Yet another LaTeX tutorial

Victor Eijkhout

August 2004

1 Document markup

If you are used to 'wysiwyg' (what you see is what you get) text processors, LaTeX may seem like a strange beast, primitive, and probably out-dated. While it is true that there is a long history behind TeX and LaTeX, and the ideas are indeed based on much more primitive technology than what we have these days, these ideas have regained surprising validity in recent times.

1.1 A little bit of history

Document markup dates back to the earliest days of computer typesetting. In those days, terminals were strictly character-based: they could only render mono-spaced built-in fonts. Graphics terminals were very expensive. (Some terminals could switch to a graphical character set, to get at least a semblance of graphics.) As a result, compositors had to key in text on a terminal – or using punched cards in even earlier days – and only saw the result when it would come out of the printer.

Any control of the layout, therefore, also had to be through character sequences. To set text in bold face, you may have had to surround it with `<B> .. the text .. </B>`. Doesn't that look like something you still encounter every day?

Such 'control sequences' had a second use: they could serve a template function, expanding to often used bits of text. For instance, you could imagine `$ADAM$` expanding to 'From our correspondent in Amsterdam:'.

LaTeX works exactly the same. There are command control sequences; for instance, you get bold type by specifying `\bf`, et cetera. There are also control sequences that expand to bits of text: you have to type `\LaTeX` to get the characters 'LATEX' plus the control codes for all that shifting up and down and changes in font size.

```
\TeX => T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX
\LaTeX => L\kern -.36em {\sbox \z@ T\vbox to\ht \z@ {\hbox
  {\check@mathfonts \fontsize \sf@size \z@ \math@fontsfalse
     \selectfont A} \vss }}\kern -.15em\TeX
```

1.2 Macro packages

The old typesetting systems were limited in their control sequences: they had a fixed repertoire of commands related to typesetting, and there usually was some mechanism to defining 'macros' with replacement text. Formally, a macro is a piece of the input that gets replaced by its definition text, which can be a combination of literal text and more macros or typesetting commands.

> An important feature of many composition programs is the ability to designate by suitable input instructions the use of specified formats. Previously stored sequences of commands or text replace the instructions, and the expanded input is then processed. In more sophisticated systems, formats may summon other formats, including themselves [”System/360 Text Processor Pagination/360, Application Description Manual,” Form No. GE20-0328, IBM Corp., White Plains, New York.].

That was the situation with commercial systems by manufacturers of typesetting equipment such as Linotype. Systems developed by (and for!) computer scientists, such Scribe or nroff/troff, were much more customizable. In fact, they sometimes would have the equivalent of a complete programming language on board. This makes it possible to take the basic language, and design a new language of commands on top of it. Such a repertoire of commands is called a macro package.

In our case, TEX is the basic package with the strange macro programming language, and LATEX is the macro package[1]. LATEX was designed for typesetting scientific articles and books: it offers a number of styles, each with slightly different commands (for instance, there are no chapters in the article style) and slightly different layout (books need a title page, articles merely a title on the first page of the text). Styles can also easily be customized. For different purposes (art books with fancy designs) it is often better to write new macros in TEX, rather than to bend the existing LATEX styles.

However, if you use an existing LATEX style, the whole of the underlying TEX programming language is still available, so many extensions to LATEX have been written. The best place to find them is through CTAN `http://wwww.ctan.org/`.

> **Exercise 1.** Discuss the difference between a macro and a function or procedure in a normal programming language. In a procedural language, looping is implemented with `goto` instructions. How would you do that in a macro language? Is there a difference in efficiency?

1.3 Logical markup

Macro packages are initially motivated as a labour-saving device: a macro abbreviates a commonly used sequence of commands. However, they have another important use: a well designed macro package lets you use commands that indicate the structure of a document rather than the formatting. This means that you would write `\section{Introduction}` and not worry about the layout. The layout would be determined by a statement elsewhere

1. In this tutorial I will say 'TEX' when a statement applies to the basic system, and 'LATEX' if it *only* applies to the macro package.

as to what macros to load[2]. In fact, you could take the same input and format it two different ways. This is convenient in cases such as an article being reprinted as part of a collection, or a book being written before the final design is commissioned.

In a well written document, there will be few explicit typesetting commands. Almost all macros should be of the type that indicates the structure, and any typesetting is taken care of in the definition of these. Further control of the layout of the document should be done through global parameter settings in the preamble.

2 The absolute basics of LaTeX

Here is the absolute minimum you need to know to use LaTeX.

2.1 Different kinds of characters

A TeX input file mostly contains characters that you want to typeset. That is, TeX passes them on from input to output without any action other than placement on the page and font choice. Now, in your text there can be commands of different sorts. So TeX has to know how to recognize commands. It does that by making a number of characters special. In this section you will learn which characters have special meaning.

- Anything that starts with a backslash is a command or 'control sequence'. A control sequence consists of the backslash and the following sequence of letters – no digits, no underscores allowed either – or one single non-letter character.
- Spaces at the beginning and end of a line are ignored. Multiple spaces count as one space.
- Spaces are also ignored after control sequences, so writing `\LaTeX is fun` comes out as 'LaTeXis fun'. To force a space there, write `\LaTeX{} is fun` or `\LaTeX\ is fun`. Spaces are *not* ignored after control symbols such as `\$`, but they are again after the 'control space' `\␣`[3].
- A single newline or return in the input file has no meaning, other than giving a space in the input. You can use newlines to improve legibility of the input. Two newlines (leading to one empty line) or more cause a paragraph to end. You can also attain this paragraph end by the `\par` command.
- Braces {,} are mostly used for delimiting the arguments of a control sequence. The other use is for grouping. Above you saw an example of the use of an empty group; similarly `\TeX{}ing is fun` comes out as 'TeXing is fun'.
- Letters, digits, and most punctuation can be typed normally. However, a bunch of characters mean something special to LaTeX: `%$&^_#~{}`. Here are their functions:
  `%` comment: anything to the end of line is ignored.
  `$,_,^` inline math (toggle), subscript, superscript. See section **??**.
  `&` column separator in tables.
  `~` nonbreaking space. (This is called an 'active character')
  `{}` Macro arguments and grouping.

2. Compare this to the use of CSS versus HTML in web pages.
3. The funny bucket character here is how we visualize the space character.

In order to type these characters, you need to precede them with a backslash, for instance `\%` to get '%'. This is called a 'control symbol'. Exception: use `$\backslash$` to get '\'.

- Some letters do not exist in all styles. As the most commonly encountered example, angle brackets `<>` do not exist in the roman text font (note that they are in the typewriter style here, in roman you would get '¡¿'), so you need to write, somewhat laboriously `$\langle$S$\rangle$` to get '$\langle$S$\rangle$'[4].

  Exercise 2. You read in a document 'This happens only in 90rest of the time it works fine.' What happened here? There are articles in print where the word 'From' has an upside down question mark in front of it. Try to think of an explanation.

2.2 LaTeX document structure

Every LaTeX document has the following structure:

```
\documentclass[ <class options> ]{ <class name> }
    <preamble>
\begin{document}
    <text>
\end{document}
```

Typical document classes are `article`, `report`, `book`, and `letter`. As you may expect, that last one has rather different commands from the others. The class options are optional; examples would be `a4paper`, `twoside`, or `11pt`.

The preamble is where additional packages get loaded, for instance

```
\usepackage{times}
```

switches the whole document to the Times Roman typeface. This is also the place to define new commands yourself (section **??**).

2.2.1 Title

To list title and author, a document would start with

```
\title{My story}
\author{B.C. Dull}
\date{2004} %leave this line out to get today's date
\maketitle
```

After the title of an article and such, there is often an abstract. This can be specified with

```
\begin{abstract}
... The abstract text ...
\end{abstract}
```

The stretch of input from `\begin` to `\end` is called an 'environment'; see section **??**.

4. That's what macros are good for.

2.2.2 Sectioning

The document text is usually segmented by calls

```
\section{This}
\subsection{That}
\subsection{The other}
\paragraph{one}
\subparagraph{two}
\chapter{Ho}
\part{Hum}
```

which all get numbered automatically. Chapters only exist in the `report` and `book` styles. Paragraphs and subparagraphs are not numbered. To prevent sections et cetera from being numbered, use `\section*{...}` and such[5].

2.2.3 Frontmatter, backmatter

You can use commands `\frontmatter`, `\mainmatter`, `\backmatter` – in `book` class only – to switch page numbering from roman to arabic, and, for the back matter, section numbering from numbers to letters.

2.3 Running LaTeX

With the last two sections you should know enough to write a LaTeX document. Now how do you get to see the final output? This takes basically two steps: formatting and viewing.

You need to know that TeX's original output format is slightly unusual. It is called 'DVI' for DeVice Independent. There are viewers for this format, but usually you need another step to print it.

Traditionally, you would run an executable called `latex` (or `tex`), which gives you a `dvi` file, which you then view with a previewer such as `xtex` or `xdvi`. To print this file, you would use

```
dvips -Pcmz foo.dvi -o foo.ps
```

to generate a `ps` file. This can be printed, or converted to `pdf`.

There are version of the `latex` executable that output to other formats, for instance `pdflatex` (there is also a `pdftex`) goes straight to `pdf`, which you can view with the Adobe Acrobat Reader, or `xpdf`. The big advantage of this approach is that you can get hyperlinks in your pdf file; see section **??**.

> **Exercise 3.** Set up a document that will have the answers to your homework exercises of the whole course.

5. This also makes the title not go into the table of contents. See section **??** on how to remedy that.

3 The TEX conceptual model of typesetting

In TEX, the question 'on what page does this character appear' is hard to answer. That is because TEX typesets all material for a page, sort of on a long scroll, before cutting a page off that scroll. That means that when a piece of text is set, you do not know if it falls before or after a certain page break.

A similar story holds for paragraph breaking, but the question on what line something occurs is not usually interesting.

4 Text elements

Here are the main elements that make up a LATEX document.

4.1 Large scale text structure

We already discussed sectioning commands in section **??**. Here are more major text elements in a LATEX document.

4.1.1 Input files

Use `\include{<file>}` to input a file beginning on a new page, and `\input` for just plain input. With

```
\includeonly{file1,file2}
```

you can save processing time – provided the files are `\include`d to begin with.

The `.tex` extension can usually be left off; because of the way TEX works, be careful with funny characters in the file name.

On Unix installations, input files can be in other directories, specified by the `TEXINPUTS` environment variable.

4.1.2 Environments

If a certain section of text needs to be treated in a different way from the surrounding text, it can be segmented off by

```
\begin{<environment name>}
... text ...
\end{<environment name>}
```

An environment defines a group, so you can have local definitions and changes of parameters.

Some predefined environments are

flushleft (flushright) for text that is left (right) aligned, but not right (left).
center for text that is centered.

quote, quotation for text that needs to be set apart, by indenting both margins. The `quote` environment is for single paragraphs, the `quotation` for multiple.

abstract for the abstract that starts an article, report, or book. In the report and book style it is set on a separate page. In the article style it is set in a smaller type size, and with indented margins.

verbatim see section **??**.

4.1.3 Verbatim text

As we have observed already, TEX has a number of special characters, which can be printed by prefixing them with a backslash, but that is a hassle. Good thing that there is a mechanism for printing input completely verbatim. For inline text, use `\verb+&###\text+` to get '`&###\text`'. The essential command here is `\verb`. Unlike with other commands that have arguments, the argument is not delimited by braces, but by two occurrences of a character that does not appear in the verbatim text. A plus sign is a popular choice. The `\verb*` variant makes spaces visible: `\verb*+{ }+` gives '`{␣}`'.

For longer verbatim text there is a `verbatim` environment. The `verbatim*` version prints each space as a ␣ symbol. To input whole files verbatim, use `\verbatiminput{file}`, which is defined in the `verbatim` package.

For TEXnical reasons, verbatim text can not appear in some locations such as footnotes or command definitions.

Exercise 4. Why does the `\verb` command not have its argument in braces?

4.1.4 Lists

Lists in LATEX are a special case of an environment; they are specified by

```
\begin{<list type>}
\item ...
\item ...
\end{<list type>}
```

The three main list types are unnumbered lists, `itemize`, numbered lists, `enumerate`, and definition or description lists, `description`.

In the case of a description list, it is mandatory to give the item label:

```
\begin{description}
\item[Do] A deer, a female deer.
\item[Re] A drop of golden sun.
...
\end{description}
```

You can give item labels in the other list types too.

Putting a list inside a list item will change the style of the item labels and numbers in a way that is dictated by the document class.

You can put a `\label` command after an item to be able to refer to the item number.

```
\begin{enumerate}
\item\label{first:item} One
\item Two comes after \ref{first:item}
\end{enumerate}
```

Output:

1. One
2. Two comes after **??**

This only makes sense with enumerate environments.

4.1.5 Tabbing

The `tabbing` environment is useful for setting pieces of text, such as source code, that use a small number of 'tab stops'. Tab stops (a term deriving from old mechanical typewriters) are locations on the line that one can 'tab to', no matter how much material is currently on the line.

Example:

```
\begin{tabbing}
The first line sets this: \=point;\\
the second jumps\>there
\end{tabbing}
```

Output:

```
The first line sets this: point;
the second jumps          there
```

The \= command in the first line defines a tab stop; in every subsequent line a \> command will jump to that position, if it has not been reached yet. There can be multiple tab stops, not necessarily defined in the same line, and tab stops can be redefined.

A more interesting case is where the tab stop is used before the line that defines it. For this case there is the `\kill` command, which prevents a line from being displayed. Example:

```
\begin{tabbing}
while \=\kill
do\>\{\\
\>$i_1\leftarrow{}$\=1\\
\>$\ldots$\>2\\
\>\}\\
while (1)
\end{tabbing}
```

Output:

```
do    {
      i_1 ← 1
      ...   2
      }
while (1)
```

4.1.6 Tabular material

The `tabular` environment generates a table. Tables are often placed independently of the text, at the top or bottom of the page; see section **??** for details. The table itself is generated by

```
\begin{tabular}{<alignment>}
... material ...
\end{tabular}
```

Each line of the table has items separated by characters, and `\\` at the end of each line but the last.

In its simplest form, the alignment directions are a combination of the letters `l,r,c`:

```
\begin{tabular}{rl}
"Philly" Joe & Jones\\ Dizzie & Gillespie\\ Art&Tatum
\end{tabular}
```

Output:

”Philly” Joe	Jones
Dizzie	Gillespie
Art	Tatum

Vertical rules are inserted by placing a `|` character in the alignment specification; horizontal lines you get from `\hline`.

```
\begin{tabular}{|r|rl|}
\hline
instrument&name&\\ \hline
drums: &"Philly" Joe & Jones\\
trumpet:& Dizzie & Gillespie\\
piano: &Art&Tatum\\ \hline
\end{tabular}
```

Output:

instrument	name	
drums:	”Philly” Joe	Jones
trumpet:	Dizzie	Gillespie
piano:	Art	Tatum

Some more tricks:

- In headings you often want to span one item over several columns. Use
  ```
  \begin{tabular}{|r|rl|}
  \hline
  instrument&\multicolumn{2}{|c|}{name}\\ \hline
  drums: &"Philly" Joe & Jones\\
  trumpet:& Dizzie & Gillespie\\
  piano: &Art&Tatum\\ \hline
  \end{tabular}
  ```
  Output:

instrument	name	
drums:	”Philly” Joe	Jones
trumpet:	Dizzie	Gillespie
piano:	Art	Tatum

- LaTeX inserts a standard amount of space between columns. You can override this with `@{<stuff>}`:
  ```
  \begin{tabular}{r@{.}l}
  2&75\\ 800&1
  \end{tabular}
  ```
  gives 2.75
  800.1
- A column specification of `p{<size>}` (where `<size>` is something like `5.08cm` or `2in`) makes the items in that column formatted as paragraphs with the width as specified.

4.1.7 Footnotes

Use the command `\footnote`. The numbering style is determined by the document class. The kinds of footnote to denote affiliation of the author of a paper and such (these often use asterisk symbols and such, even if footnotes are numbered in the rest of the document) are given by the command `\thanks`.

There are two common cases where want more detailed control over footnotes:

- You want to dictate the label yourself, for instance using the same label again (this often happens with author affiliations)
- You want to place a footnote in a table or such; LaTeX has trouble with that.

In such cases you can use `\footnotemark` to place the marker, and `\footnotetext` for the text. You can also set or change the `footnote` counter explicitly with counter functions (section **??**), or use

```
\footnote[<label>]{<text>}
```

where the label is used instead, and the counter is not increased.

4.1.8 Text boxes

Sometimes you want a small amount of text to behave like one or more paragraphs, except not as part of the main page. The main commands for that are

```
\parbox[pos]{width}{text}
\begin{minipage}[pos]{width}  text  \end{minipage}
```

The optional `pos` parameter specifies whether the top (`t`) or bottom (`b`) line of the box should align with surrounding text: top-aligned box of text:

```
Chapter 1. \parbox[t]{2in}{\slshape Introduction. First easy
  lessons. Exercises. More about things to come. Conclusions}
```

Output:

Chapter 1. *Introduction. First easy lessons. Exercises. More about things to come. Conclusions*

The default is a vertically centered position.

The `minipage` environment is meant for longer pieces of text; it can also handle other environments in the text.

The `\mbox` command is for text (or other objects) that need to stay on one line.

4.2 Minor text issues

4.2.1 Text styles

You can switch between roman (the style for this text), *italic* (also called 'cursive'), *slanted* (in some typefaces, italic and slanted may be identical), and **bold** with the commands `\texrm`, `\textit`, `\textsl`, and `\textbf` respectively, used as

```
Text is stated \textbf{boldly} or \textsl{with a slant}.
```

These combinations are not independent: nesting the commands can give you **bold *slanted*** text.

The above commands are mostly for short bits of text. See section **??** for commands to change font parameters in a more global manner.

If you are using italics for emphasis, consider using `\emph` instead, which works better, especially if you emphasize something in text that is already italic.

4.2.2 Fonts and typefaces

You already saw commands such as `\textrm` and `\textit` for switching from one type style to another. These commands hide a more complicated reality: LaTeX handles its fonts as combination of three parameters. These individual switches can be used inside a group, or as an environment:

```
{\ttfamily This is typewriter text}
\begin{mdseries}
 This text is set in medium weight.
\end{mdseries}
```

Here are the categories and possible values.

family roman, sans serif, typewriter type: `\rmfamily`, `\sffamily`, `\ttfamily`.
series medium and bold: `\mdseries`, `\bfseries`.
shape upright, italic, slanted, and small caps: `\upshape`, `\itshape`, `\slshape`, `\scshape`.

4.2.3 Comments

Anything from `%` to the end of line is ignored. For multiline comments, load either

```
\usepackage{comment}
```

or

```
\usepackage{verbatim}
```

and in both cases surround text with

```
\begin{comment}
to be ignored
\end{comment}
```

where the closing line *has to be* on a line of its own.

4.2.4 Hyphenation

Sometimes TEX has a hard time finding a breakpoint in a word. When you are fixing the final layout of a document, you can help it with `helico\-pter`. If TEX consistently breaks your name wrong, do

```
\hyphenation{Eijk-hout}
```

in the preamble.

This is not the mechanism for telling TEX about a new language; see section **??**.

To keep text together, write `\mbox{do not break}`. You could also write this with a non-breaking space as `do~not~break`. (See also section **??**.) It is a good idea to write

```
A~note on...
increase by~$1$.
```

to prevent single characters at the beginning of a line (first example), or the end of a line (second example). The second example could even give a line with a single character on it if it were to occur at the end of a paragraph.

4.2.5 Tilde

The tilde character has special meaning as a nonbreaking space; see section **??**. To get a tilde accent, use `\~`. To get a literal tilde, do `\~{}`, `$\sim$`, or `\char`\~`. If you need a tilde in URLs, consider using the `url` or `hyperref` package; see section **??**.

4.2.6 Accents

In basic TEX, accents are formed by putting a control symbol of that accent in front of the letter:

```
Sch\"on b\^et\'e
```

for ‘Schön bêté’. If you have an occasional foreign word in English text this works fine. However, if your terminal allows you to input accented characters, you can use them in LATEX with the `inputenc` package.

Standard TEX (or LATEX) does not understand Unicode encodings such as UTF-8.

4.2.7 Line/page breaking

In general, you should leave line and page breaking to TEX, at most adjusting parameters. However, should you really need it,
you can use the commands `\linebreak[<num>]` and `\pagebreak[<num>]`, where the number is `1,2,3,4`, with 4 the highest
urgency. There is also `\nolinebreak` and `\nopagebreak` with a similar urgency parameter.

In this last paragraph there was a `\linebreak` after 'need it'. You notice that TEX still tried to fill out to the right margin, with ugly consequences. After 'highest' there was a `\newline`, which really breaks then and there. Similarly, there is `\newpage`.

There is also `\nolinebreak` and `\nopagebreak`, both with optional numerical parameter, to discourage breaking.

4.2.8 Manual spacing

Most of the time you should leave spacing decisions to LATEX, but for custom designs it is good to know the commands.

```
\hspace{1cm} \hspace*{1in} \hspace{\fill}
\vspace{1cm} \vspace*{1in} \vspace{\fill}
```

- The ⋆-variants give space that does not disappear at the beginning or end of a line (for horizontal) or page (vertical).
- A space of size `\fill` is infinite: this means it will stretch to take up however much space is needed to the end of the line or page.

4.2.9 Drawing lines

Let us get one thing out of the way: underlining is a typewriter way of emphasizing text. It looks bad in typeset text, and using italics or slanted text is a much better way. Use `\emph`.

Lines can be used a typographical decorations, for instance drawn between the regular text and the footnotes on a page, or as part of chapter headings. The command is

```
\rule[lift]{width}{height}
```

Example

```
1\ \rule{2cm}{\fboxrule}\ The title
```

Output:

1 ___________ The title

You can draw a whole box around text: `\fbox{text}` gives $\boxed{\text{text}}$. The thickness of the line is `\fboxrule`.

4.2.10 Horizontal and vertical mode

TEX is in horizontal or vertical mode while it is processing[6]. In horizontal mode, elements – typically letters – are aligned next to each other; in vertical mode elements are stacked on

6. The story is actually more complicated; for the whole truth see the notes about TEX.

top of one another. Most of the time you do not have to worry about that. When TEX sees text, it switches to horizontal mode, and LATEX environments will briefly switch to vertical mode so that they start on a new line.

In certain cases you want to force vertical mode; for that you can use `\par`. You can force things into a line with `\mbox` (section **??**). In rare cases, `\leavevmode`.

5 Tables and figures

Tables and figures are objects that typically do not appear in the middle of the text. At the very least they induce a paragraph break, and often they are placed at the top or bottom of a page. Also, some publishers' styles demand that a document have a list of tables and a list of figures. LATEX deals with this by having environments

```
\begin{<table or figure>}[placement]
... table or figure material ...
\caption{Caption text}\label{tabfig:label}
\end{<table or figure>}
```

In this,

- The 'placement' specifier is a combination of the letters `htbp` for 'here', 'top', 'bottom', and 'page', telling LATEX where to place the material, if possible. Suppose a placement of `[ht]` is given, then the material is placed 'right here', unless there is not enough space on the page, in which case it will be placed on top of the page (this one or the next).
- Table material is given by a `tabular` environment; see section **??**.
- Figure material needs some extra mechanism, typically provided by another package; see section **??**.
- The caption goes into the list of tables/figures.
- The label will refer to the number, which is automatically generated.

The list of tables/figures is generated by the command `\listoftables` or `\listoffigures`.

6 Math

TEX was designed by a computer scientist to typeset some books with lots of mathematics. As a result, TEX, and with it LATEX's, math capabilities are far better than those of other typesetters.

6.1 Math mode

You can not just write formulas in the middle of the text. You have to surround them with `$<formula>$` or `\(<stuff>\)` for inline formulas, or

```
\begin{displaymath} ... \end{displaymath}
\[ ... \]
```

for unnumbered and

```
\begin{equation} ... \end{equation}
```
for numbered displayed equations respectively. You can refer to an equation number by including a `\label` statement.

In math mode, all sorts of rules for text typesetting are changed. For instance, all letters are considered variables, and set italic: `$a$` gives 'a'. Roman text is either for names of functions, for which there are control sequences – `\sin(x)` gives '$\sin(x)$' – or for connecting text, which has to be marked as such:
```
\forall x \in \mathbf{R}
\quad \mathrm{(sufficiently large)} \quad: \qquad x>5
```
Output:

$$\forall x \in \mathbf{R} \quad \mathrm{(sufficiently large)} \quad: \qquad x>5 \tag{1}$$

A formula is limited to one line; if you want to break it, or if you need several formulas vertically after one another, you have to do it yourself. The `eqnarray` environment is useful here. It acts as a three-column alignment.
```
\begin{eqnarray}
\sin x&=&x-\frac{x^3}{3!}+\frac{x^5}{5!}- \nonumber \\
      &&{}-\frac{x^7}{7!}+\cdots
\end{eqnarray}
```
Output:

$$\begin{aligned} \sin x &= x-\frac{x^3}{3!}+\frac{x^5}{5!}- \\ &\quad {}-\frac{x^7}{7!}+\cdots \qquad (2) \end{aligned}$$

Note the use of `\nonumber` here; with the `eqnarray*` all lines would be unnumbered by default.

In AMS LaTeX there is an `align` environment which looks better than `eqnarray`.

6.2 Super and subscripts

In math mode, the character ^ denotes a superscript, and _ denotes a subscript: `x_i^2` is x_i^2. (Outside of math mode these characters give an error.) Sub and superscripts of more than one character have to be grouped.

6.3 Grouping

Grouping, in math mode as outside, is done with braces: `x_{i-1}^{n^2}` looks like $x_{i-1}^{n^2}$.

6.4 Display math vs inline

Math looks different when used inline in a paragraph from that used as display math. This is mostly clear for operators with 'limits':

text mode: $\sum_{i=1}^\infty$ $\displaystyle \mathrm{displaymode:} \sum_{i=1}^\infty$

6.5 Delimiters, matrices

Delimiters are `()[]\{\}`. You can prefix them with `\big`, `\Big` and such, but TEX can resize them automatically:

```
\left( \frac{1}{1-x^2} \right)
\left\{ \begin{array}{ccc}
    \mathrm{(a)}&\Rightarrow&x>0\\
    \mathrm{(b)}&\Rightarrow&x=0\\
    \mathrm{(c)}&\Rightarrow&x<0
        \end{array} \right.
```

Output:

$$\left(\frac{1}{1-x^2} \right)\left\{ \begin{array}{ccc} \mathrm{(a)}&\Rightarrow&x>0\\ \mathrm{(b)}&\Rightarrow&x=0\\ \mathrm{(c)}&\Rightarrow&x<0 \end{array} \right. \tag{3}$$

Note that with `\right.` you get a omitted right delimiter.

In the above example you also saw the `array` environment, which can be used for anything tabular in math mode, in particular matrices. Here is a good example of a matrix. Note the different kinds of dots:

```
 A = \left( \begin{array}{cccccc}
       a_{11}&0&&\ldots&0&a_{1n}\\
       &a_{22}&0&\ldots&0&a_{2n}\\
       &&\ddots&\ddots&\vdots&\vdots\\
       &&&a_{n-2n-2}&0&a_{n-2n}\\
       &\emptyset&&&a_{n-1n-1}&a_{n-1n}\\
       &&&&&a_{nn}
     \end{array} \right)
```

Output:

$$A = \left(\begin{array}{cccccc} a_{11}&0&&\ldots&0&a_{1n}\\ &a_{22}&0&\ldots&0&a_{2n}\\ &&\ddots&\ddots&\vdots&\vdots\\ &&&a_{n-2n-2}&0&a_{n-2n}\\ &\emptyset&&&a_{n-1n-1}&a_{n-1n}\\ &&&&&a_{nn} \end{array} \right) \tag{4}$$

6.6 There is more

See a good book for the whole repertoire of symbols. If what LATEX has is not enough, you can also get AMS LATEX, which has even more fonts and tricky constructs.

7 References

7.1 Referring to document parts

One of the hard things in traditional typesetting is to keep references such as ‘see also section 3’ in sync with the text. This is very easy in LATEX. You write

```
\section{Results}\label{section:results}
```

after which you can use this as

```
see also section~\ref{section:results}
on page~\pageref{section:results}.
```

The `\label` command can appear after headings, or in general every time some counter has been increased, whether that's a section heading or a formula number.

LaTeX implements this trick by writing the information to an auxiliary file – it has extension `.aux` – and reading it in next run. This means that a LaTeX document usually has to be typeset twice for all references to be both defined and correct. You get a reminder after the first run if a second one is needed, or if there are missing or duplicately defined labels.

> **Exercise 5.** A document with references usually takes two passes to get right. Explain why a table of contents can increase this number to three.

7.2 Table of contents

Something that typically goes into the front or back matter is the table of contents. This gets inserted automatically by the command `\tableofcontents`. No other actions required. You can add your own material to the contents with `\addcontentsline` or `\addtocontents`.

7.3 Bibliography references

Another kind of the reference is that to external bibliographies. This needs a bit more work.

- You write `\cite{Knuth:1978}` where you want the citation.
- At the end of your document you write

  ```
  \bibliographystyle{plain}
  \bibliography{cs}
  ```

  to get the bibliography included.
- The bibliography references have to be defined in a file `cs.bib`.
- After running LaTeX once, you need to invoke `bibtex <yourfile>`, which creates another auxiliary file, this time with `.bbl` extension, and run LaTeX once or twice more.

The bibliography files have a syntax of their own, but you can figure that out from looking at some examples.

7.4 Index

Finally, a document can have an index. For this you need to have a statement `\usepackage{makeidx}` in the preamble, `\printindex` wherever you want the index, and commands `\index{<some term>}` throughout your document. Additionally, as with `bibtex`, you need to run the program `makeindex` to generate the external `.ind` file.

Further indexing commands: `\index{name!sub}` for subentry; `\index{foo@\textit{foo})` for sorting under 'foo' but formatted differently.

8 Some TEXnical issues

8.1 Commands inside other commands

For deep technical reasons you can get completely incomprehensible error messages by writing things like

```
\section{My first section \footnote{and not my last}}
```

Remedy that by writing

```
\section{My first section \protect\footnote{and not my last}}
```

8.2 Grouping

Most modern programming languages have a block structure of some sort, where variables can be declared inside a block, and are no longer known after the block ends. TEX has a stronger mechanism, where assignments to a variable made inside a block are reverted at the end of that block.

In LATEX you notice that only `\newcommand` and `\newenvironment` declarations are local; `\newcounter`s are global, as are `\setcounter` assignments. However, `\savebox` assignments are local.

9 Customizing LATEX

LATEX offers a number of tools (on top of the possibility of doing straight TEX programming) for customizing your document. The easiest customization is to change already defined parameters. However, you can also define new commands and environments of your own.

In fact, several of the customization we will see in this section are not part of standard LATEX, but have been written by other users. If they do not come with your installation, you can download them from the Central TEX Archive Network; see section **??**.

9.1 Page layout

9.1.1 Layout parameters

Page layout is controlled by parameters such as `\textheight`, `\textwidth`, `\topmargin` (distance to the running head, not to the first text line), and `\odd/evensidemargin` (distance to the 'spine' of the document). These are set with commands like

```
\setlength{\textwidth}{10in}
\addtolength{\oddsidemargin}{-1cm}
```

Some lengths are 'rubber length'

```
\setlength{\parskip}{10pt plus 3pt minus 2pp}
```

9.1.2 Page styles

Use the commands

```
\pagestyle{<style>}
```

and

```
\thispagestyle{<style>}
```

to change the style of all pages or one page. Available styles are `empty` (no page numbers), `plain` (the default), and `headings` (page numbers and running headers). See also section **??** for many more options.

For two-sided printing, use the `twoside` option for the document class.

> **Exercise 6.** Take a look at the headers and footers in Oetiker's 'Not so short introduction' and 'TEX by Topic' (the LATEX and TEX part of the handout). Can you find a reason to prefer one over the other from a point of usability? In both books, what is the rationale behind the header on the odd pages? See in particular page 35 of the former and 77 of the latter. Do you agree with this design?

9.1.3 Running page headers

The `headings` page style (section **??**) uses running heads that can change through the document. For instance it would have chapter titles in the left page head and section titles in the right head. You can achieve this effect yourself by using the `myheadings` page style, and using the

```
\markright{<right head>}
\markboth{<left>}{<right>}
```

You have access to these texts as `\rightmark` and `\leftmark`; this is needed in the `fancyhdr` style.

9.1.4 Multicolumn text

Load

```
\usepackage{multicol}
```

and write

```
\begin{multicol}{3}
text in three column mode
\end{multicol}
```

9.2 New commands

You can define your own commands in LATEX. As example of a a simple command, consider an often used piece of text

```
\newcommand{\IncrByOne}{increased by~$1$}
```

The replacement text can have parameters:

```
\newcommand{\IncrDecrBy}[2]{#1creased by~$#2$}
```

In this definition, the number of arguments is listed after the command name: `[2]`, and occurrences of the arguments in the replacement text are indicated by `#1`, `#2` etc. Example calls: `\IncrDecrBy{in}{5}`, `\IncrDecrBy{de}{2}`.

The declaration of a new command can specify an optional argument. If we define

```
\newcommand{\IncrDecrBy}[2][in]{#1creased by~$#2$}
```

the `[in]` specification means that the first argument is optional (only the first argument can ever be optional) with a default value of `in`. Example calls:

```
\newcommand{\IncrDecrBy}[2][in]{#1creased by~$#2$}
\IncrDecrBy[de]{1}, \IncrDecrBy{5}.
```

Output:

> decreased by 1, increased by 5.

To redefine an existing command, use `\renewcommand`.

9.3 New environments

It is possible to define environments, by specifying the commands to be used at their start and end:

```
\newenvironment{example}%
  {\begin{quote}\textbf{Example.}}%
  {\end{quote}}
```

which, used as `\begin{example}...\end{example}` gives a `quote` environment that starts with the word 'Example' in bold face. While defining that environment does not save a lot of typing, it is a good idea nevertheless from a point of view of logical markup. Using the example environment throughout ensures a uniform layout, and makes design changes easy if you ever change your mind.

Special case: defining mathematical statements with

```
\newtheorem{majorfact}{Theorem}
\newtheorem{minorfact}[majorfact]{Lemma}
\begin{minorfact}Small fact\end{minorfact}
\begin{majorfact}Big fact\end{majorfact}
```

giving

> **Lemma 1** *Small fact*

> **Theorem 2** *Big fact*

The optional argument in the definition of `lemma` makes it use the `theorem` counter.

> **Exercise 7.** Why does this not work:
> ```
> \newenvironment{examplecode}%
>  {\textbf{Example code.}\begin{verbatim}}{\end{verbatimm}}
> ```

Exercise 8. Write macros for homework typesetting; make one master document that will contain all your homework throughout this course.

1. Define an environment `exercise` so that
   ```
   \begin{exercise}
   My answer is...
   \end{exercise}
   ```
   gives

   Problem 5. My answer is. . .

   The counter should be incremented automatically. List your solution in your answer, and find a way that the listing is guaranteed to be the code you actually use.
2. Write a macro `\Homework` that will go to a new page, and output

   Answers to the exercises for chapter 3

   at the top of the page. The `exercise` environment should now take the question as argument:
   ```
   \begin{exercise}{Here you paraphrase the question that was asked}
   My answer is...
   \end{exercise}
   ```
   and this outputs

   Problem 3.8 *Here you paraphrase the question that was asked*
   My answer is. . .

   (Hint: read the section on text boxes. Also be sure to use `\par` to get LaTeX to go to a new line.) Allow for the question to be more than one line long. Unfortunately you can not get verbatim text in the question. Find a way around that.

9.4 Counters

LaTeX has a number of counters defined, for sections and such. You can define your own counters too. Here are the main commands:

create A new counter is created with
```
\newcounter{<name>}[<other counter>]
```
where the name does *not* have a backslash. The optional `other counter` indicates a counter (such as `chapter`) that resets the new counter every time it is increased. (To do this reset for an already existing counter, check out the `chngcntr` package.)

change values A counter can be explicitly set or changed as
```
\setcounter{<name>}{<value>}
\addtocounter{<name>}{<value>}
```
The command `\refstepcounter` also make the new value the target for a `\label` command.

use To get a counter value numerically, use `\value`. To print the value, use
```
\arabic{<name>}, \roman{<name>}, \Roman{<name>}
```
et cetera.

9.5 Lengths

Parameters such as `\textwidth` (section **??**) are called 'lengths'. You can define your own with

```
\newlength{\mylength}
\setlength{\mylength}{5in}
```

These lengths can be used in horizontal or vertical space commands (section **??**) for your own designs.

9.6 The syntax of `\new...` commands

Have you noticed by now that the name you define starts with a backslash in `\newcommand` and `\newlength`, but not in `\newenvironment` or `\newcounter`? Confusing.

10 Extensions to LaTeX

LaTeX may be a package on top of TeX, but that doesn't mean that the programming power of TeX is no longer available. Thus, many people have written small or large extensions to be loaded in addition to LaTeX. We will discuss a couple of popular ones here, but first we'll see how you can find them.

10.1 How to find a package, how to use it

Packages are typically loaded in the file preamble with

```
\usepackage{pack1,pack2,...}
```

(These course notes load about a dozen packages.)

Many popular packages are already part of the standard LaTeX distribution, but you will have to search to find where they are stored on your computer. Make a document that uses a common package, say `fancyhdr`, and see in the log output on the screen or in the log file where the file is loaded from. A typical location is `/usr/share/texmf/...`. With a bit of searching you can also find[7] the documentation, which can be a `dvi`, `ps`, or `pdf` file.

If you have heard of a package and it is not on your system, go to the 'Comprehensive TeX Archive Network' (CTAN for short) and download it from there: `http://www.ctan.org/`.

10.2 Fancy page headers and footers

The `fancyhdr`[8] package provides customized headers and footers. The simple interface is

```
\lhead{<text>}  \chead{<text>}  \rhead{<text>}
```

7. For instance using the Unix command '`find`'.
8. This supersedes the `fancyheadings` package.

for specifying text to get left, center, and right in the header. Likewise `\lfoot` and such for the footer.

This is assuming that all pages are the same. If you format for two-sided printing (section **??**), you can specify different text for odd and even pages:

```
\fancyhead[LE,RO]{<text>}
```

for text that is Left on Even and Right on Odd pages. Typically, you specify text for `[LE,RO]` and `[RE,LO]`, for instance

```
\fancyhead[EL,OR]{\textsl{\rightmark}}
```

(see section **??**).

10.3 Pdf file generation

Making beautiful pdf documents, complete with hyperlinks and table of contents, from your LaTeX files is simplicity itself. Insert

```
\usepackage[pdftex]{hyperref}
```

in the preamble, and format with `pdflatex`. That's it. Do see section **??** about including pictures.

10.4 Graphics

Because of TeX's ancient origins – and its desire to be machine-independent – graphics is not standard, and frankly a bit of a hassle. The basic TeX system knows nothing about graphics, it just keeps some space open. An extension mechanism ('specials') then puts information in the output file that the printer driver can use to place graphics. With `pdflatex` this situation has become a bit less messy: now any graphics go straight into the pdf output file.

10.4.1 The `picture` environment

There is a way to generate graphics from inside LaTeX, using some graphics fonts rather than a full drawing mode. While primitive and limited, the `picture` environment has two advantages:

- It is easier to get the fonts for labels to be the same as the text font.
- Since it involves explicit drawing instructions, you can automatically draw bar charts and such.

10.4.2 Including external graphics

Most of the time, you will have graphics to include that come from some drawing package. Using the `graphicx` package, you write

```
\includegraphics[key=value,...]{<file name>}
```

where the file name can refer to any format, but if you use pdflatex, Postscript can not be used; if your picture is in Postscript, you can convert it with `ps2pdf`.

Commands such as `\includegraphics`, as well as similar commands in other packages, leave space in your document for the graphic. Now you have to be careful: you can not leave space for a 3 inch picture, an inch from the bottom of the page. Here are two approaches for placing a picture:

- Just place it where you want it, and if it falls on a page break, deal with it later by moving it.
- Put the figure in a floating figure object (section **??**) and let LaTeX sort out the placement.

You can also have text wrap around a figure, by using the `wrapfig` package.

There is a package `color` for colour output.

10.5 Other languages than English

The fact that TeX and LaTeX were written by Americans becomes obvious in a couple of places.

- Various typographical conventions are geared towards American English.
- Words like ‘Chapter’ are the default in the style files[9].

To address this and make LaTeX easier to use with other languages, there is a package `babel`.

9. They used to be hard-wired, so the situation is improved.

Contents