

L^AT_EX

Iain Rendall

Chapter 1

Session 1

1.1 Document Structure and Document Classes

Every \LaTeX document can be divided into a *preamble*, *body* and *postamble*. The simplest preamble consists of a declaration of the document *class* and issues the `\begin{document}` instruction. For example

```
\documentclass{article}
\begin{document}
```

The body is terminated by the postamble, which apart from exceptional cases consists simply of the `\end{document}` instruction.

There are a number of standard \LaTeX document classes and various features of the most commonly used are summarised below:

The **article** and **report** classes are very similar – the main differences are that the **report** class creates a title page by default, and also starts each section on a new page.

Each of these classes allows the use of the `\maketitle` command – title and author may be set and an abstract defined as follows:

```
\title{Title Text}
\author{Author Name}
\abstract{Abstract Text}
```

which should be placed in the preamble (preceding `\end{document}`). The command `\maketitle` should be placed at the start of the body (immediately after `\begin{document}`).

The **book** class allows creation of a document with divisions at the level of *part* and *chapter*, which are also available in the **report** class, but which are not available in the **article** class.

The **letter** class is also available, which defines particular elements of a formal letter, including signature, address and closing text.

All of these classes can take options – most usefully to specify the size of text to be used and whether the output is to be two-sided (with margins suitable for binding) – for example,

```
\documentclass[10pt, a4paper, twoside]{book}
```

1.2 Dimensions and Space in L^AT_EX Documents

Dimensions in L^AT_EX can be measured in *points*, *centimetres* or *inches*. These are abbreviated as `pt`, `cm` and `in` respectively. Also the units `em`, `en` and `ex` can be used in a document. An `em` is the width of a lower-case m, an `en` is the width of a lower-case n. An `ex` is the height of a lower-case x.

There are several methods available to make space in a document. First of all, the line-break instruction `\` takes an optional argument giving a vertical distance to place after the line-break, for example

```
\[1.5ex]
```

places an additional 1.5 `ex` vertical space after the line-break. Also, `\`* makes a new line without starting a new paragraph. `\linebreak[n]` produces n line-breaks ($0 \leq n \leq 4$). Paragraph indentation can be controlled with `\indent` and `\noindent`.

Horizontal and vertical space may also be inserted using the `\hspace` and `\vspace` commands. Each of these accepts an argument giving the amount of space. Variant forms `\hspace*` and `\vspace*` force insertion of a space (the standard versions do not add space in certain circumstances, for example a vertical space would not appear after a new page). `\vspace*{10pt}` forces insertion of a ten point vertical space, for instance.

Space can also be used to advantage in certain mathematical contexts, to improve appearance: for example

$$\int_0^\pi \sin x \, dx \quad \text{is better than} \quad \int_0^\pi \sin x dx$$

and the first is produced with `\int_0^\pi{\sin x, dx}`. `\,` is useful for creation of small horizontal spaces in general.

Another useful way of creating space is the *inter-word* space. This is useful for instance for names given with initials: `J.~D.~Brown` produces J. D. Brown while without the `~` it appears as J. D. Brown, which assumes the end-of-sentence space, resulting in a poor appearance. Using the inter-word space also prevents a line-break.

There are a number of important *lengths* defined for a page in a L^AT_EX document, summarised as follows –

<code>\baselineskip:</code>	default vertical distance between lines in a paragraph
<code>\parindent:</code>	default paragraph indentation
<code>\parskip:</code>	additional extra vertical space between paragraphs
<code>\topmargin:</code>	top margin of page
<code>\oddsidemargin:</code>	inner margin on odd-numbered pages
<code>\evensidemargin:</code>	inner margin on even-numbered pages

Each of these can be changed using the `\setlength` command, for example `\setlength{\parindent}{0pt}` sets zero paragraph indentation.

It may be useful to allow a flexible value for `\parskip`, perhaps to allow ‘floats’ to appear in a better position on the page. This is done with, for example, `\setlength{\parskip}{1ex plus 0.5ex minus 0.2ex}`.

1.3 Page and Section Numbering, Titles, Headers

Page numbering is affected by the `\pagestyle` and `\pagenumbering` commands.

`\pagestyle{option}` sets the page style as follows, according to the value of `option`:

plain:	default centre-bottom number
empty:	no page numbering
headings:	running headings taken from sectioning commands
myheadings:	running headings specified by <code>\markright</code> for example

It should be noted that `\thispagestyle` affects a single page – `\thispagestyle{empty}` prevents a page number appearing for instance.

The style of numbering can be set using `\pagenumbering{option}` where `option` is one of the following:

arabic:	Arabic numerals
roman:	lower-case Roman numerals
Roman:	upper-case Roman numerals
alph:	lower-case letters
Alph:	upper-case letters

By default numbering in an abstract is in lower-case Roman numerals, while the body of the text is numbered with Arabic numerals. If necessary the page number can be adjusted using

```
\setcounter{\pagenumber}{page-number}
```

A range of sectioning commands are available in L^AT_EX. These are as follows:

```
\part
\chapter
\section
\subsection
\subsubsection
\paragraph
\subparagraph
```

All of these accept one optional and one mandatory argument. The mandatory argument is the full section title, while the optional argument is the title appearing in the Table of Contents. Note that `\part` and `\chapter` are only available in the `book` and `report` classes. In addition, further sectioning commands may be available, depending on the environment, such as `\theorem` or `\proposition`. All the commands are in the same form, for example

```
\section[Homology]{Homology of Homogeneous Spaces of Lie Groups}
```

creates a section with the given full title and the short title in the Table of Contents. The section number can be adjusted with

```
\setcounter{\section}{section-number}.
```

As already mentioned, the standard sectioning commands are used to define the title text and the text appearing in the Table of Contents. However, variant versions of the commands are available which do not set the text appearing in the Table of Contents. As an example

```
\subsection*{Continuous Deformation}
```

creates an un-numbered section with no Table of Contents entry.

Also as mentioned earlier, in the `headings` page style, running headers are taken from titles specified in sectioning commands. In the `myheadings` style, however, the running headers are taken from text specified using `\markleft`, `\markright` or `\markboth`. These can be used to set the running headers on just the left page, just the right page, or both. For instance

```
\markboth{POSTSCRIPT}{Further Comments}
```

produces the text `POSTSCRIPT` as the left page header, and `Further Comments` as the right page header.

The Table of Contents is produced using the `\tableofcontents` command, usually immediately after `\maketitle`.

In connection with sectioning, it is usual to have a blank page between sections. In fact, it is often the case that sections begin on an odd-numbered page (with either one or two blank pages preceding). A single blank page is created using the instruction `\clearpage`. If the instruction `\cleardoublepage` is used instead, one or two blank pages will be inserted so that the next section does start on an odd-numbered page.

1.4 Character Styles

Character styles are dependent partly on the fonts specified in the document, but all fonts should have bold and italic available. Text can be emphasised using the `\emph` command. All text within its scope is emphasised, and by default that means that it is italicised. For instance,

The following word is `\emph{emphasised}`.

Font shapes can be specified as follows –

```
\textup{text}: upright font (default)
\textit{text}: italic font
\textsl{text}: slanted font
\textsc{text}: small capitals font
\textmd{text}: medium series font (default)
\textbf{text}: boldface font
\textrm{text}: Roman (default)
\textsf{text}: sans serif
\texttt{text}: typewriter (fixed width)
```

These commands can be combined, for instance with

```
\textit{\textbf{text}}
```

which produces bold italic text within Roman text and bold text within italic text (the processor defaults to simple bold in an italic context).

Another useful way of changing font is to specify a font using the `\newfont` command. This accepts font names – for example `cmr10` for the Computer Modern Roman font. There is no need to use this for standard fonts, but it may be useful for other fonts, such as a script font or Cyrillic. For instance

```
\newfont{\ms}{rsfs10}
```

allows use of the script mathematical font `rsfs10` with, for example, `{\ms S}` to produce a script S.

Font size in \LaTeX is controlled using font size commands with suggestive names: for example

```
{\tiny{text}} produces text.
```

The following table summarises the notional size of each font in points –

<code>\tiny:</code>	5
<code>\scriptsize:</code>	7
<code>\footnotesize:</code>	8
<code>\small:</code>	9
<code>\normalsize:</code>	10
<code>\large:</code>	12
<code>\Large:</code>	14
<code>\LARGE:</code>	18
<code>\huge:</code>	20
<code>\Huge:</code>	24

Text styles in mathematics mode are set similarly to text mode – available are `\mathrm`, `\mathbb`, `\mathbf` and `\mathcal`.

It may be necessary to adjust the size of a font in mathematics mode, for instance compare $\frac{a}{\sqrt{a+b}}$ and $\frac{a}{\sqrt{a+b}}$. The latter was produced with

```
\frac{\displaystyle a}{\displaystyle\sqrt{a+b}}.
```

`\displaystyle` produces larger text, while `\textstyle` produces smaller text.

1.5 Boxes and Minipages

L^AT_EX has a variety of *box* constructions, and these are summarised below –

`\makebox`, `\mbox`

The `\makebox` command creates a box containing the specified text, and the command takes optional width and position arguments. Its general format is `\makebox[width][position]{text}`

where the width is specified in any valid unit of measurement (e.g. points, centimetres) and where the position is one of

- c: centred (default)
- l: flush left
- r: flush right

`\mbox{text}` creates a box just wide enough to contain the text given by its argument.

`\framebox`, `\fbox`

The `\framebox` command is identical to the `\makebox` command, except that it draws a frame around the outside, for instance `\framebox{text}`. Optional arguments are the same as for `\makebox`. `\fbox` is the same as `\mbox` except that it draws a frame.

`\parbox`

A parbox is a box with contents rendered in paragraph mode. The form of the command is

`\parbox[position]{width}{text}`

and while the width and text are mandatory, the position is optional, and is one of `t` or `b`, aligning the parbox by its top or bottom line respectively. A parbox is intended for a small section of text – in particular, paragraphs should not be declared inside a parbox. If this is required, a minipage should be used instead.

A minipage is created as follows –

```
\begin{minipage}[position]{width}
text
\end{minipage}
```

The minipage environment is similar to a `\parbox` command, taking the same optional position argument (`t` or `b`) and mandatory width. The text, however, is not given as an argument, it is placed in the environment. Paragraph-making instructions and environments can be used inside a minipage. The following is an example minipage –

This is some example text placed in a minipage environment using L^AT_EX.

1.6 BIBTeX

BIBTeX is a companion to L^AT_EX and manages bibliographic references. Using BIBTeX makes it possible to store citation information only once, then a citation can be made in any document, formatted in the style specified.

Two commands should be included in a L^AT_EX document to invoke the BIBTeX bibliography. These are

```
\bibliography{bib1, bib2, ...} and  
\bibliographystyle{style}.
```

The first of these identifies which BIBTeX bibliographies L^AT_EX should use. These will be .bib files. The second sets the style of bibliography L^AT_EX should use – this can be, for example, `plain`, `amsplain`, `abstract`, and there are many other styles too. The `\cite` command should be used in the document to insert citations, with the citation name, for instance `\cite{coverings}`.

Finally, to create the required output, run L^AT_EX once, then run BIBTeX, then run L^AT_EX another twice.

BIBTeX entries are made in a standard format. This consists of a declaration of the type of entry (always beginning with the @ symbol), followed by a varied number of fields, dependent upon the entry type. By way of an example,

```
@BOOK{coverings,  
  author = "Malden, E. S.",  
  title = {{Branched Coverings of Singular Spaces}},  
  edition = "Second",  
  publisher = {Cambridge University Press},  
  address = "Cambridge",  
  year = 2007}
```

The text of a field may be surrounded by either double-quote marks or braces, as seen above, or in the case of a numeric entry, these may be omitted. Note that titles may be auto-capitalised in some styles. To prevent this, enclose the title in double braces.

The fields in an entry fall into three classes, *mandatory*, *optional* and *ignored*. Fields recognised by the standard bibliography styles are

Different entry types have varying combinations of these. For example `author`, `title`, `publisher` and `year` are mandatory for `BOOK`. For `MISC`, no field is mandatory, and optional fields are `author`, `title`, `howpublished`, `month`, `year`, `note` and `key`. Types include `article`, `book`, `inbook`, `inproceedings` and so on.

A useful command is the `@STRING` command, for instance

```
@STRING{JKT = "Journal of Knot Theory"} allows use of the abbreviation  
JKT in the BIBTeX file. The @COMMENT command can be used to enter com-  
ments.
```

1.7 Managing L^AT_EX Source Files

Many L^AT_EX users create monolithic source files containing the whole content of a document. It is more manageable if the document is broken up into sections, perhaps chapters, or perhaps smaller units. L^AT_EX has a mechanism for allowing this. A L^AT_EX file can load other L^AT_EX files using `\include`, for example,

```
\documentclass{article}
\begin{document}
\include{macros}
\include{chapter1}
\include{chapter2}
\include{chapter3}
\include{chapter4}
\include{chapter5}
\include{chapter6}
\end{document}
```

could be used as a ‘top-level’ file, including the source of the chapters. Those chapter L^AT_EX files may in turn include other source files, sections and subsections for example.

It is also convenient to define any macros to be used in the document in a separate file.

Chapter 2

Session 2

2.1 Introduction

Commands can be defined in two ways –

```
\newcommand    defines a new command
\renewcommand  redefines an existing command
```

A simple example, which produces an arrow with the label \cong above it, is

```
\newcommand{\isoarrow}{\stackrel{\cong}{\rightarrow}}
```

which produces the following: $\xrightarrow{\cong}$

Note that this command requires that L^AT_EX is operating in ‘mathematics’ mode (for instance, set with \$). It may appear natural to define

```
\newcommand{\isoarrow}{\stackrel{\cong}{\rightarrow}}
```

but this will not work correctly if the processor is already in mathematics mode. The proper way to deal with this in the command is to use `\ensuremath`, which switches to mathematics mode if necessary, but does nothing in mathematics mode. The resulting command definition would be

```
\newcommand{\isoarrow}{\ensuremath{\stackrel{\cong}{\rightarrow}}}
```

Commands can take arguments – the general format is

```
\newcommand{\commandname}[number of arguments]{command definition}
```

and each argument is represented by a number preceded by #. The first argument is #1, the second is #2, and so on.

For example, suppose we want a command `\nvector` to print (x_1, x_2, \dots, x_n) or similar. The single argument to the command will be the letter to be used:

```
\newcommand{\nvector}[1]{\ensuremath{(\#1_1, \#1_2, \dots, \#1_n)}}
```

allows (z_1, z_2, \dots, z_n) to be produced with `\nvector{z}`.

`\renewcommand` can be used to redefine an existing command. As a simple example, if it is required to have the text **References** instead of **Bibliography** when a bibliography is created in L^AT_EX then the following will suffice –

```
\renewcommand{\bibname}{References}
```

Note that while a command defined with `\newcommand` can take any number of arguments, a command redefined with `\renewcommand` must take the same number of arguments as the original.

`\newcommand` can be given a default value for an argument, for example

```
\newcommand{\examplecommand}[3][i]
```

defines a command accepting three arguments, with `i` as the default value of the first argument.

2.2 Lengths in L^AT_EX Macros

In a number of situations, it is necessary to manipulate lengths in L^AT_EX macros. These variables, like commands, have names beginning with a backslash character. The following commands are available to work with length variables –

<code>\newlength:</code>	define a new length variable
<code>\setlength:</code>	set the value of a length variable
<code>\addtolength:</code>	add to a length variable
<code>\settowidth:</code>	set a length variable to the width of specified text
<code>\settoheight:</code>	set a length variable to the height of specified text
<code>\settodepth:</code>	set a length variable to the descender size of specified text

If a new length variable is to be used, it must first be declared with `\newlength`, for instance,

```
\newlength{\parboxwidth}
```

creates a length variable called `\parboxwidth`.

A length variable can be assigned a value with `\setlength`, for example

```
\setlength{\parboxwidth}{440pt}
```

sets the value of `\parboxwidth` to 440 points.

As an example to show the use of these commands, consider creating a macro to place material inside a frame, with a legend at the left side, centred on the page –

$$(1) \quad \boxed{\oint f(z) dz = 0}$$

which can be achieved by defining

```
\newcommand{\labelmidframe}[3]{  
  \begin{center}  
    #1\hspace{#2}  
    \fbox{\ensuremath{\displaystyle#3}}  
  \end{center}  
}
```

and using `\labelmidframe{(1)}{20pt}{\oint f(z)\,dz = 0}`

Now suppose this macro is to be rewritten so that it is not the legend together with the frame which is centred in the page – suppose the legend is to be placed at the left and the frame is to be central in the page:

$$(1) \quad \boxed{\oint f(z) dz = 0}$$

This can be done by defining

```
\newsavebox{\legend}  
\newsavebox{\expression}  
\newlength{\legendwidth}  
\newlength{\expressionwidth}  
\newlength{\spacer}  
  
\newcommand{\labelframe}[2]{  
\savebox{\legend}{#1}  
\savebox{\expression}{\fbox{\ensuremath{\displaystyle#2}}}  
\settowidth{\legendwidth}{\usebox{\legend}}  
\settowidth{\expressionwidth}{\usebox{\expression}}  
\setlength{\spacer}{.5\textwidth}  
\addtolength{\spacer}{-\legendwidth}  
\addtolength{\spacer}{-.5\expressionwidth}  
\begin{flushleft}  
\usebox{\legend}\hspace{\spacer}\usebox{\expression}  
\end{flushleft}  
}
```

and this illustrates a number of useful techniques:

First, the commands `\newsavebox`, `\savebox` and `\usebox` are used. Material can be placed in a *savebox*. A savebox is a variable capable of storing typeset material. It has the advantage that it can be used to place the material on the page at any point, simply by referring to the savebox by name – for instance

```
\newsavebox{\localbox}  
\savebox{\localbox}{\textbf{\Large Local savebox}}  
\usebox{\localbox}
```

produces **Local savebox**

The `\newsavebox` command creates a savebox variable, the `\savebox` command stores the required data in the savebox, while `\usebox` places it on the page. In the above example, the required space to centralise the frame on the page is calculated based on the `\textwidth` variable, the width of the legend and the width of the frame. Note that `\addtolength` has to be used to subtract the necessary lengths – this is done by changing the sign. Note also that constant multipliers can be specified as in the above example.

Please be aware that spaces within macro definitions can affect output, and this can include end-of-line. A method for avoiding this is to terminate lines in macro definitions with a `%` symbol (this is used to mark a comment, and results in the rest of the line being ignored, including end-of-line).

2.3 if/else in L^AT_EX Macros

Macros can include branching (if/then/else decisions), which necessitates use of the `ifthen` package, included with `\usepackage{ifthen}`.

As a simple example of this, consider the following definition –

```
\newcommand{\tt}[2]{
\ifthenelse{\equal{#1}{b}}{
\textbb{#2}
}{
\ifthenelse{\equal{#1}{i}}{
\textit{#2}
}{
\ifthenelse{\equal{#1}{t}}{
\texttt{#2}
}{
}
}
```

Note the termination of each case with `}`.

This can be used to print text in bold, italic or typewriter –

```
\ts{b}{X} produces   X
\ts{i}{X} produces   X
\ts{t}{X} produces   X .
```

2.4 Loops in L^AT_EX Macros

It is possible to have loops in macros. As an example of this, suppose we want a macro to insert a space specified as a multiple of a given small unit space (e.g. 1 point). The macro will accept a single parameter giving the number of unit spaces. This can be achieved by defining

```
\newcounter{spcount}

\newcommand{\nspace}[1]{
\whiledo{\value{spcount}>0}
{\hspace*{1pt}\addtocounter{spcount}{-1}}
}
```

Then, for instance,

Text `\nspace{2}` with `\nspace{2}` space produces Text with space.

Text `\nspace{6}` with `\nspace{6}` space produces Text with space.

2.5 Exercises

2.5.1 Weekday

Write a macro (using the `ifthen` package) to print the day of the week (Monday, Tuesday, etc.) given a number –

`\weekday{1}` to print ‘Monday’, `\weekday{2}` to print ‘Tuesday’, and so on.

2.5.2 Bulleted Paragraph

Write a macro to ‘bullet’ a paragraph – this should place a bullet-point (easily obtained with `$(\bullet$)` at the left side of the page, with the paragraph indented by the required amount to separate it from the bullet-point, and filling the remaining width.

For example, `\bulletpar{text}`.

2.5.3 Bulleted Paragraph with Position Option

Modify the previous macro so that it accepts an optional parameter giving the alignment of the paragraph relative to the bullet-point: `t` for top-alignment, `c` for centre-alignment and `b` for bottom-alignment.

For example, `\bulletpar[t]{text}`.

Chapter 3

Session 3

3.1 Graphics in L^AT_EX

L^AT_EX provides an interface for inclusion of graphics in documents, using either the `graphics` or `graphicx` package. As with other packages, these can be used once a `\usepackage` instruction has been issued. These packages allow simple inclusion of graphics from external files in various formats, for instance JPEG or PNG files. They also allow scaling and rotation of graphics images.

Images are included with the `\includegraphics` instruction. This has different forms in the two packages – first, `graphics`:



was produced with
`\includegraphics{arnold.png}`



was produced with
`\hspace*{-12pt}\rotatebox{25}{\includegraphics{arnold.png}}`



was produced with
`\scalebox{.5}{\includegraphics{arnold.png}}`, and



was produced with
`\scalebox{.75}{\rotatebox{-10}{\includegraphics{arnold.png}}}`.

The simple `\includegraphics` command includes a graphic at its default (natural) size, but the size can be adjusted using a `\scalebox`. This accepts an optional second argument to specify a vertical scale factor – the first argument is the horizontal scale factor and if the second argument is not given, the vertical scale factor is set equal to the given horizontal scale factor. For instance



was produced with
`\begin{center}`
`\scalebox{1.2}{0.75}{\includegraphics{arnold.png}}`
`\end{center}`

A graphics image can also be scaled using the `\resizebox` command. This specifies the actual size of the image's bounding box, for instance

```
\resizebox{20mm}{10mm}{\includegraphics{image.jpg}}
```

sets the width to 20mm and the height to 10mm. If the aspect ratio of the image is to be preserved, one dimension can be specified while the other is specified with an exclamation mark, for instance

```
\resizebox{20mm}{!}{\includegraphics{image.jpg}}.
```

There is also an `\includegraphics*` version of the `\includegraphics` command, which clips the image to the size of the specified bounding box. This is done with

```
\includegraphics*[x1,y1][x2,y2]{file}
```

where $(x1, y1)$ and $(x2, y2)$ specify the coordinates of the bottom left and top right corner. Note that the bounding box can also be specified in the regular `\includegraphics` command, in the same way. By default, measurements are in points, but any valid unit can be used.

In the `graphicx` package, the command to include an image is of the form

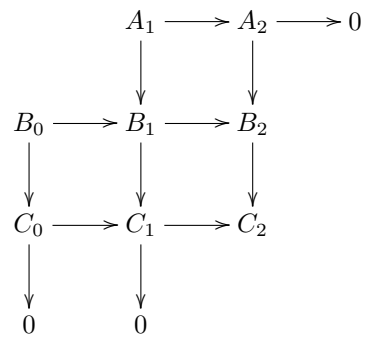
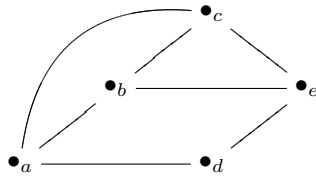
```
\includegraphics[key-value list]{file}
```

and again an `\includegraphics*` version is available. This is equivalent to including the `clip` key. The key-value list is a list of key=value pairs, separated by commas. Keys which are boolean are specified as true by inclusion of the key, false otherwise. The following gives a summary of the main keys:

bb	This specifies the bounding box of the image. This is done by giving four values stating the coordinates of the left-bottom, left-top, right-bottom and right-top. <code>[bb=10 10 120 120]</code> for example. Measurements are in points by default.
viewport	This specifies an area to be taken from within the bounding box to produce the image. It is also specified with four arguments, like bb , but the position is relative to the bottom-left of the bounding box.
trim	This specifies amounts to trim from the left, bottom, right and top of the bounding box, e.g. <code>[trim=1 1 1 1]</code> .
angle	Angle of rotation, counter-clockwise, in degrees.
origin	This specifies the centre of a rotation in terms of standard reference positions, lb, lB, lc, lt, b, c, t, rb, rB, rc, rt.
width	Required width.
height	Required height.
scale	Scale factor.
clip	Boolean value (key present/absent) specifying whether the image is to be clipped to the bounding box size.

3.2 Exercises

Write suitable code to produce the following diagrams –



L^AT_EX

Iain Rendall

1 METAPOST

METAPOST is software capable of producing graphics as *Encapsulated PostScript* and it is based on METAFONT, created by Donald Knuth as an adjunct to T_EX for the purpose of “creating entire families of fonts from a set of dimensional parameters and outline descriptions”. METAPOST was created some ten years later by John Hobby, whose aim was to produce a graphics language based on METAFONT, but producing PostScript output and allowing the inclusion of typeset material rendered by T_EX or L^AT_EX.

Encapsulated PostScript can be included in a L^AT_EX file with `\includegraphics`, for example

```
\includegraphics{figure1.eps}
```

will insert an image in the document, contained in the file `figure1.eps`.

Note that while L^AT_EX will have no difficulty with this, pdfL^AT_EX will have to treat it differently. In that case, the image will have to be converted to a different format – PNG, JPEG or PDF are all accepted. To create a PDF file from the encapsulated PostScript, for instance, use `epstopdf figure1.eps`. This will create `figure1.pdf`, which can be included in a L^AT_EX file to be processed with pdfL^AT_EX. Alternatively, use the `latex` command to process the document, then obtain PDF from the DVI output. For example

```
latex document1
dvips -o document1.ps document1.dvi
ps2pdf document1.ps
```

METAPOST allows graphics to be created from a sequence of instructions contained in a source file. For example `mpost image1.mp` (or `mpost image1`) creates an output file in EPS format from the source file `image1.mp`.

A METAPOST source file can contain definitions for more than one figure – for example suppose the source file `figures.mp` is as follows:

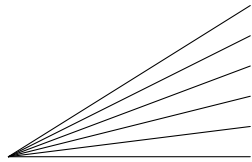
```
beginfig(1);
  drawing instructions
endfig;
beginfig(2);
  drawing instructions
endfig;
end;
```

then running `mpost figures.mp` will produce EPS files `figures.1` and `figures.2`.

2 Coordinates and Drawing in METAPOST

Coordinates in METAPOST are in \mathbb{E}^2 (\mathbb{R}^2 with the Euclidean metric). Note that having points in a figure with negative coordinates does not produce a document with the figure outside the page margin: the figure will be created with a bounding box of the correct size, but the absolute position is not specified.

Lines can be drawn connecting specified points using the `draw` instruction: for example



was produced from

```
beginfig(1);
u=.4cm;
draw (0,0)--(8u,0);
draw (0,0)--(8u,u);
draw (0,0)--(8u,2u);
draw (0,0)--(8u,3u);
draw (0,0)--(8u,4u);
draw (0,0)--(8u,5u);
endfig;
end;
```

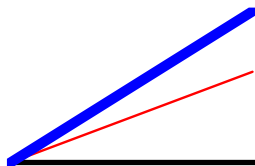
METAPOST has the ability to program loops, however, so this could have been achieved with

```
beginfig(1);
u=0.4cm;
for i=0 upto 5:
  draw (0,0)--(8u,i*u);
endfor
endfig;
end;
```

There are several ‘pens’ available of different shapes and sizes. For example

```
beginfig(2);
u=.4cm;
draw (0,0)--(8u,0) withpen pensquare scaled 2pt;
draw (0,0)--(8u,3u) withpen pencircle withcolor red;
draw (0,0)--(8u,5u) withpen pensquare scaled 2.8pt withcolor blue;
endfig;
end;
```

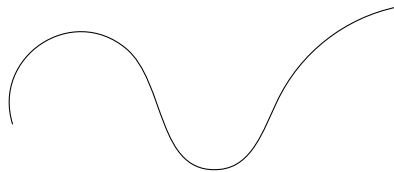
produces



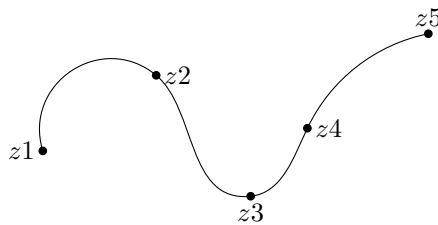
Arcs can be drawn in METAPOST and this is done by specifying points along the curve. For instance

```
beginfig(3);
u=.5cm;
z1=(0,0);
z2=(3u,2u);
z3=(5.5u,-1.2u);
z4=(7u,0.6u);
z5=(10.2u,3.1u);
draw z1..z2..z3..z4..z5;
endfig;
end;
```

produces



and the points $z1$ to $z5$ are shown here:



Several features of METAPOST are illustrated in the above figure showing the points $z1$ to $z5$. There is a macro definition to create a circular path, `fill` is used and there are labels. The figure was created from the source code on the following page. First, some general comments:

In METAPOST, the letters x , y and z are reserved for use as abscissa, ordinate and coordinate pair respectively. Other letters can **not** be used for these purposes.

METAPOST has *path* variables which can store a path. These path variables must have names entirely composed of alphabetic characters.

Macros can be defined with the `def` keyword, and macros can either carry out actions or define objects. Here a macro is used to define a circular path, because the path is used twice – first to draw the path, then to fill the path. Without this need, the macro could have been defined as

```
def circle(expr p, r)=
  draw p+(r,0)..p+(0,r)..p-(r,0)..p-(0,r)..cycle
enddef;
```

The source code for the figure is

```
beginfig(4);
u=.5cm;
r=.05cm;
def circle(expr p,r)=
  p+(r,0)..p+(0,r)..p-(r,0)..p-(0,r)..cycle
enddef;
z1=(0,0);
z2=(3u,2u);
z3=(5.5u,-1.2u);
z4=(7u,0.6u);
z5=(10.2u,3.1u);
draw z1..z2..z3..z4..z5;
path pa;
pa=circle(z1,r);
draw pa;
fill pa withcolor black;
path pb;
pb=circle(z2,r);
draw pb;
fill pb withcolor black;
path pc;
pc=circle(z3,r);
draw pc;
fill pc withcolor black;
path pd;
pd=circle(z4,r);
draw pd;
fill pd withcolor black;
path pe;
pe=circle(z5,r);
draw pe;
fill pe withcolor black;
label.lft(btex $z1$ etex, z1);
label.rt(btex $z2$ etex, z2);
label.bot(btex $z3$ etex, z3);
label.rt(btex $z4$ etex, z4);
label.top(btex $z5$ etex, z5);
endfig;
end;
```

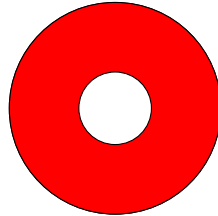
Colours to be used in drawing are set using `withcolor`, and the predefined colours are red, yellow, green, blue, black and white. Further colours can be defined as RGB (Red-Green-Blue) triplets, for example

```
color c;
c:=(0,98,120);
fill p withcolor c;
```

Labels are placed in a figure using the `label` command. This accepts optional specification of the label position relative to the object to which the label is attached. The position is indicated by one of `top`, `bot`, `lft`, `rt`, or on the 'corners' of the object at `ulf`, `urt`, `llf` or `lrt` (upper-left, upper-right, lower-left, lower-right).

3 Filling Areas in METAPOST

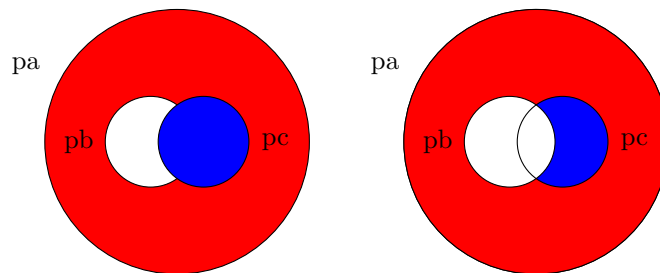
Areas can be filled with colour using the `fill` command. They can also be cleared using `unfill`. This can be useful in certain situations – for instance the annulus



was produced with

```
beginfig(5);
u=.4cm;
def circle(expr p,r)=
  p+(r,0)..p+(0,r)..p-(r,0)..p-(0,r)..cycle
enddef;
path pa;
path pb;
pa=circle((0,0),3.5u);
pb=circle((0,0),1.2u);
draw pa;
draw pb;
fill pa withcolor red;
unfill pb;
draw pa;
draw pb;
endfig;
end;
```

and note that the order of `fill` and `unfill` is important. For example the two figures

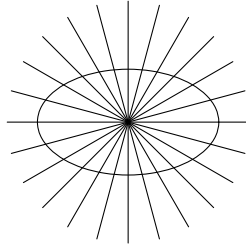


are produced with different orders of `fill` and `unfill`. Both fill `pa` first, but while in the first `pb` is unfilled before `pc` is filled, in the second, `pc` is filled before `pb` is unfilled.

Observe that filling covers anything drawn on the boundary of the curve being filled, so if outlines are to be shown in a figure, these will have to be drawn *after* any `fill`.

4 Cutting by Paths, Intersection Points

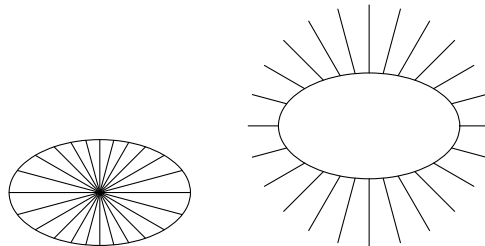
METAPOST is able to use a path to determine where a line or curve is to begin or end. That is, the line or curve may be drawn *up to* the point at which it reaches the path, or in may be drawn *from* the point at which it meets the path.



was produced with

```
beginfig(7);
u=.2cm;
def ellipse(expr p,a,b)=
  p+(a,0)
  for th=1 upto 24:
    ..p+(a*cosd(15*th),b*sind(15*th))
  endfor;
enddef;
path p;
p=ellipse((0,0),6u,3.5u);
draw p;
for th=1 upto 24:
  draw (0,0)--(8u*cosd(15*th),8u*sind(15*th));
endfor;
endfig;
end;
```

while

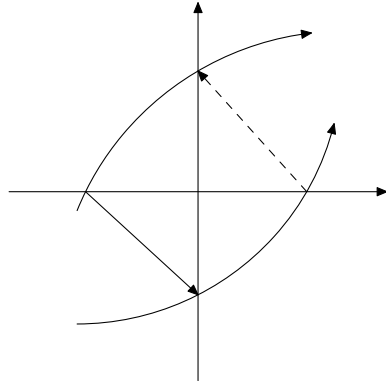


were produced substituting the following loops in the source code above:

```
for th=1 upto 24:
  draw (0,0)--(8u*cosd(15*th),8u*sind(15*th)) cutafter p;
endfor;

for th=1 upto 24:
  draw (0,0)--(8u*cosd(15*th),8u*sind(15*th)) cutbefore p;
endfor;
```

METAPOST can determine points of intersection and this can be a useful feature. Consider the following:



which comes from the source code

```

beginfig(10);
u=0.5cm;
path ax;
path ay;
path pa;
path pb;
ax=(-5u,0)--(5u,0);
ay=(0,-5u)--(0,5u);
pa=(-3.2u,-.5u)..(-2u,1.5u)..(3u,4.2u);
pb=(-3.2u,-3.5u)..(2u,-1.2u)..(3.6u,1.8u);
drawarrow ax;
drawarrow ay;
drawarrow pa;
drawarrow pb;
z1=pa intersectionpoint ax;
z2=pb intersectionpoint ay;
drawarrow z1--z2;
z3=pb intersectionpoint ax;
z4=pa intersectionpoint ay;
drawarrow z3--z4 dashed evenly;
endfig;
end;

```

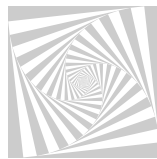
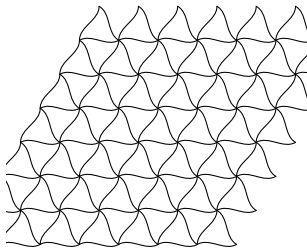
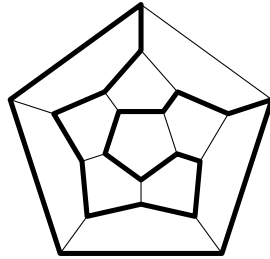
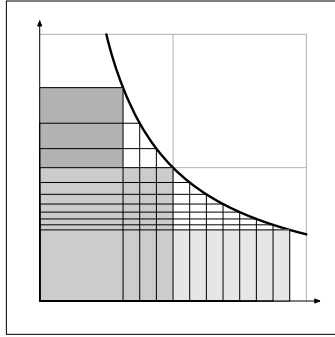
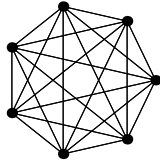
Comments:

This last example also illustrates different types of line (solid and dashed) and uses `drawarrow`, which functions like `draw` but places an arrowhead at the distal end of the line. If `drawarrow` is used on a curve, the arrowhead will be suitably rotated. Lines can be drawn solid (the default), or they can be drawn dashed in various styles (`dashed evenly`, `dashed withdots` for instance). In the case of dashing, a line can also be scaled, for example

```
draw p dashed withdots scaled 1.1;
```

This latter feature can be invaluable in making dashed or dotted lines fit nicely.

5 Examples (Exercises?)



For a number of interesting examples and further information, see
<http://tex.loria.fr/prod-graph/zoonekynd/metapost/metapost.html>
<http://www.ursoswald.ch/metapost/tutorial.html>