babylonians had developed detailed tables and algorithmic procedures for some fairly involved computational tasks as demonstrated in their clay tablets. Among the tables routinely used by the Babylonians was a table of reciprocals[2].

  Mayans in Central America apparently were the first ones to develop the fixed point positional notation. Their system was based on radix 20.

Knuth then discusses the long history of the positional notation.

A *Decimal Computer* is the one whose basic unit of data was the decimal digit, encoded in one of several schemes like binary-coded decimal (BCD). The Wikipedia article [14] on Decimal Computer says, "The rapid improvements in general performance of binary machines eroded the value of decimal operations." The use of the binary notation was a radical departure from the past then and it was spurred by John von Neumann's design of the computer to which several unsung heroes contributed. W. Buchholz mentions the reasons for using binary instead of decimal notation for an IBM computer in [1].

The MIX (and its successor, MMIX) computer designed by Knuth is oblivious to the base in most cases. Although there are exceptions, most algorithms are unaffected by the choice of the base.

[1] I had to write a computer program using only integer arithmetic to find an accurate representation; I am amazed at the ability of the ancients who did it without any help from machines!

[2] I wrote a computer program to generate this table.

Table 2.1: A sample Babylonian table of reciprocals.

Decimal	Babylonian	Decimal	Babylonian	Decimal	Babylonian
2	30	11	–	20	3
3	20	12	5	21	–
4	15	13	–	22	–
5	12	14	–	23	–
6	10	15	4	24	$2, 30$
7	–	16	$3, 45$	25	$2, 24$
8	$7, 30$	17	–	26	–
9	$6, 40$	18	$3, 20$	27	$2, 13, 20$
10	6	19	–	28	–

Representing Negative Numbers

We are only considering consistent notations for numbers. How numbers are actually realized in a physical computer is an important but separate topic. For now, our concern is about a *representation*. We need to *abstract* out a machine and concentrate on one thing–representation of numbers, all kinds of numbers. Presently, we focus on negative rational numbers.

Knuth urges his readers to study the *machine* seriously. He designed his MIX (these Roman literals represent the number 1009) computer in the Volume 1 of his books, specified its complete instruction set, implemented most algorithms (that he discussed in his books) in MIX's instruction set, and then upgraded it in 2009 to MMIX (which is pronounced *Em-micks* and which represents the number 2009) that uses the RISC (diminished Instruction Set Computer) instruction format. He described MMIX in what he called the Volume 1 - Fascicle 1. If you read his *The Art of Computer Programming Volume 1: Fundamental Algorithms*, you should skip §1.3 MIX and read §1.3′ MMIX in Fascicle 1 instead. Martin Ruckert ported all the MIX programs to MMIX in the MMIX Supplement to Knuth's books (see also [18]). We study MMIX in 1.

There are several ways to represent negative numbers in a computer. We consider a decimal computer ([14]) in this section. A digit can represent 10 distinct values. The 'sign' applies to the integer part of the positional notation (2.1).

- Signed magnitude representation. We reserve one *position* for the minus sign. If we have ten positions for the digits, then we need another position to hold just the $-$ sign:

  $$-1234567890$$

  One disadvantage is 0000000000 and -0000000000 are distinct representations for the same number, zero. This may seem like a minor limitation, but an interesting challenge is to represent negative numbers without the provision of a special "sign" position (also called the "sign digit."

- Radix complement representation. This eliminates the need for a sign digit. We first introduce the idea of the "complement" of a number and then employ it to represent negative integers. The Wikipedia page on the Method of Complements [17] gives an excellent overview of how mechanical calculators handled negative numbers using this method in the decimal notation and its generalization, including the arithmetic involved. The generalizations are called radix complement and diminished radix complement. When the radix is 10, the two complements are called *ten's* complement and *nines'* complement, respectively (Knuth prefers the placement of apostrophes as shown).

Definition 1. *The ten's complement of an n-digit number y is defined as* $y_c = 10^n - y$ *and the nines' complement is defined as* $y_{dc} = (10^n - 1) - y$.

$$y_c = y_{dc} + 1 \tag{2.2}$$

follows from Definition(1).

In the decimal[3] arithmetic system with n integral digits, the number 10^n is called the *modulus*. It is the smallest positive integer that cannot be represented in the system. Calculations are said to be done *modulo* 10^n. This entails modular arithmetic when manipulating integers. An *overflow* occurs when arithmetic operations result in numbers greater than or equal to the modulus. You can ignore an overflow, but acknowledging it is usually helpful. A general-purpose computer is a finite-resource-machine. Since we are not used to such a limitation (when we need, we can always use another digit in our paper-and-pencil calculations), we cannot always appreciate the issues (e.g. overflow) that it has to address.

Consider two n-digit integers, x and y. Let's first see how we might calculate $x + y$. Why, just add like in grade school. And that is the way to do it if no overflow occurs. For instance, if $n = 1$, $x = 5$, and $x = 6$, then, if the overflow is ignored, a computer will report $x + y$ as $11 \mod 10^1 = 11 \mod 10 = 1$.

[3]The arithmetic applies to all integral bases greater than 1; there is nothing special about 10

Chapter 3

Figures, Tables, and Source Code

Figures and tables are usually placed together with a numbered *caption* (see Figure 3.1). The main text *must* contain a *reference* to each figure and the actual figure should be positioned *after* the first reference in the LaTeX source text.

3.1 Let Them Float!

Placing figures and tables is one of the most difficult tasks in typesetting because they usually take up a lot of space and often cannot be placed on the current page in the running text. These elements must therefore be moved to a suitable place on subsequent pages, which is very tedious to do manually (but necessary in *Word*, for example).

When positioning these elements, an attempt is made, on the one hand, to leave as little empty space as possible in the text flow and, on the other hand, not to move the figures and tables too far away from the original text passage. In LaTeX, this works

Figure 3.1: Coca-Cola Werbung 1940 [13].

largely automatically by treating figures, tables and the like as “floating bodies”.

The idea that illustrations, for example, will hardly ever fit exactly in the desired position and possibly not even on the same page may appear strange or even frightening for many beginners. Nevertheless, for the time being LaTeX should confidently be left to do this work and the author should *not* manually intervene! Only when the complete document “stands” and the automatic placement still appears unsatisfactory, interventions in *individual* situations are justified.[1]

3.2 Captions

For figures, the caption is usually placed at the *bottom*, while for tables—depending on the adopted convention—*above* (as in this document), but sometimes also at the *bottom*. In LaTeX numbering of figures is also done automatically, as well as the entry into the (optional) list of figures at the end of the document.[2]

Captions are marked in LaTeX using the `\label{}` statement, which must immediately follow the `\caption{}` statement:

```
\begin{figure}
    \centering
    \includegraphics[width=.95\textwidth]{cola-public-domain-photo-p}
    \caption{Coca-Cola Werbung 1940 \cite{CocaCola1940}.}
    \label{fig:CocaCola}
\end{figure}
```

The *name* of the label (`fig:CocaCola`) can be chosen arbitrarily. The specific tag “`fig:`” is (as mentioned in Section **??**) just a useful aid to better distinguish different label types when writing.

The length of the captions can be very different. Depending on the application and style, sometimes a very short caption (Figure 3.1) or a longer one (Figure 3.2) results. Note how short captions are *centered* while long captions use *justified* formatting (LaTeX does this automatically). Captions should *always* end with a period.[3]

3.3 Figures

For the inclusion of graphics in LaTeX the use of the standard package `graphicx` [3] is recommended (automatically loaded by the `hagenberg-thesis` package). With the current workflow (using `pdflatex`) images and graphics can only be included in the following formats:

- PNG: for gray, B/W and color raster images (preferred),
- JPEG: for photographs (if not otherwise available),

[1]By adding specific placement instructions (see [11, p. 39]).

[2]A separate list of figures at the end of the document is easy to create, but it is unnecessary in a thesis (and everywhere else, actually). It should therefore be omitted. However, if your advisor insists on it, you can find instructions on how to add it to your document in the wiki of the `hagenberg-thesis` GitHub repository.

[3]Interestingly, some instructions call for the exact opposite, supposedly because with classic lead type the final dots are often “broken away” in printing. One can believe that or not, but it certainly does not matter in digital printing.

Figure 3.2: Example of a long caption text. UNIVAC launched the Model 751, the first high-performance computer with semiconductor memory, in 1961. More than fifty of these computers were sold in the U.S.A. in the first year of production, primarily to military agencies, insurance companies, and major banks. It was replaced two years later by the Model 820, developed in cooperation with SPERRY. This may sound plausible, but it is complete nonsense, and the picture actually shows a System/360 mainframe computer from IBM. Image source [15].

- PDF: for vector graphics (illustrations, line drawings etc.).

For raster images, PNG should be used if possible, because the images it contains are compressed losslessly and therefore do not have any visible compression artifacts. In contrast, JPEG should only be used if the original material (photo) is already available in this format.

3.3.1 Where Are Graphic Files Located?

Images and graphics are usually stored in a subdirectory (or several subdirectories), in the case of this document in images/. This is done by the following statement at the beginning of the main document main.tex (see also Appendix **??**):

```
\graphicspath{{images/}}
```

The path `graphicspath` (relative to the main document) can be changed at any time within the document, which is quite useful if, e.g., the graphics of individual chapters are to be stored separately in corresponding directories.

3.3.2 Adjusting Picture Size

The *size* of the printed picture can be controlled by specifying a certain width or height or a scaling factor, e.g.:

```
\includegraphics[width=.85\textwidth]{ibm-360-color}
\includegraphics[scale=1.5]{ibm-360-color}
```

Note that file extensions need not be explicitly specified. This is particularly convenient when multiple workflows with different file types are used.

3.3.3 Framing Graphics

The macro `\fbox{...}` can optionally be used to create a thin frame around the graphic, for example:

```
\fbox{\includegraphics[height=50mm]{ibm-360-color}}
```

This is usually only necessary for raster images, especially if they are very bright towards the edge and would not be distinguishable from the background without a frame.

3.3.4 Raster (Pixel) Praphics

In general, raster images should be prepared in advance so that they lose as little quality as possible when printed later. It is therefore advisable to set the image size (resolution) correctly in advance (e.g. with *Photoshop*). Common resolutions related to the final image size are:

- color and gray scale images: 150–300 dpi,
- binary (black/white) images: 300–600 dpi.

A much higher resolution does not make sense due to the screening required in laser printing, even with 1200 dpi printers. Especially *screenshots* should not be displayed too small, otherwise they are hard to read (max. 200 dpi, better 150 dpi). Consider that the work should still be legible in all details even as a photocopy.

Careful With JPEG!

As a rule, images intended for use in print documents should never be saved using lossy compression methods. In particular, the use of JPEG should be avoided if possible, even if it makes many files much smaller. The exception is when the original data is only available in JPEG and has not been edited or resized for embedding. Otherwise, PNG should always be used.

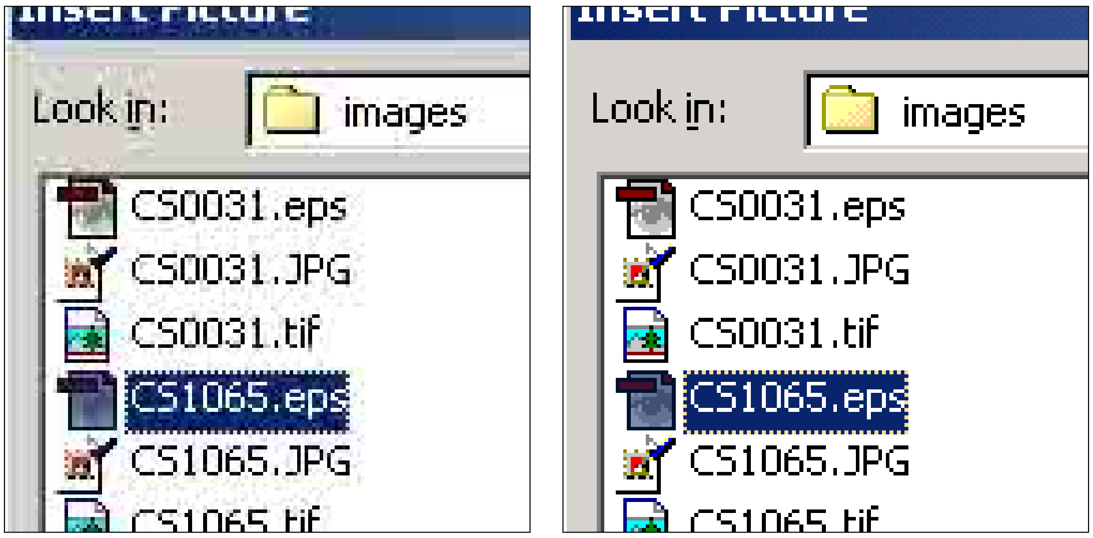
Often color screenshots are subjected to JPEG compression, although its devastating consequences should be visible to anyone (see Figure 3.3).[4]

3.3.5 Vector (Line) Graphics

For illustrations and schematic diagrams (z. B. flowcharts, entity-relationship diagrams or other structural representations), vector graphics (PDF) should always be used. Rasterized graphics, as they are usually available as GIF or PNG files on web pages, have no place in a print document; if necessary they must be re-drawn with an appropriate vector graphics tool (of course not without referencing the original source).

In this case, PDF is really the only choice, being a universal and standardized container format for many other applications. A suitable graphics program, e.g., *Adobe Illustrator* or *Inkscape*, is required to create PDF vector graphics. Note that some popular tools do not support direct export of PDF files or generate unclean files. Before deciding on a particular drawing software, this should be tried out in case of doubt. If everything else fails, PDFs can also be generated by most printer drivers.

[4]The JPEG process is designed for *natural* photos and should only be used for these.



(a) JPEG (b) PNG

Figure 3.3: Typical JPEG bungle. Screenshots and similar raster images available as originals should *never* be compressed with JPEG for print documents. The result (a) not only looks dirty compared to the uncompressed original (b), but may also become illegible in print.

Font Embedding in Graphics

The rendering of text elements depends on the fonts installed on the computer (or printer) and the form of font embedding in the source document. Correct display on the screen of your computer does not mean that the same document will be displayed exactly the same way on another computer or printer. This consideration is particularly important when print documents are made available online. Therefore, make sure that the fonts used in your graphics appear exactly as intended in the printout (see also Section 3.3.6).

Stroke Widths – Avoid *Hairlines*!

In graphics programs such as *Illustrator* and *Inkscape*, which are essentially based on PDF or SVG functionality, it is possible to define lines in terms of their thickness as “hairlines”. This should produce “as thin as possible” lines in the output. The result depends exclusively on the respective printer and is therefore hardly predictable. *Conclusion:* Avoid hairlines and always use concrete line widths ($\geq 0.25\,\text{pt}$) instead!

3.3.6 Using TeX Fonts in Graphics

As a general rule, fonts used in graphics should match the typography of the main text as closely as possible. An interesting tool for this is *Ipe Extensible Drawing Editor*,[5] a drawing program that generates text in graphics directly with LaTeX (including mathematical expressions) and uses PDF as its native file format. For images created

[5] https://ipe.otfried.org/

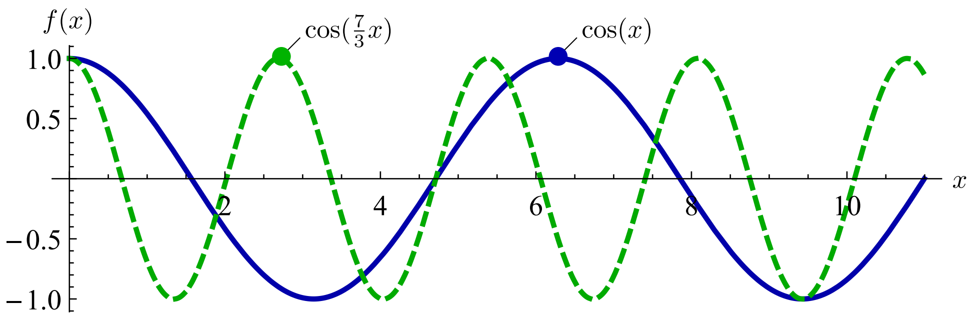


Figure 3.4: Example of using the `overpic` package to insert LaTeX elements over an imported graphic. In this case, the mathematical elements x, $f(x)$, $\cos(x)$ and $\cos(\frac{7}{3}x)$ as well as two diagonal straight lines and filled (colored) circles were inserted . All this is drawn on top of a vector graphic imported from file `mathematica-example.pdf`.

with other external graphics programs, you can use at least *similarly* looking fonts (like *Times-Roman* or *Garamond*). However, it is also possible to use the *Computer-Modern* (CM) font family from TeX/LaTeX directly to generate graphics. Some ports of CM are available as *TrueType* fonts, which can also be used in conventional graphics programs under *Windows* and *macOS*:

- Recommendable is the “BaKoMa Fonts Collection”,[6] which contains (beside the CM standard fonts) also the mathematical fonts of the AMS family.
- Another option are the “LM-Roman” Open-Type fonts,[7] which were specifically developed for use in the LaTeX environment. They are also part of the standard LaTeX distributions, such as MikTeX. These fonts include dedicated glyphs for “umlauts” and are therefore well suited for German texts as well.

Of course, these fonts must first be installed on your computer before you can use them.

3.3.7 Grapics With LaTeX Overlays (Using `overpic`)

Sometimes it is necessary to overlay an existing image or graphic with LaTeX's own (vector) elements, e.g., for markers or labels. A typical example is shown in Figure 3.4 where a PDF graphic created with *Mathematica* is annotated with mathematical text elements. This is accomplished using the `overpic` package, where the underlying graphic is not imported with `\includegraphics` but `\begin{overpic}` ... `\end{overpic}`, with a similar syntax:

```
\begin{overpic}[width=0.85\textwidth]{mathematica-example}
    \put(101,14){$x$}%
    \put(4,31){$f(x)$}%
    \put(29.5,28){\line(1,1){2}}%
    ...
\end{overpic}
```

The `overpic` environment also forms a `picture` environment[8] where LaTeX drawing

[6] http://ctan.org/pkg/bakoma-fonts
[7] http://www.gust.org.pl/projects/e-foundry/latin-modern
[8] https://www.overleaf.com/learn/latex/Picture_environment

Table 3.1: Programming languages at a glance.

Language	Type	Typical Use	Standards
C++	Compiled	Applications	ISO/IEC 14882:2020
COBOL	Compiled	Business	ISO/IEC 1989:2014
JavaScript	Interpreted	Web	ECMA-262
Python	Interpreted	Machine Learning	PEPs

instructions (such as `\put` and the like) can be placed, as shown in Figure 3.4.[9] The x/y positions are specified as a percentage of the image width. Further details can be found in the source code.

3.3.8 Figures With Multiple Elements

If multiple images or graphics are combined into one figure, a common caption is typically used, as shown in Figure 3.5. In the main text, a reference to a particular part of the figure, such as the single-row roller bearing in Figure 3.5 (c), could look like this:

```
... Figure~\ref{fig:Bearings} (c) ...
```

3.3.9 Source Citations in Captions

If images, graphics or tables from other sources are used, then their origin must be made clear in any case, and preferably directly in the caption. For example, if an illustration from a book or other citable publication is used, then it should be included in the bibliography and cited as usual with `\cite{..}` as demonstrated in Fig. 3.5. Further details on this type of source citation can be found in Chapter **??** (esp. Section **??**).

3.4 Tables

Tables are often used to present numerical relationships, test results, etc. in a clear form. A simple example is Table 3.1, the associated LaTeX source can be found in Program 3.1.

As arguments of the `tabular` environment the alignments of the individual columns are specified. The number of arguments thus determines the number of columns. Valid items are `l` for left-aligned, `c` for centered, and `r` for right-aligned. The column width results from the length of the contents, there are no automatic line breaks. To set the width and thus create automatic line wraps, `p{width}` (paragraph mode) is used, where `width` is a valid length specification (see [16] for details). The `@{}` items remove the (usually unwanted) margin at the left and right table borders.

The demand for an attractive appearance of tables has increased noticeably in recent years. For example, many authors and publishers using LaTeX now follow some simple

[9]The default drawing instructions in LaTeX are quite restrictive, so the `pict2e` package is additionally used (see https://ctan.org/pkg/pict2e).

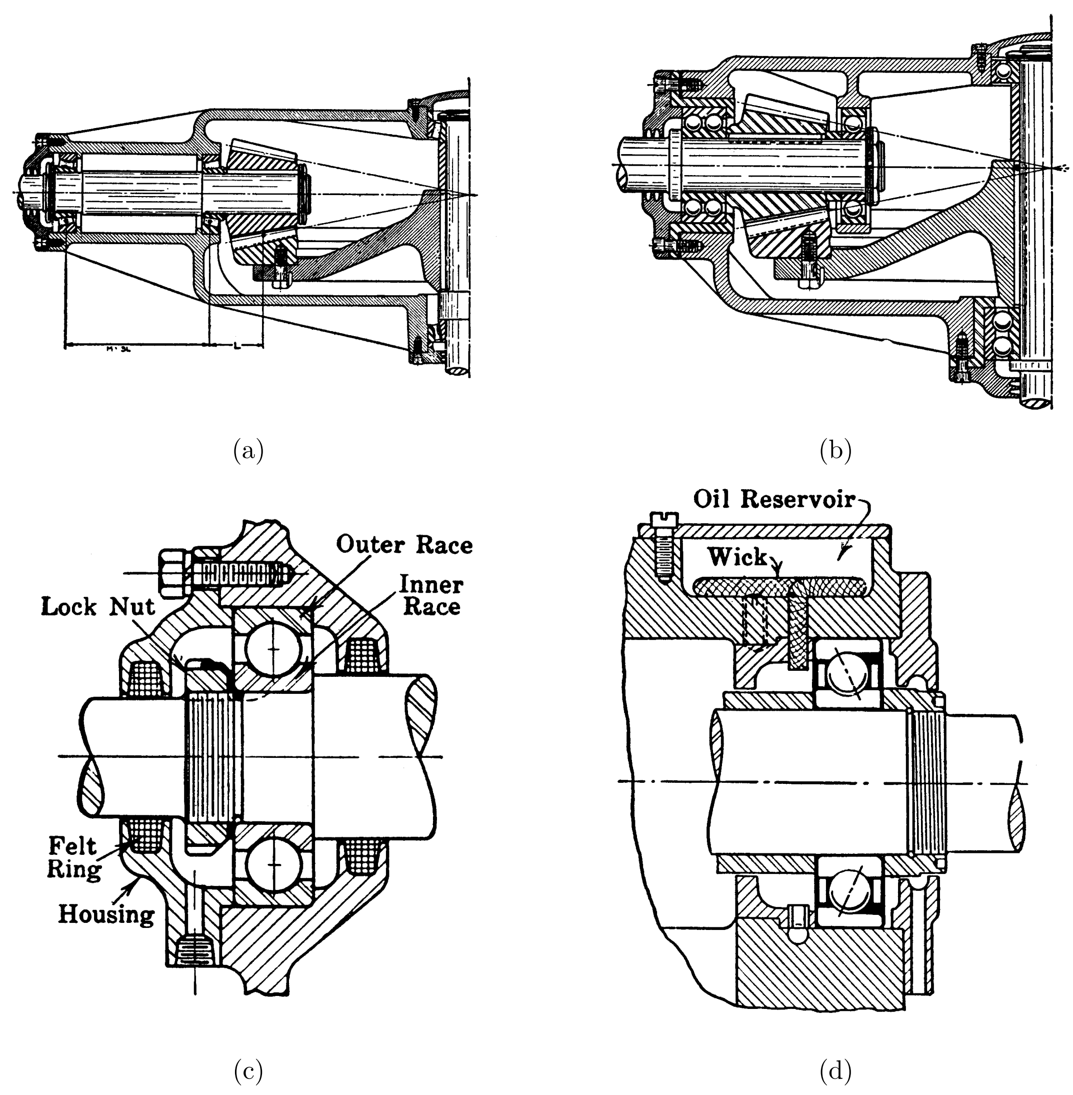


Figure 3.5: Various machine elements in an illustration with multiple elements. *Overhang mounting* (a), *straddle mounting* (b), single row roller bearing (c), lubrication of roller bearings (d). This figure uses an ordinary table (`tabular`) with 2 columns and 4 rows (details can be found in the source code). Image source [4].

design guidelines for tables [5], of which particularly the first two determine their basic layout:

1. Never use vertical lines.
2. Never use double lines.
3. Place units in the column header (not in the table content area).
4. A decimal separator is always preceded by a digit; thus write "0,1" not just ",1".

The LaTeX package `booktabs` makes it easy to meet these requirements Within the `tabular` environment (which defines the actual table), the number of columns—preferably set left-justified (`l` specifiers)— are defined first. `\toprule` marks the be-

Program 3.1: LaTeX source code for Table 3.1. The generation of the displayed listing itself is described in Section 3.5.

```
\begin{table}
    \caption{Programming languages at a glance.}
    \label{tab:programming-languages}
    \centering
    \setlength{\tabcolsep}{10pt} % separator between columns (standard = 6pt)
    \renewcommand{\arraystretch}{1.25} % vertical stretch factor (standard = 1.0)
    \begin{tabular}{@{}llll@{}}
        \toprule
        Language   & Type        & Typical Use      & Standards          \\
        \midrule
        C++        & Compiled    & Applications     & ISO/IEC 14882:2020 \\
        COBOL      & Compiled    & Business         & ISO/IEC 1989:2014  \\
        JavaScript & Interpreted & Web              & ECMA-262           \\
        Python     & Interpreted & Machine Learning & PEPs               \\
        \bottomrule
    \end{tabular}
\end{table}
```

Table 3.2: Example of a table with multiline text in narrow columns. Here the columns are too narrow for justification, so left alignment ("ragged-right") is used.

Method	Implem.	Features	Status
polygon shading	SW/HW	flat-shaded polygons	
flat shading with z-buffer	SW/HW	depth values	
goraud shading with z-buffer	SW/HW	smooth shading, simple fog, point light sources	SGI entry models
phong shading with z-buffer	SW/HW	highlights	
texture mapping with z-buffer	SW/HW	surface textures, simple shadows	SGI high end, flight simulators

ginning of the table, followed by the header, which is terminated by `\midrule`. This is followed by the lines with the table contents. Using `\bottomrule` the table is closed with another horizontal line. `\midrule` calls can also occur more often to divide the table. If horizontal lines are needed that should not span all columns, `\cmidrule` can be used.

3.4.1 Long Texts in Table Columns

Sometimes it is necessary to fit relatively large amounts of text into narrow columns in tables, as in Table 3.2. In this case, it makes sense to go without justification and at the same time relax the strict hyphenation rules. Details can be found in the corresponding LaTeX source text.

3.4.2 Multipage Tables

Often tabular information needs more than one page. Here the float-element created by the `table` environment becomes a problem, because it prevents breaks across multiple pages. To allow page breaks *within* tables, the `longtable` package can be used.[10] It replaces the `tabular` environment and does not require a surrounding `table` environment.[11]

The table definition is largely the same as shown in Program 3.1. Only the commands `\endhead` and `\endfoot` are added. They outline the head and footer areas to be repeated on each page. If these should be different for the header on the first page and the footer on the last page, `\endfirsthead` and `\endlastfoot` are used.

Table 3.3 shows a concrete example for using `longtable`. A specific header is assigned to the first page, defining the main caption text and the associated label. The following pages show an abbreviated header ("*continued*") with the *same* table number, which is only defined once in the main header. If horizontal and vertical spacing are to be modified, these statements must be placed *before* the beginning of the table and enclosed with `{...}` or `\begin{block}` ...`\end{block}`[12] to restrict their scope. Consult the source code of this document for additional details.

Table 3.3: A long table (using `longtable`) that breaks over two pages. Note that different table headers are defined for the first page and subsequent pages and the associated label is only assigned once (referring to *this* page).

First Column	Second Column
The column on the right contains a lot of text.	There is a lot of text in this column, which creates a long column. Here, too, it can make sense to set the content left-aligned. However, `longtable` does not wrap lines in such columns, only between individual lines.
More lines follow here.	This content serves only as a placeholder.

[10] http://mirrors.ctan.org/macros/latex/required/tools/longtable.pdf

[11] Note that a `longtable` is *no* float element but always gets inserted at the current text position. This may lead to large empty blocks if the table header and/or the first table row do not fit onto the current page.

[12] The (dummy) "`block`" environment is defined in `hgb.sty`. It does nothing but provides a limited scope for temporarily setting (and automatically resetting) LaTeX variables.

Table 3.3 (*continued*)

First Column	Second Column
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.

Table 3.3 (*continued*)

First Column	Second Column
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.
More lines follow here.	This content serves only as a place-holder.

3.4.3 Joining Columns and Rows

To combine several columns in a table into one, the statement

```
\multicolumn{number}{format}{text}
```

is used. Here `number` defines the set of columns to be joined. `format` specifies the alignment to use analogous to the specification in `tabular`-environment and `text` is the included text.

Analogously, to combine several lines into one, you can use

```
\multirow{number}{width}{text}
```

Here `number` represents the number of lines to be joined into one, `width` defines the joint column width. The same specifications are used as in the `tabular` environment. Additionally, `*` and `=` can be specified. The former sets the width created by the text, the latter takes the width of the column from the `tabular` specification. `text` is the content to be inserted.

The `multirow` command is placed in the first of the lines to be joined. The following rows remain empty. Table 3.4 shows a simple example with joined columns and rows.

Table 3.4: A table with joined columns and rows.

Column 1	Columns 2–3	
Row 1 Row 2	This text extends over two lines.	This text too.

Table 3.5: Language-specific code environments defined in hgb .sty.

C (ANSI):	`\begin{CCode}`	... `\end{CCode}`
C++ (ISO):	`\begin{CppCode}`	... `\end{CppCode}`
C#:	`\begin{CsCode}`	... `\end{CsCode}`
CSS:	`\begin{CssCode}`	... `\end{CssCode}`
HTML:	`\begin{HtmlCode}`	... `\end{HtmlCode}`
Java:	`\begin{JavaCode}`	... `\end{JavaCode}`
JavaScript:	`\begin{JsCode}`	... `\end{JsCode}`
LaTeX:	`\begin{LaTeXCode}`	... `\end{LaTeXCode}`
Objective-C:	`\begin{ObjCCode}`	... `\end{ObjCCode}`
PHP:	`\begin{PhpCode}`	... `\end{PhpCode}`
Python:	`\begin{PythonCode}`	... `\end{PythonCode}`
Swift:	`\begin{SwiftCode}`	... `\end{SwiftCode}`
XML:	`\begin{XmlCode}`	... `\end{XmlCode}`
Generic:	`\begin{GenericCode}`	... `\end{GenericCode}`

3.5 Program Texts

The inclusion of program texts (source code) is a frequent necessity, especially of course for work in areas related to computing.

3.5.1 Formatting Program Code

There are special packages for LaTeX to display programs which, among other things, also perform automatic line numbering, in particular the packages `listings`[13] and `listingsutf8`.[14]

These are also used to implement the language-specific environments listed in Table 3.5. Their use is extremely simple, e.g., for source code in the C programming language one writes

```
\begin{CCode}
    ...
\end{CCode}
```

The source code within these environments is interpreted in the respective programming language, while comments are preserved. These environments can be used standalone

[13] https://ctan.org/pkg/listings
[14] https://ctan.org/pkg/listingsutf8

(in the main text) or within float environments (esp. `program`). In the first case, the source text even wraps across page boundaries. With `/+` … `+/` an escape option to LaTeX is provided, which is useful for setting labels for referencing individual program lines, e.g., with

```
int w = ip.getWidth(); /+\label{ExampleCodeLabel}+/
```

An example with Java is shown in Prog. 3.2, where the above label is placed in line 14. Note that mathematical text (such as in line 21 of Prog. 3.2) can also be placed inside escaped comments.

Numbering of the Code Lines

All code environments listed in Table 3.5 can be used with optional arguments, which are especially useful to control the line numbering. In the default case (i.e., without additional specifications), with

```
\begin{someCode} ...
```

all code lines (including blank lines) are continuously numbered starting at 1. For consecutive code segments it is often helpful to let the numbering continue from the previous section, enabled by specifying the optional argument `firstnumber=last`:

```
\begin{someCode}[firstnumber=last] ...
```

To disable the numbering of the code lines altogether it is sufficient to specify the optional argument `numbers=none`:

```
\begin{someCode}[numbers=none] ...
```

In this case, of course, the use of line labels in the code has no effect.

3.5.2 Program Code Placement

Since source texts can become quite bulky, this task is not always easy to solve. Depending on the size and the relation to the main text, there are essentially three ways for including program text:

a) in the main text for short program pieces,
b) as float elements (`program`) for medium-sized programs up to one page, or
c) in the Appendix (for long programs).

Program Code in the Main Text

Short code sequences can be embedded in the running text without further ado, as long as they are of immediate importance at the given places. For example, the following (rudimentary) Java method `extractEmail` searches for an e-mail address in a given string:

```
static String extractEmail(String line) {
    line = line.trim(); // find the first blank
    int i = line.indexOf(' ');
    if (i > 0)
    return line.substring(i).trim();
    else
```

```
        return null;
    }
```

This code segment was produced with

```
\begin{JavaCode}[numbers=none]
static String extractEmail(String line) {
    line = line.trim(); // find the first blank
    ...
}
\end{JavaCode}
```

(see Section 3.5.1). In-line program pieces should be no more than a few lines long and, if possible, should not be divided by page breaks.

Program Code in Float Elements

Suppose longer code sequences are necessary, which should appear near the running text. In that case, they should be treated as float elements in the same way as illustrations and tables. These program texts should not exceed the size of one page. In an emergency, up to two pages can be packed into consecutive float elements, each with its own caption. In `hgb.sty` a float environment `program` is defined, which is used analogously to `table`:

```
\begin{program}
\caption{The title of this piece of program.}
\label{prog:xyz}
\begin{JavaCode}
  class Foo {
    ...
  }
\end{JavaCode}
\end{program}
```

If desired, the caption can also be placed at the bottom (but in any case consistently and not mixed). Of course, a linear sequence in the final printed image must not be expected here either, so phrases such as "… in the *following* program snippet …" should be avoided and numerical cross references used instead. See Programs 3.1 and 3.2 for examples.

Program Texts in the Appendix

For longer program texts, especially if they include complete implementations and are not directly relevant in any local context, storage in a separate appendix at the end of the document should be resorted to. For references to individual details, either short excerpts can be placed in the running text or appropriate page references can be used. Such an example is the LaTeX source code in Appendix **??** (page **??**).[15]

[15]It is generally questionable if the printed inclusion of much implementation code is at all useful for the reader or not better provided electronically (on physical media or online) and only described selectively.

Program 3.2: Example of a program listing (Java) as a float element.

```java
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class My_Inverter implements PlugInFilter {
    int agent_velocity;
    String title = ""; // just to test printing of double quotes

    public int setup (String arg, ImagePlus im) {
        return DOES_8G;
    }

    public void run (ImageProcessor ip) {
        int w = ip.getWidth();
        int h = ip.getHeight();

        /* iterate over all image coordinates */
        for (int u = 0; u < w; u++) {
            for (int v = 0; v < h; v++) {
                int p = ip.getPixel(u, v);
                ip.putPixel(u, v, 255 - p); // invert: I'(u,v) ← (255 − I(u,v))
            }
        }
    }
} // end of class My_Inverter
```

Chapter 4

Mathematical Formulas, Equations and Algorithms

Typesetting mathematical elements is certainly one of the strongest features of LaTeX. A distinction is made between mathematical elements in continuous text and free-standing equations, which are usually numbered consecutively. Analogous to figures and tables, this makes it easy to cross-reference equations. Here are only some examples and special topics, much more can be found in [8, Ch. 7] and [12].

4.1 Mathematical Elements in Continuous Text

Mathematical symbols, expressions, equations, etc. are marked in the continuous text by pairs `$ ... $`. Here is an example:

> The near infinity point is at $\bar{a} = f \cdot (f/(K \cdot u_{\mathrm{max}}) + 1)$, so with a lens set to ∞, everything is in focus from distance $\bar{a}$ on. Focusing the lens to the distance $\bar{a}$ (i.e., $a_0 = \bar{a}$), everything in the range $[\frac{\bar{a}}{2}, \infty]$ will be in focus.

It is important to ensure that the height of the individual math items in the text does not become too large.

Common Mistakes

In continuous text, simple variables are often typeset with plain text, i.e., without using correct mathematical symbols, as in “X-axis” instead of “$X$-axis” (`$X$-axis`).

Mathematical Fonts

LaTeX uses slightly different fonts for regular text and in math mode. The following basic math fonts are available:

Roman	`$\mathrm{Roman}$`,
Italic	`$\mathit{Italic}$`,
$\mathbf{Bold}$	`$\mathit{Bold}$`,
$\mathsf{SansSerif}$	`$\mathsf{Sans Serif}$`,
$\mathtt{Typewriter}$	`$\mathttt{Typewriter}$`,
$\mathcal{CALLIGRAPHIC}$	`$\mathcal{CALLIGRAPHIC}$`,
$\mathbb{BLACKBOARD}$	`$\mathbb{BLACKBOARD}$`,
$\mathfrak{Fraktur}$	`$\mathfrak{Fraktur}$`.

In some situations, the `\boldsymbol{..}` macro may come in useful. It can convert any mathematical symbol into a boldface version, for example, $\mathbf{A}$ (`$\mathbf{A}$`) denotes a matrix and $\boldsymbol{x}$ (`$\boldsymbol{x}$`) is a vector.

Line Breaks

With longer mathematical elements in the continuous text problems with line breaks are inevitable. Usually LaTeX allows line breaks only at the "=" operator, elsewhere one can use `\allowbreak` to enable line breaks. Here is a small example:

a) For example, (bla bla bla) a simple row vector is defined in the form $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$.
b) For example, (bla bla bla) a simple row vector is defined in the form $\boldsymbol{x} = (x_0, x_1,$
$\ldots, x_{n-1})$.

The line in a) should extend beyond the page margin, but b) contains `\allowbreak` in several places and should therefore wrap cleanly.

4.2 Displayed Expressions and Equations

Displayed mathematical expressions can be generated in LaTeX by `\[` ... `\]` The result will be centered, but will not be numbered. For example,

$$y_0 = 4x^2$$

is produced by `\[y_0 = 4 x^2\]` or, alternatively,

```
\begin{displaymath} y_0 = 4 x^2 \end{displaymath}
```

4.2.1 Numbered Equations

However, most often in such cases the `equation` environment is used to produce numbered equations that can be referred to at any time in the text. For example,

```
\begin{equation}
  f(k) = \frac{1}{N} \sum_{i=0}^{k-1} i^2 .
  \label{eq:MyFirstEquation}
\end{equation}
```

creates this equation:

$$f(k) = \frac{1}{N} \sum_{i=0}^{k-1} i^2. \tag{4.1}$$

With `\ref{eq:MyFirstEquation}` you get the number (4.1) of this equation as usual (see also Section 4.2.5). The same equation *without* numbering can be generated with the `equation*` environment.

> Note that *equations* are a *part of the main text* in terms of content, and therefore, in addition to proper linguistic *transitions*, *punctuation* (as shown in Equation 4.1) must be observed. If you are unsure, you should look at suitable examples in a good math textbook.

For those interested, more on the intimate connection between mathematics and prose can be found in [9] and [6].

4.2.2 Multiline Equations

For multiline equations LaTeX offers the `eqnarray` environment which, however, generates somewhat idiosyncratic spacing. It is recommended to use the extended possibilities of the `amsmath` package[1] for this right away. Here is an example with two equations aligned to the = sign,

$$f_1(x,y) = \frac{1}{1-x} + y, \tag{4.2}$$

$$f_2(x,y) = \frac{1}{1+y} - x, \tag{4.3}$$

generated with the `align` environment from the `amsmath` package:

```
\begin{align}
  f_1 (x,y) &= \frac{1}{1-x} + y , \label{eq:f1} \\
  f_2 (x,y) &= \frac{1}{1+y} - x , \label{eq:f2}
\end{align}
```

4.2.3 Multi-Case Constructs

With the `cases` environment from `amsmath`, case distinctions, e.g., in function definitions, are very easy to accomplish. For instance, the recursive definition

$$f(i) = \begin{cases} 0 & \text{for } i = 0, \\ f(i-1) + f(i) & \text{for } i > 0. \end{cases} \tag{4.4}$$

was produced with the following commands:

```
\begin{equation}
  f(i) =
  \begin{cases}
    0             & \text{for $i = 0$,}\\
    f(i-1) + f(i) & \text{for $i > 0$.}
  \end{cases}
\end{equation}
```

Note the use of the very handy `\text{..}` macro, which can be used to insert ordinary text in math mode, and again the punctuation within the equation.

[1]American Mathematical Society (AMS). `amsmath` is part of the LaTeX default installation and is automatically imported by `hgb.sty`.

4.2.4 Equations With Matrices

Again, `amsmath` offers some advantages over using the standard LaTeX constructs. For this purpose, a simple example of using the `pmatrix` environment for vectors and matrices,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}, \tag{4.5}$$

generated with the following instructions:

```
 1 \begin{equation}
 2   \begin{pmatrix}
 3       x' \\
 4       y'
 5   \end{pmatrix}
 6   =
 7   \begin{pmatrix}
 8       \cos \phi &          -\sin \phi \\
 9       \sin \phi & \phantom{-}\cos \phi
10   \end{pmatrix}
11   \cdot
12   \begin{pmatrix}
13       x \\
14       y
15   \end{pmatrix} ,
16 \end{equation}
```

A useful detail in this is the TeX macro `\phantom{..}` (in line 9), which inserts its argument invisibly and is used here as a placeholder for the minus sign above it. As an alternative to `pmatrix`, the `bmatrix` environment can be used to create matrices and vectors with square brackets. Numerous other mathematical constructs of the `amsmath` package are described in [10].

4.2.5 References to Equations

When referring to numbered formulas and equations, it is usually sufficient to indicate the corresponding number in round brackets, e.g.,

"... as can be derived from (4.2) ..."

To avoid misunderstandings, however—especially in texts with only few mathematical elements—"Equation 4.2", "Eq. 4.2" or "Eq. (4.2)" should be written (consistently, of course).

Note: Forward references to equations (placed further back in the text) are *extremely uncommon* and should be avoided! If one still believes to need such a thing, then usually a mistake was made in the content structure.

4.3 Mathematical Symbols

Special macros are needed for a large part of the mathematical symbols. Some of the most common ones are listed in the following.

4.3.1 Number Sets

Some frequently used symbols are unfortunately not included in the original mathematical character set of LaTeX, suchh as the symbols for the real and natural numbers. In the `hagenberg-thesis` package these symbols are defined as macros[2] `\R`, `\Z`, `\N`, `\Cpx`, `\Q` ($\mathbb{R}, \mathbb{Z}, \mathbb{N}, \mathbb{C}, \mathbb{Q}$), e.g.,

$$x \in \mathbb{R}\ ,\ k \in \mathbb{N}_0,\ z = (a + \mathrm{i} \cdot b) \in \mathbb{C}.$$

4.3.2 Operators

In LaTeX dozens of mathematical operators are defined for various purposes. Of course, the arithmetic operators $+$, $-$, $\cdot$ and $/$ are needed most often. A frequent mistake (probably resulting from programming practice) is the use of $*$ for simple multiplication—correct is $\cdot$ (`\cdot`).[3] For statements like "a field with 25×70 meters" (but also almost *only* for that) it makes sense to use the $\times$ (`\times`) operator – and *not* simply the text character "x"!

4.3.3 Variables (Symbols) With Multiple Characters

Especially in the mathematical specification of algorithms and programs it is often necessary to use symbols (variable names) with more than one character, e.g.,

$$Scalefactor \leftarrow p^2 \cdot 1.5\ ,$$

falsely implemented by

```
$Scalefactor \leftarrow Scalefactor^2! \verb!\cdot 1.5$
```

The reason is that LaTeX interprets the character sequence "Scalefactor" as the product of 11 single, consecutive variables S, c, a, l, e, ... and inserts appropriate spaces between them. The *correct* way is to combine these letters with `\mathit{..}` to *one* symbol. The difference is clearly visible in this case:

Wrong:	$Scalefactor^2$	$\leftarrow$	`$Scalefactor^2$`
Correct:	$\mathit{Scalefactor}^2$	$\leftarrow$	`$\mathit{Scalefactor}^2$`

Generally, such long symbol names should be avoided anyway and short symbols used instead wherever possible (e.g., focal length $f = 50\,\mathrm{mm}$ instead of $\mathit{focallength} = 50\,\mathrm{mm}$).

4.3.4 Functions and Operators

While symbols for variables are traditionally (and in LaTeX automatically) set *italic*, names of functions and operators are usually typeset in *roman* fonts, as for example in

$\sin\theta = \sin(\theta + 2\pi)$ $\leftarrow$ `$\sin \theta = \sin(\theta + 2 \pi)$`

This happens with the already predefined standard functions (like `\sin`, `\cos`, `\tan`, `\log`, `\max` u. v. a.) automatically. This convention should also be followed for self-defined functions, such as in

$\mathrm{dist}(A, B) := |A - B|$ $\leftarrow$ `$\mathrm{dist}(A,B) := |A-B|$`

[2]Based on the *AMS Blackboard Fonts*.

[3]The $*$ character is usually reserved for the *convolution operator*.

4.3.5 Units of Measurement and Currencies

When specifying units of measurement, normal font is usually used (no italics) should be used, e.g.:

> The maximum speed of the *Bell XS-1* is 345 m/s with a takeoff weight of 15 t. The prototype cost over US$ 25,000,000, or about € 19,200,000 in today's conversion.

The blank space between the number and the unit of measurement is intentional. The $ sign is created with `\$` and the Euro symbol (€) is created with the `\euro` macro.[4]

4.3.6 Commas in Decimal Numbers (Math Mode)

In math mode (i.e., within `$ ... $`, `\[ ... \]` or in equations) LaTeX generally follows the Anglo-American convention that *dot* (`.`) is used as the comma sign decimal numbers. For example, `$3.141$` produces the output "3.141", as one would expect. Unfortunately, to use a European comma in decimal numbers, it is *not* sufficient to simply replace `.` with `,`. In this case the comma is interpreted as *punctuation character* and the result looks like this:

`$3,141$` $\rightarrow$ $3, 141$

(note the additional blank space after the comma). This behavior can be redefined globally in LaTeX, but this in turn leads to a number of unpleasant side effects. A simple (though not very elegant) solution is to write decimal numbers in math mode like this:

`$3{,}141$` $\rightarrow$ $3{,}141$

4.3.7 Mathematical Tools

For the creation of complicated equations it is sometimes helpful to resort to special software or interactive tools. Among other things, LaTeX statements for mathematical equations can be exported from Microsoft's *Equation Editor* or *Mathematica* in a relatively simple way and copied directly (usually with some manual rework) into your own LaTeX document.

4.4 Algorithms

Algorithmic representations are an important means of accurately describing computational processes. By using *mathematical notation* (symbols and operators) on the one hand and the *sequence structures* (decisions, loops, procedures, etc.) familiar from programming, algorithms are a proven link between the mathematical formulation and the associated program code.

[4]The € symbol is not included in the original LaTeX character set but is provided by the `marvosym` package (`\EUR`).

An essential aspect of an algorithmic description—which should be structurally as similar as possible to the implementation— is its *independence* from a specific programming language. This results in better readability, broader applicability, and increased sustainability (possibly beyond the lifetime of a programming language). When formulating algorithms, one should consider the following, among other things:[5]

- Use the same short symbols (such as $a, i, x, S, \alpha \ldots$) in algorithms as you would in mathematical definitions and equations.
- If possible, use proper *mathematical* operators, such as $=$, $\leq$, $\cdot$, $\wedge$ etc., instead of the associated programming constructs `==`, `<=`, `*`, `&&`, respectively.
- Similarly. do not use elements or syntax of a specific programming language (for example, a "`;`" at the end of a statement is unnecessary).
- If an algorithm becomes too long for a page, consider how to divide it sensibly into smaller modules (perhaps the associated program structure is not optimal either).

For the notation of algorithms in mathematical form or even for pseudocode, no special support is provided in LaTeX itself. However, there are several useful LaTeX packages for this purpose, including `algorithmicx`, which is also used here because of its simple syntax, but in the improved version `algpseudocodex`.[6] The example in Algorithm 4.1 was created using the float environment `algorithm` and the `algpseudocodex` package (see the source code in Program 4.1). For better readability, vertical rules are used (`indLines=true`) and the optional keyword "`end`" at the end of blocks is omitted (`noEnd=true`).

[5]See also http://mirrors.ctan.org/macros/latex/contrib/algorithms/algorithms.pdf (Section 7).

[6]Style hgbalgo.sty of `hagenberg-thesis` extends the packages `algorithmicx` and `algpseudocodex` (see https://ctan.org/pkg/algpseudocodex) by providing improved indentation, colors, etc.

Algorithm 4.1: Example of an algorithm for bicubic interpolation in 2D [2], typeset with package `algpseudocodex`. Function Cubic1D(x), used in lines 8 and 9, calculates the weight given to the interpolation value at some one-dimensional position x.

```
 1: function BicubicInterpolation($I, x, y$)              ▷ two-dimensional interpolation
        Input: $I$, original image; $x, y \in \mathbb{R}$, continuous position.
        Returns the interpolated pixel value at position $(x, y)$.
 2:     $val \leftarrow 0$
 3:     for $j \leftarrow 0, \ldots, 3$ do                              ▷ iterate over 4 lines
 4:         $v \leftarrow \lfloor y \rfloor - 1 + j$
 5:         $p \leftarrow 0$
 6:         for $i \leftarrow 0, \ldots, 3$ do                        ▷ iterate over 4 columns
 7:             $u \leftarrow \lfloor x \rfloor - 1 + i$
 8:             $p \leftarrow p + I(u, v) \cdot \mathsf{Cubic1D}(x - u)$
            Sometimes it is useful to insert a longer, unnumbered statement extending
            over multiple lines with proper indentation. This can be done with the (non-
            standard) command \StateNN[]{..}. For long numbered (multi-line) state-
            ments use the standard \State command.
 9:         $val \leftarrow val + p \cdot \mathsf{Cubic1D}(y - v)$
10:     return $val$

11: function Cubic1D($x$)                              ▷ piecewise cubic polynomial (1D)
12:     $z \leftarrow 0$
13:     if $|x| < 1$ then
14:         $z \leftarrow |x|^3 - 2 \cdot |x|^2 + 1$
15:     else if $|x| < 2$ then
16:         $z \leftarrow -|x|^3 + 5 \cdot |x|^2 - 8 \cdot |x| + 4$
17:     return $z$
```

Program 4.1: Source code for Algorithm 4.1. As you can see, empty lines can be used here as well, which significantly improves readability.

```
\begin{algorithm}
\caption{Example of an algorithm for bicubic interpolation in 2D typeset
with the package \texttt{algpseudocodex} (from \cite{BurgerBurge2022}).
Function $\Call{Cubic1D}{x}$, used in lines \ref{alg:wcub1} and
\ref{alg:wcub2}, calculates the weight given to the interpolation value at
some one-dimensional position $x$.}
\label{alg:Example}

\begin{algorithmic}[1]      % [1] = all lines are numbered
\Function{BicubicInterpolation}{$I, x, y$} \Comment{two-dimensional interpolation}
  \Input{$I$, original image; $x,y \in \R$, continuous position.}
  \Returns{the interpolated pixel value at position $(x,y)$.\algsmallskip}

  \State $\mathit{val} \gets 0$

  \For{$j \gets 0, \ldots, 3$} \Comment{iterate over 4 lines}
    \State $v \gets \lfloor y \rfloor - 1 + j$
    \State $p \gets 0$
    \For{$i \gets 0, \ldots, 3$} \Comment{iterate over 4 columns}
      \State $u \gets \lfloor x \rfloor - 1 + i$
      \State $p \gets p + I(u,v) \cdot \Call{Cubic1D}{x - u}$ \label{alg:wcub1}
    \EndFor

    \StateNN[2]{Sometimes it is useful to insert a longer, ...}

    \State $\mathit{val} \gets \mathit{val} + p \cdot \Call{Cubic1D}{y - v}$
        \label{alg:wcub2}
  \EndFor
  \State\Return $\mathit{val}$
\EndFunction

\medskip   % \medskip can be used here, because we are in  vertical  mode
\hrule

\Function{Cubic1D}{$x$} \Comment{piecewise cubic polynomial (1D)}
  \State $z \gets 0$
    \If{$|x| < 1$}
      \State $z \gets |x|^3 - 2 \cdot |x|^2 + 1$
    \ElsIf{$|x| < 2$}
      \State $z \gets -|x|^3 + 5 \cdot |x|^2 - 8 \cdot |x| + 4$
    \EndIf
    \State\Return{$z$}
\EndFunction

\end{algorithmic}
\end{algorithm}
```

Appendix A

Some Technical Details

A.1 Floating Point Long Multiplication by Babylonians

Babylonians had a form of floating point representation of numbers. Consider the fraction $\frac{1}{6561}$ which is the square of $\frac{1}{81}$. They carried out the conventional long multiplication to yield a representation for $\frac{1}{6561}$ from that for $\frac{1}{81}$: $(44\ \ 26\ \ 40)_{60}$.

			44	26	40
	×		44	26	40
			44×40	26×40	40×40
		44×26	26×26	40×26	
	44×44	26×40	40×44		
					=1600=26×60+**40**
				26	**40**
				=2106=35×60+**6**	**40**
			35	**6**	**40**
			=4231=70×60+**31**	**6**	**40**
		70	**31**	**6**	**40**
		=2358=39×60+**18**	**31**	**6**	**40**
	39	**18**	**31**	**6**	**40**
	=1975=32×60+**55**	**18**	**31**	**6**	**40**
32	**55**	**18**	**31**	**6**	**40**
32	**55**	**18**	**31**	**6**	**40**

Carries are shown in red and *sexagesimal digits* in their final form in bold face. The procedure is mechanical and can be accurately described as a computer algorithm. You don't need to know the absolute location of the point. But one deduces that $\frac{1}{6561}$ is smaller than $\frac{1}{3600} = \frac{1}{60^2}$. Therefore Babylonians would have written the number with two leading zeros as 0.(0)(0)(32)(55)(18)(31)(6)(40) which is a sexagesimal equivalent of the modern decimal notation.

References

Literature

[1] Werner Buchholz. “Fingers or fists (the choice of decimal or binary representation”. *Communucations of the ACM* 02.12 (1959), pp. 3–11. DOI: 10.1145/368518.368529 (cit. on p. 3).

[2] Wilhelm Burger and Mark James Burge. *Digital Image Processing. An Algorithmic Introduction.* 3rd ed. Cham: Springer, 2022. DOI: 10.1007/978-3-031-05744-1 (cit. on p. 29).

[3] David P. Carlisle. *Packages in the ‘graphics’ bundle.* Mar. 5, 2021. URL: http://mirrors.ctan.org/macros/latex/required/graphics/grfguide.pdf (cit. on p. 7).

[4] Virgil Moring Faires. *Design of Machine Elements.* Originalausgabe 1920. The Macmillan Company, 1934 (cit. on p. 13).

[5] Simon Fear. *Publication quality tables in LaTeX.* Version v1.61803398. Jan. 14, 2020. URL: http://mirrors.ctan.org/macros/latex/contrib/booktabs/booktabs.pdf (cit. on p. 13).

[6] Nicholas J. Higham. *Handbook of Writing for the Mathematical Sciences.* 3rd ed. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 2020. URL: https://www.maths.manchester.ac.uk/~higham/hwms/ (cit. on p. 24).

[7] Donald E. Knuth. “Ancient Babylonian algorithms”. *Communucations of the ACM* 15.07 (1972), pp. 671–677. DOI: 10.1145/361454.361514 (cit. on p. 3).

[8] Helmut Kopka and Patrick William Daly. *Guide to LaTeX.* 4th ed. Tools and Techniques for Computer Typesetting. Reading, MA: Addison-Wesley, 2003 (cit. on p. 22).

[9] Nathaniel David Mermin. “What’s Wrong with these Equations?” *Physics Today* 42.10 (1989), pp. 9–11. DOI: 10.1063/1.2811173 (cit. on p. 24).

[10] Frank Mittelbach et al. *The amsmath package.* Version 2.17o. May 13, 2023. URL: http://mirrors.ctan.org/macros/latex/required/amsmath/amsmath.pdf (cit. on p. 25).

[11] Tobias Oetiker et al. *The Not So Short Introduction to LaTeX 2_ε. Or LaTeX 2_ε in 139 minutes.* Version 6.4. Mar. 9, 2021. URL: http://mirrors.ctan.org/info/lshort/english/lshort.pdf (cit. on p. 7).

[12] Herbert Voß. *Math mode.* Version 2.47. Jan. 30, 2014. URL: http://mirrors.ctan.org/obsolete/info/math/voss/mathmode/Mathmode.pdf (cit. on p. 22).

Media

[13] Marion Post Wolcott. *Natchez, Miss.* Library of Congress Prints and Photographs Division Washington, Farm Security Administration/Office of War Information Color Photographs. Aug. 1940. URL: https://www.loc.gov/pictures/item/2017877479/ (cit. on p. 6).

Online sources

[14] *Decimal Computer*. Feb. 24, 2010. URL: https://en.wikipedia.org/wiki/Decimal_computer (visited on 09/22/2024) (cit. on pp. 3, 4).

[15] IBM. *System 360. From Computers to Computer Systems*. Mar. 7, 2012. URL: https://www.ibm.com/ibm/history/ibm100/us/en/icons/system360/impacts/ (visited on 11/06/2023) (cit. on p. 8).

[16] *LaTeX/Lengths*. Aug. 4, 2018. URL: https://en.wikibooks.org/wiki/LaTeX/Lengths (visited on 11/06/2023) (cit. on p. 12).

[17] *Method of Complements*. Nov. 20, 2024. URL: https://en.wikipedia.org/wiki/Method_of_complements (visited on 09/11/2024) (cit. on p. 5).

[18] Martin Ruckert. *mmix.cs.hm.edu*. 2014. URL: https://www.mmix.cs.hm.edu/index.html (visited on 11/06/2014) (cit. on p. 4).

Measurement Box

— Check the size of this box in your printout! —

width = 100mm
height = 50mm

— Remove this page after printing and checking! —