4 Lists, tables, figures

It is perfectly possible to write whole documents using nothing but section headings and paragraphs. As mentioned in section 2.6 on page 56, novels, for example, usually consist just of chapters divided into paragraphs. However, it's more common to need other features as well, especially if the document is technical[1] in nature or complex in structure.

In Chapter 2 starting on page 43 we saw how to create a hierarchical document structure with chapters and sections and paragraphs; this chapter covers the other building-blocks which you need within your structure: lists, tables, figures (including images), boxes like sidebars and panels, block quotations, and verbatim text (raw text like computer program listings). In Chapter 5 starting on page 135 we will cover the textual tools that you need *inside* text: footnotes, marginal notes, cross-references, citations, indexes, and glossaries.

[1] It's worth pointing out that 'technical' doesn't necessarily mean 'computer technical' or 'engineering technical', least of all 'mathematical technical': it just means it contains a lot of $\tau\acute{\epsilon}\chi\nu\eta$, Greek for specialist material or artistry. A literary analysis such as *La Textualisation de* Madame Bovary (on the marginal notes in the manuscripts of Flaubert's novel) is every bit as technical in the literary or linguistic field as the maintenance manual for the Airbus 380 is in the aircraft engineering field.

4.1 Lists

Lists are useful tools for arranging thoughts in a digestible format, usually a small piece of information at a time. There are four basic types of list, shown in below.

Table 4.1 – Types of list

Random or itemized lists (sometimes called 'arbitrary' or 'bulleted' lists) where the order of items is unimportant. The items are often prefixed with a bullet or other symbol for clarity or decoration, but are sometimes simply left blank, looking like miniature paragraphs (when they are known as 'simple' or 'trivial' lists).	**Enumerated or ordered lists** (sometimes called 'sequential' or 'numbered' lists) where the order of items is critical, such as sequences of instructions or rankings of importance. The enumeration can be numeric (Arabic or Roman), or lettered (uppercase or lowercase), and can be programmed to be hierarchical (1.a.viii, 2.3.6, etc).
Descriptive or labelled lists (sometimes called 'discussion' lists), which are composed of subheadings or topic labels (usually unnumbered but typographically distinct), each followed by one or more indented paragraphs of discussion or explanation.	**Inline lists** which are sequential in nature, just like enumerated lists, but are *a*) formatted *within* their paragraph; *b*) usually labelled with letters like this example; and *c*) often mutually inclusive or exclusive, with the final item prefixed by 'and' or 'or' respectively.

There are actually two other types, segmented lists and reference lists, but these are much rarer, and outside the scope of this document.

The structure of lists in LaTeX is identical for each type, but with a different environment name. Lists are another example of this LaTeX technique (environments), where a pair of matched commands surrounds some text which needs special treatment.

Within a list environment, list items are always identified by the command `\item` (followed by an item label in [square brackets] in the case of labelled lists). You don't type the bullet or the number or the formatting, it's all automated.

4.1.1 Itemized lists

To create an itemized list, use the *itemize* environment:

```
\begin{itemize}
\item Itemized lists usually have a bullet;
\item Long items use `hanging 'indentation, whereby
  the text is wrapped with a margin which brings it
  clear of the bullet used in the first line of
  each item;
\item The bullet can be changed for any other
  symbol, for example from the \textsf{bbding} or
  \textsf{pifont} package.
\end{itemize}
```

☞

- ✏ Itemized lists usually have a bullet;
- ✏ Long items use ‘hanging indentation’, whereby the text is wrapped with a margin which brings it clear of the bullet used in the first line of each item;
- ✏ The bullet can be changed for any other symbol, for example from the bbding or pifont package.

The default list bullet is the normal round, solid one (•), which is also available with the command \textbullet if you load the textcomp package. See section 7.6.2 on page 214 for details of how to change the settings for list item bullets.

Nested itemized lists (see section 4.1.6 on page 93) used differing symbols for their bullets as well as more indentation and less spacing.

4.1.2 Enumerated lists

To create an enumerated list, use the *enumerate* environment:

```
\begin{enumerate}
\item Enumerated lists use numbering on each item
  (can also be letters or roman numerals);
\item Long items use ‘hanging ’indentation in just
  the same way that itemized lists do;
\item The numbering system can be changed for any
  level.
\end{enumerate}
```

☞

1. Enumerated lists use numbering on each item (can also be letters or roman numerals);

2. Long items use ‘hanging indentation’, in just the same way that itemized lists do;

3. The numbering system can be changed for any level.

See section 4.1.6 on page 93 for details of how to change the numbering schemes for each level.

In standard LaTeX document classes, the vertical spacing between items, and above and below the lists as a whole, is more than between paragraphs. If you want tightly-packed lists, use the enumitem package, which provides an environment option *noitemsep* for the three main list environments (there is also a *nosep* option for even more compact spacing). Both these options come *after* the environment name, not before; eg `\begin{itemize}[nosep]`

4.1.3 Description lists

To create a description list, use the *`description`* environment:

```
\begin{description}
\item[Identification:] description lists require
  a topic for each item given in square brackets;
\item[Hanging indentation:] Long items use this
  in the same way as all other lists;
\item[Reformatting:] Long topic labels can be
  reprogrammed to fold onto multiple lines.
\end{description}
```

☞

Identification:: description lists require a topic for each item given in square brackets;

Hanging indentation:: Long items use this in the same way as all other lists;

Reformatting:: Long topic labels can be reprogrammed to fold onto multiple lines.

I *very strongly* recommend using the enumitem package with its *unboxed* environment option for *description* lists, which avoids the spacing problems with LaTeX's default handling of long labels. This package has so many good features I tend to load it for virtually every document I create.

All three of these types of lists can have multiple paragraphs per item: just type the additional paragraphs in the normal way, with a blank line between each. So long as they are still contained within the enclosing environment, they will automatically be indented to follow underneath their head item.

4.1.4 Inline lists

Inline lists are a special case, as they require the use of the enumitem or paralist packages.

The enumitem package with the *inline* option provides 'starred' versions of the three standard list types to do this: *enumerate**, *itemize**, and *description**. It uses a specification in the optional argument for formatting the labels (for example, italic letters and an upright parenthesis), and it also provides extensive support for the punctuation and conjunction between items, making it unnecessary to type it separately for each item (and differently for the last-but-one).

```
\usepackage[inline]{enumitem}
...
\textbf{\itshape Inline lists}, which are sequential
in nature, just like enumerated lists, but are
\begin{enumerate*}[label=\textit{\alph*}),
       itemjoin={{; }},itemjoin*={{; and }}]
\item formatted within their paragraph
\item usually labelled with letters
\item usually have the final item prefixed with
  `'and or `'or
\end{enumerate*}, like this.
```

☞ ***Inline lists***, which are sequential in nature, just like enumerated lists, but are *a*) formatted within their paragraph; *b*) usually labelled with letters; and *c*) usually have the final item prefixed with ‘and’ or ‘or’, like this.

See Chapter 6 starting on page 159 for details of the font-changing commands used in the optional arguments to the *enumerate** shown in this example.

Exercise 19 – List practice

1. Add a list or two to your document.
2. Copy and paste any two of the ones described here to practice with.
3. Read the documentation for the enumitem package or the paralist package and use it to change the layout of the lists you have added.

4.1.5 Reference lists and segmented lists

Reference lists are visually indistinguishable from numbered or lettered lists, but the numbering or lettering does *not* imply a sequence. The numbers or letters are just used as labels so that the items can be referred to from elsewhere in the text (as in 'see item 501(c)3'). In this sense they are really a kind of sub-sectional division, and LaTeX's `\paragraph` or `\subparagraph` commands (with appropriate renumbering) would probably be a far better solution than using a list. Label them and refer to them with `\label` and `\ref` as for any other cross-reference (see section 5.3 on page 138).

Segmented lists are a highly specialised structure and outside the scope of this document. For details of their usage, see the 'TEI Guidelines' (Burnard and Sperberg-McQueen 1995).

4.1.6 Lists within lists

You can start a new list environment within the item of an existing list, so you can embed one list inside another up to four deep. The lists can be of any type, so you can have a description list containing an item in which there is a numbered sub-list, within which there is an item containing a bulleted sub-sub-list.

1. by default an outer enumerated list uses Arabic numerals;

    (a) an embedded enumerated list is lettered in lowercase;

        i. a third level is numbered in lowercase Roman numerals;

            A. the fourth level uses uppercase alphabetic letters.

Multiple embedded lists automatically change the bullet or numbering scheme so that the levels don't get confused, and the spacing between levels is adjusted to become slightly tighter for more deeply nested levels.

- by default the outer itemized list item has a bullet;

    – an embedded itemized list uses a dash;

        ∗ a third level uses an asterisk;

            · the fourth level uses a small bullet.

Table 4.2 – Default numbering for nested numbered lists

Level	Default	Counter	Label command
1	digit.	enumi	\theenumi
2	(letter)	enumii	\theenumii
3	roman.	enumiii	\theenumiii
4	LETTER.	enumiv	\theenumiv

These are only defaults and can easily be changed by redefining the relevant set of values. You could also add a fifth and further levels, although I suspect that would mean your document structure needed some careful analysis, as lists embedded five deep will probably confuse your readers.

The values for lists come in pairs: for each level there is a counter to count the items and a command to produce the label:[2]

Note that each counter and command ends with the Roman numeral value of its level (this is to overcome the rule that LaTeX commands can only be made of letters — digits wouldn't work here). To change the format of a numbered list item counter, just renew the meaning of its label:

```
\renewcommand{\theenumi}{\Alph{enumi}}
\renewcommand{\theenumii}{\roman{enumii}}
\renewcommand{\theenumiii}{\arabic{enumiii}}
```

This would make the outermost list use uppercase letters, the second level use lowercase roman, and the third level use ordinary Arabic numerals. The fourth level would remain unaffected.

[2] In fact, any time you define a counter in LaTeX, you automatically get a command to reproduce its value. So if you defined a new counter example to use in a teaching book, by saying \newcounter {example}, that automatically makes available the command \theexample for use when you want to display the current value of example.

Exercise 20 – Nesting

Extend your use of lists by nesting one type inside a different one.

4.2 Tables

Tabular typesetting is the most complex and time-consuming of all textual features to get right. This holds true whether you are using cold or hot metal type, typing in plaintext form, using a typewriter or a wordprocessor, using LaTeX, using HTML or XML, using a DTP system, or some other text-handling package.

> Printers charge extra when you ask them to typeset tables, and they do so for good reason: Each table tends to have its own peculiarities, so it's necessary to give some thought to each one, and to fiddle with the alternative approaches until finding something that looks good and communicates well. (Knuth 1986)

Fortunately, LaTeX provides a table model with a mixture of defaults and configurability to let it produce very high quality tables with a minimum of effort. There are two things you need to know before you start: one is the terminology (see the panel

Lists and Tables: a caution to the unwary

Treat lists with care: people sometimes use tables for labelled information which is really a list and would be better handled as such. They often do this because their wordprocessor has no way to do what they want (usually to place the item label level with the description or explanation) *except* by using a table, hence they are misled into believing that their text is really a table when it's actually not.

'Terminology for tables and figures' on p. 96) and the other is what 'floats' are (see below

Terminology for tables and figures

LaTeX, in common with standard typesetting practice, uses the words 'Table' and 'Figure' with a very precise meaning:

Table: a formal textual feature, numbered, with a *caption*, and containing an aligned *grid* of numbers or text, referred to from the surrounding document (as in 'See Table 5'). A Table is the whole thing, not just the grid;

Figure: a formal graphical feature, numbered, with a caption, and an *image*, *diagram*, or piece of text, referred to from the surrounding document (as in 'See Figure 5'). A Figure is the whole thing, not just the image, diagram, or text.

Most importantly, Tables and Figures *float* (see section 4.2.1).

You can also have *informal tables*, which are simply grid alignments occurring between two paragraphs, with no caption or number or reference: there's one in section 1.5.2 on page 16; and you can have *informal figures* which are just images or diagrams done as *illustrations*. Informal figures and tables do *not* float.

The grid alignment of information in rows and columns in a Table is called a 'tabulation' or 'tabular matter', done with a `tabular` environment, and we'll deal with that in below.

4.2.1 Floats

Tables and Figures (and several other features of documents like sidebars) are what printers and publishers refer to as 'floats'. This means they are not part of the normal stream of your text, but separate freestanding entities, positioned in a part of the page to themselves (top, middle, bottom, left, right, or wherever the layout designer has specified). They always have a caption describing them and they are always numbered so they can be referred to from elsewhere in the text.

LaTeX automatically floats Tables and Figures, depending on how much space is left on the page at the point that they are processed. If there is not enough room on the current page, the float is by default moved to the top of the next page.

The positioning can be changed by moving the *`table`* or *`figure`* environment to an earlier or later point in the file, or by using the optional argument to the *`table`* or *`figure`* environment. This can be any mix of the letters `h` ('here'), `t` ('top'), `b` ('bottom'), `p` ('page by itself') to recommend where the Table or Figure should go (order is not significant: LaTeX will pick the best fit). To make your recommendation stronger, precede the first letter with an exclamation mark (`!`).

In this example you can see a Table requested to go here(!) or if not, at the top of the page; and a Figure requested to go at the bottom of the page or if necessary, on the next full page by itself.

```
\begin{table}[!ht]
...
\end{table}

\begin{figure}[bp]
 ...
\end{figure}
```

Authors sometimes want many figures or tables occurring in rapid succession, which is poor writing, as it's not just unfair to the reader, but raises the problem of how they are going to fit on the page and still leave room for text. In extreme cases, LaTeX will give up trying, and stack them all up until the end of the chapter or section for you to decide manually where to put them.

The skill is to space your tables and figures out within your text so that they intrude neither on the thread of your argument or discussion, nor on the visual balance of the typeset pages. But this is a skill few authors have, and it's one point at which professional typographic advice or manual intervention may be needed.

If you are unable to arrange things easily, as a last resort you can use the float package and the option letter capital H ('Here, dammit!'). Be aware that figures or tables using this package option *are no longer floats* so the onus is on you to ensure that the numbering sequence is not disrupted.

> Remember that if there really is not enough space 'here' on the page, then it *really* won't fit, and you will HAVE TO move things manually.

The float package also lets you create new classes of floating object (sidebars, examples, exercises, etc).

Please now read this section a second time. Getting the hang of floats can take a while if you've never come across the idea before. Most writers strongly recommend writing the document in its entirety first, and not worrying about where the floats end up until the text is complete and not likely to change any more. *Then* start moving any floats that are misplaced.

4.2.2 Normal tables

To create a LaTeX Table, use the *table* environment containing a \caption command followed by a \label command (the label is what you will use to refer to the table: we haven't done this yet, but see section 5.3.1 on page 138 if you're curious).

```
\begin{table}
  \caption{Project expenditure to year-end 2022}
  \label{ye2022exp}
  ...
\end{table}
```

Numbering is automatic, but the \label command MUST *follow* the \caption command, not the other way round. The numbering automatically includes the chapter or section number in document classes where this is appropriate. The \caption command has an optional argument to provide a short caption if the full caption would be too long for the List of Tables:

```
\caption[Project 2022]{Project expenditure to
  year-end 2022 showing utility costs as a
  separate item}
```

The caption is centered if it's shorter than one line; otherwise it is set full out. The caption and ccaption packages offer extensive customisation of the caption, including location and font.

below is an example that we can use to show how tables work (if you're reading this in on the web, ignore the shading: that's a part of the webpage style, not the LaTeX table).

Table 4.3 – Project expenditure to year-end 2022

	Item	**€ Amount**
a)	Salaries (2 part-time research assistants)	28,000
	Conference fees and travel expenses	14,228
	Computer equipment (5 workstations)	17,493
	Software licenses	3,562
b)	Rent, light, heat, etc	1,500
	Total	64,783

The Institute also contributes to items at (b).

4.2.3 Simple tabular matter

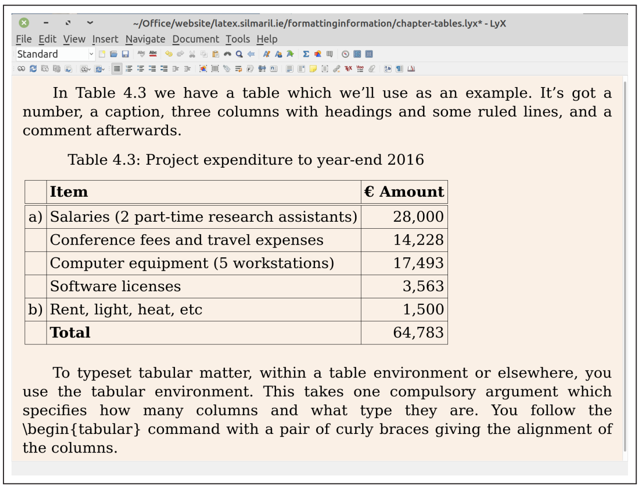
To typeset the grid within a *table* (or elsewhere),[3] you use the *tabular* environment. There are four ways to enter the data:

By hand: you can enter the tabular matter (cell data) by typing it in, which is perhaps the most common method, especially for small quantities of data;

In a grid tool: many LaTeX editors come with a pop-up grid tool like a miniature spreadsheet, which makes creating tabular matter easier, at the cost of some loss of fine control (see below).

[3] You can use the *tabular* environment anywhere you need stuff aligned in rows and columns, not just in a figure.

Figure 4.1 – Table being edited in LyX’s tabular editor


~/Office/website/latex.silmaril.ie/formattinginformation/chapter-tables.lyx* - LyX
File Edit View Insert Navigate Document Tools Help
Standard

In Table 4.3 we have a table which we’ll use as an example. It’s got a number, a caption, three columns with headings and some ruled lines, and a comment afterwards.

Table 4.3: Project expenditure to year-end 2016

	Item	**€ Amount**
a)	Salaries (2 part-time research assistants)	28,000
	Conference fees and travel expenses	14,228
	Computer equipment (5 workstations)	17,493
	Software licenses	3,563
b)	Rent, light, heat, etc	1,500
	Total	64,783

To typeset tabular matter, within a table environment or elsewhere, you use the tabular environment. This takes one compulsory argument which specifies how many columns and what type they are. You follow the \begin{tabular} command with a pair of curly braces giving the alignment of the columns.

With a package: if the quantity of data is very large and is already in a spreadsheet or database, or if it is data which will change frequently before you are finished your document, you can use the datatool package (formerly known as csvtools) to read the data from a Comma-Separated Values (CSV) spreadsheet import/export file (see section 4.2.4 on page 103). If the data changes, you just re-export it and re-run LaTeX.

For large numbers of tables in big documents (eg theses) this is by far the most accurate and time-saving method.

As an image: it is also possible to include a ‘table’ which has actually been captured as an image from elsewhere, such as a screenshot from a spreadsheet (so it’s not really a table just a picture of one). We will see how to include images in section 4.3 on page 113 on Figures, where they are more common.

In above and Table 4.3 on the preceding page there is the table which we’ll use as an example. It’s got a number, a caption, three columns with headings and some ruled lines, and a comment afterwards.

The `tabular` environment: This takes one compulsory argument which specifies how many columns there will be, and what type they are. You give one letter for each column using one of `l`, `c`, and `r` for a left-aligned, centered, or right-aligned column. The number of letters MUST be the same as the number of columns you are putting in the table.

```
\begin{tabular}{clr}
...
\end{tabular}
```

In the example in Table 4.3 on page 99, the tabular setting has three columns, the first one centered, the second left-aligned, and the third one right-aligned, so it is specified as `{clr}`. The dcolumn package provides a `d` column type for decimal alignment, and there are others we shall come across later.

Note

Each cell of these types (`c`, `l`, `r`, `d`) can hold only *one line of data* in its cell. If you need multi-line cells (like miniature paragraphs), see section 4.2.4 on page 103

Cell and row division: You can then type in each row, making sure each cell's data in the row is separated with an `&` character, and each row ends with a double backslash (`\\`).[4]

```
a)&Salaries (2 part-time research assistants)&28,000\\
```

You don't need to add any extra spaces or do any manual formatting, although you can if you want: LaTeX just uses the column specifications to know how to format it.

If a cell has nothing to go in it, you just don't type anything, but the ampersand must still be there:

[4] Note that this use of the double backslash to signal the end of a row is subtly different from the use we saw in section 1.10.5 on page 36 to terminate a normal text line prematurely. Here it marks the end of a table row.

```
&Total&64,783\\
```

Column headings: These are often set in **bold type**, as in the example (see 'Cell formatting' below).

```
&\textbf{Item}&\textbf{\EUR\ Amount}\\[6pt]\hline
```

In this case there is also some extra space (6pt, see 'Row spacing' below) to make it look nicer, and a horizontal line across the table (see the list item 'Table rules' below).

The data for a row may be longer than the width of the screen window in your editor, but it can take up as many lines on the screen as needed; the end of the row is always signalled by the double backslash, so LaTeX knows when it's time for the next row.

Cell formatting: Font changes can be done within a cell (bold, italic, etc; we'll come on to these later, see section 6.2.4 on page 189) and these changes are limited to the cell in which they occur: they do not 'bleed' across cells (in the example, the column headings have each been made bold separately).

Row spacing: Additional vertical white-space *below* a row (but above a rule) can be specified by giving a dimension in [square brackets] immediately after the double backslash which ends the row (3pt in the case of the last row before the totals in the example). A negative value will decrease the spacing below that row.

If the line below a horizontal rule looks too close, it can be optically spaced by adding a *strut* at the start of the next line (that is, *after* the `\hline`). A 'strut' is hidden vertical rule a little bit higher than the row-height; hidden because its width is zero, making it invisible, as in the example code. Use the `\rule` command for this, with a width of 0pt and height of 1.2em, just a fraction higher than the text, which will force the rows apart by 0.2em.

```
\begin{table}
\caption{Project expenditure to year-end 20}
\label{ye2022exp}
\centering\smallskip
\begin{tabular}{clr}
  &\textbf{Item}&\textbf{\EUR\ Amount}\\\hline\rule{0pt}{1.2em}
a)&Salaries (2 part-time research assistants)&28,000\\
  &Conference fees and travel expenses&14,228\\
  &Computer equipment (5 workstations)&17,493\\
  &Software&3,562\\
b)&Rent, light, heat, etc.&1,500\\[3pt]\cline{2-3}
\rule{0pt}{1.2em}&Total&64,783\\
\end{tabular}
\par\medskip\footnotesize
The Institute also contributes to (a) and (b).
\end{table}
```

Table rules: A line across the whole table is done with the \hline command after the double-backslash which ends a row.

For a line which only covers some of the columns, use the \cline command (in the same place), with the column range to be ruled in curly braces. If only one column needs a rule, it must still be given as a range (eg in the example, {3-3}).

Vertical rules (between columns) can be specified in the column specifications with the vertical bar character (|) before, after, or between the l, c, r letters. This character creates rules which extend the whole height of a table: it is not necessary to repeat them every row.

I have indented the code example given just to make the elements of the table clearer to read: this is for editorial convenience, and has no effect on the formatted result (see Table 4.3 on page 99). If you copy and paste this into your example document, you will need to add the marvosym package to your Preamble, which will let you use the official CEC-conformant Euro symbol command \EUR (€ as distinct from €).

4.2.4 More complex tabular formatting

TeX's original *tabular* environment was designed for classical numerical grids, where each cell contains a single value. If you need a cell to contain multiline text, like a miniature paragraph, you can use the column specification letter p (paragraph) followed by a width in curly braces instead of an l, c, or r. So p{3.5cm} would mean a column 3.5cm wide, where each cell can contain paragraph-style text, for example:

```
\begin{tabular}{cp{3.5cm}r}
```

These p column specifications are *not* multi-row (row-spanned) entries: they are single cells which can contain multiple lines of typesetting: the distinction is extremely important. These paragraphic cells are typeset justified (two parallel margins) and the baseline of the top line of text is aligned with the baseline of neighbouring cells in the row.

The array package provides some important enhancements which overcome the limitations of the p cells:

Vertical alignment: In addition to the p, whose vertical alignment baseline is the the top line of text, the array package provides the m and b letters. These work the same way as p (followed by a width in curly braces), but their vertical alignment baseline is the middle or bottom of the cell respectively.

Prefixes and suffixes: With the array package, any column specification letter can be preceded by >{} with some LaTeX commands in the curly braces. These commands are applied to every cell in that column, so to make a p column typeset ragged-right you would say, for example, >{\raggedright}p{3.5cm} (or \raggedleft, or \centering).

Note that if you do this, the last column specification MUST include a prefix or suffix containing the \arraybackslash command, to revert the meaning of the double-backslash, which gets redefined by horizontal formatting commands like \raggedright, otherwise you will get errors when the end-of-row double-backslash is not recognised.

There is a suffix format as well: you can follow a column letter with <{} with code in the curly braces (often used to turn off math mode started in a prefix).

The colortbl package lets you colour rows, columns, and cells; and the dcolumn package mentioned above provides decimal-aligned column specifications for scientific or financial tabulations. Multicolumn (column-spanning) is built into LaTeX tables with the \multicolumn command; but for multi-row (row-spanning) cells you need to add the multirow package. Multi-page and rotated (landscape format) tables can be done with the longtable, rotating, and landscape packages.

The LaTeX table model is very different from the HTML auto-adjusting model used in web pages; it's closer to the Continuous Acquisition and Life-cycle Support (CALS) table model used in technical documentation formats like DocBook. However, auto-adjusting column widths are possible with the tabularx and tabulary packages, offering different approaches to dynamic table formatting.

You do not need to format the tabular data in your editor: LaTeX formatting will typeset the table using the column specifications you provided. You can therefore arrange the layout of the data in your file for your own convenience: you can give the cell values all on one line, or split over many lines: it makes no difference so long as the cells are separated with the & and the rows are ended with the double-backslash.

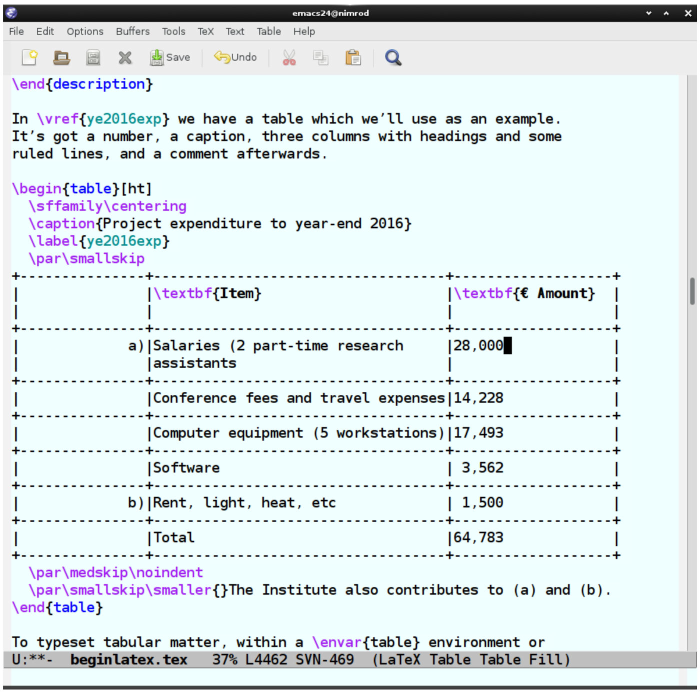
As mentioned earlier, some editors have a grid-like array editor for entering tabular data. Takaaki Ota provides an excellent *tables-mode* for *Emacs* which uses a spreadsheet-like interface and can generate LaTeX table source code (see below).

4.2.5 More on tabular spacing

Extra space, called a 'shoulder', is automatically added on both sides of all columns by default. The initial value is 6pt, so you get that amount left and right of the tabulation; because it is added left and right of every cell, the space between columns is therefore 12pt by default. This can be adjusted by changing the value of the \tabcolsep dimension *before* you begin the tabular environment.

```
\setlength{\tabcolsep}{3pt}
```

Figure 4.2 – Tables mode for *Emacs*



```
\end{description}

In \vref{ye2016exp} we have a table which we'll use as an example.
It's got a number, a caption, three columns with headings and some
ruled lines, and a comment afterwards.

\begin{table}[ht]
  \sffamily\centering
  \caption{Project expenditure to year-end 2016}
  \label{ye2016exp}
  \par\smallskip
+---------------+-----------------------------------+-------------------+
|               |\textbf{Item}                      |\textbf{€ Amount}  |
|               |                                   |                   |
+---------------+-----------------------------------+-------------------+
|             a)|Salaries (2 part-time research     |28,000             |
|               |assistants                         |                   |
+---------------+-----------------------------------+-------------------+
|               |Conference fees and travel expenses|14,228             |
+---------------+-----------------------------------+-------------------+
|               |Computer equipment (5 workstations)|17,493             |
+---------------+-----------------------------------+-------------------+
|               |Software                           | 3,562             |
+---------------+-----------------------------------+-------------------+
|             b)|Rent, light, heat, etc             | 1,500             |
+---------------+-----------------------------------+-------------------+
|               |Total                              |64,783             |
+---------------+-----------------------------------+-------------------+
  \par\medskip\noindent
  \par\smallskip\smaller{}The Institute also contributes to (a) and (b).
\end{table}

To typeset tabular matter, within a \envar{table} environment or
U:**-  beginlatex.tex   37% L4462 SVN-469  (LaTeX Table Table Fill)
```

The shoulder can be omitted in specific locations by adding the code `@{}` in the appropriate place[s] in the column specification argument. For example to omit it at the left-hand and right-hand sides of a tabular setting, put it at the start and end of the column specifications (putting it between two column specifications will remove all space between those columns).

```
\begin{tabular}{@{}clr@{}}
```

You can also use `@{}` to insert different spacing between columns (or at the right-hand and left-hand sides) by enclosing a spacing value; for example, `@{\hspace{2cm}}` could be used to force a 2cm space between two columns.

To change the row-spacing in a tabular setting, you can redefine the `\arraystretch` command (using `\renewcommand` because it's defined as a command, not a length). The value of `\arraystretch` is actually a ***multiplier***, preset to 1, so

`\renewcommand{\arraystretch}{1.5}` would set the baselines of your tabular setting one and a half times further apart than normal.

Exercise 21 – Calculate vertical spacing in a tabular environment

> Assume that you are making a table in the default size of 10pt type on a 12pt baseline. You want a 14pt baseline, so what value would you set `\arraystretch` to?

It is conventional to centre the tabular setting within a Table, using the *center* environment (note US spelling) or the `\centering` command (as in the example) — the default is flush left — but this is an æsthetic decision. Your journal or publisher may insist instead that all tabular material is set flush left or flush right (not the individual columns; the whole tabular setting inside the table).

If there is no data for a cell, just don't type anything — but you still need the `&` separating it from the next column's data. The astute reader will already have deduced that for a table of n columns, there must always be $n-1$ ampersands in each row. The exception to this is when the `\multicolumn` command is used to create cells which span multiple columns, when the ampersands of the spanned columns are omitted. The `\multicolumn` command takes three arguments: the number of columns to span; the format for the resulting wide column; and the contents. So to span a centred heading across three columns you would write `\multicolumn{3}{c}{The new heading}`.

The `\multicolumn` command can also be used to replace a *single* column if you need to vary some prefixing or suffixing or alignment specified in the column specification. For example if you have a right-aligned column (eg numbers) but you want one of the cells to be some text centered, you could write `\multicolumn{1}{c}{no data}`. In this case, of course, you keep all the ampersands, because you are not actually spanning columns.

4.2.6 Techniques for alignment

As mentioned earlier, it's perfectly possible to use the *tabular* environment to typeset any grid of material — it doesn't have to be inside a formal table. There are also other ways to align material without using a tabular format.

4.2.6.1 Using tabular alignment outside a table

By default, LaTeX typesets *tabular* environments *inline* to the surrounding text. That is, the *tabular* environment acts like a single character within the paragraph. This also means if you want an alignment displayed by itself, not as part of a formal table, you can put it between paragraphs (with a blank line or \par before and after) so it gets typeset separately; or put it inside a positioning environment like *center*, *flushright*, or *flushleft*.

One side-effect of this is that small and intricately constructed micro-tabulations can be used to good effect when creating special effects like logos, as they they get treated like a character and can be typeset anywhere.

Tabular setting can be used wherever you need to align material side by side, such as in designing letterheads, where you may want your company logo and address on one side and some other information on the other side to line up with each other. One common way to implement 'spring margins'[5] like this is to create two columns of whatever fraction of the page width you need (but adding to 1, of course), and removing for the extra space that would otherwise be added automatically between columns and at the edges:

```
\begin{tabular}{
          @{}
          >{\raggedright}p{.75\textwidth}
          @{}
          >{\raggedleft\arraybackslash}p{.25\textwidth}
          @{}}
```

[5] The term 'spring margins' comes from the DOS wordprocessor *PC-Write* and seems to be due to its author, the late Bob Wallace. I am not aware of any other mainstream system at the time that implemented them.

```
left-hand material
&
right-hand material\\
\end{tabular}
```

As mentioned earlier, the `@{}` suppresses the inter-column gap (or the shoulder left or right) so that the total width available will be the full text width of the page.

Exercise 22 – Create a tabulation

Create one of the following in your document:

1. a formal Table with a caption showing the number of people in your class broken down by age and sex;
2. an informal tabulation showing the price for three products;
3. the logo YEAR 2000 (hint: section 4.6.2 on page 127).

4.2.6.2 Alignment in general

Within the two-dimensional plane of conventional typesetting, there are two sets of axes to which the elements of the document should align: horizontal and vertical.

- □ The vertical axes are the left and right edges of the paper, the left and right margins of the text area, indentation, any internal temporary left and right margins (as for lists, block quotation, displayed mathematics, the left and right edges of illustrations, etc), and any internal column boundaries of a *tabular* environment.
- □ The horizontal axes are the top and bottom edges of the paper, the top and bottom margins of the text area, the space for running headers and footers, the top and bottom edges of all 'pool' items (see the start of this chapter), the baseline

of the text, and any internal row boundaries of a *tabular* environment.

Warning

If someone says they want something 'aligned', you need to ask 'aligned to *what*, exactly'? It's not always obvious, and in unusual cases it's not always easy to find out how to calculate or access an axis without careful study of the internal programming of a class or package.

By default, LaTeX starts each line up against the left-hand margin: if indentation is used, then the first lines of paragraphs will be indented, *except* for the first paragraph after a heading.[6]

□ Depending on the language you select in the babel or polyglossia packages, the first lines of first paragraphs after a heading may *not* be indented (for example in French typesetting).

□ In right-to-left languages, the alignment is reversed, and lines start up against the right-hand margin, and (see below) end against the left-hand margin.

The typeset line extends to the right-hand margin, and the process of justification ensures that all line-ends, apart from the last in a paragraph, align with this margin. The exception is when a *raggedright* or *raggedleft* or *centering* alignment has been specified.

Alignment to the four paper edges is extremely rare except in magazines and specialist formats like corporate reports or white papers, where images may be positioned to the edge[s] of the paper, and are said to 'bleed' off the sheet. It is of course possible in LaTeX but it is well outside the scope of this introductory text for beginners.

[6] This is known as Anglo-American usage (and applies to those countries and their [legacy] colonies, when using the English language).

4.2.6.3 Alignment within pool items

While typesetting a paragraph, LaTeX has no way to become aware of whereabouts a particular word or letter is being placed, for two reasons:

1. The justification of the paragraph does not start until after the whole paragraph has been typeset; only then does TeX start testing for line-end breakpoints, assigning them penalties, and inserting the variable spacing between words. This process is synchronous with the typesetting of the paragraph, and the next paragraph will not be started until justification of the one just ended is complete.

2. The positioning of the paragraph vertically on the page does not start until well after at least a whole page's worth of material has been typeset and justified, and the 'galley' of accumulated material comes close to filling up. At this point, TeX pauses typesetting of the next page (which it has already started), finds the optimum place to break the page, sends the completed page to the output, resets the accumulator to the remaining material, and then resumes typesetting. This process of page-building is therefore *a*synchronous with the process of typesetting, and the point at which access to already-typeset material ceases to be possible is not predictable in meaningful terms.

This means that doing things with stuff that has already been dealt with really isn't possible, and requests for it have to be respectfully declined. Anything you need to do with an item, whether it's a letter or a word or a paragraph, like applying a font change or putting it in a box, for example, needs to be done *in situ*, *before* it disappears down TeX's throat.

While there are packages for dealing with *completed* paragraphs, such as reledpar for typesetting synchronised parallel-page (eg dual-language) editions, access to the inside of the paragraphs is not possible at this stage. It *is*, however, possible to typeset material into a box, and then do things with it, including emptying it all back out again, in a limited manner. This makes it possible to see how much space a particular item is going to occupy, and then decide whether or not to treat it in a certain way. Standard LaTeX does this when deciding if a table or figure caption is narrow enough to fit centered on one line, or if it needs to set it full-out.

Packages which provide their own alignment options, such as enumitem for finer control of lists, usually specify in the documentation how to manipulate the shape and appearance of their environments. A substantial amount of this is about how to align one atomic value, such as a heading or title, with another one, such as the word which comes after it. In the case of the lettrine for dropped initial capitals, it's about how to adjust the capital (up, down, right, left) with respect to the indented rectangle into which it is to fit. In the case of the colortbl package for coloured rows and columns and cells in tables, it includes details of how to get the coloured block microadjusted.

4.2.6.4 Alignment to margins

The geometry package has extensive features for specifying the paper size, page size, margin sizes (left and right, if you are typesetting for double-sided work), marginal gaps, the head and foot settings for headers, footers, footnotes, and the gaps between them.

The description of line-alignment in the preceding section holds true for all text typeset inside further environments, for example in an *abstract* or a *quotation*, and within all lists, as well as the *p*/*m*/*b* column formats within a *tabular* setting. So long as you remain aware of the possible effect of unscoped formatting commands on lower-level nested environments, you can nest one environment inside another to an unspecified depth, and the rules of alignment will continue to be applied as much as possible. However, as with HTML and CSS, it is possible to overuse or abuse nesting, as it makes the code obscure.

Because the nesting of environments implies encapsulation, access to the alignment points (eg margins) of an outer environment is often not possible inside a deeper-nested environment. The TeX language model allows for the inheritance of settings defined at a higher level, but where these values are implemented as part of the code creating both the current environment *and* a higher one (eg lists inside lists), they will occupy the same space, and only the local value will be accessible. In such cases, any values needed would have to be saved in a variable accessible to the lower-level environment. In 30+ years of using LaTeX I have only ever needed to do this once.

4.2.6.5 Grids

Outside the *tabular* environment, LaTeX does *not* use a grid system. Its origins in mathematics mean that because displayed equations can occupy non-integer numbers of 'lines' (compared with text, which always occupies a whole number of lines), it was judged better for quality to allow flexible space *between* headings and paragraphs. Over the depth of a whole page, this minute amount of flexibility usually absorbs the fractional part of a line-height due to overspill in formulas (part of which 'rubberisation' led Leslie Lamport to choose LaTeX as the name for his set of macros).

There is a grid package available which enables grid setting in double-column documents, but overall there is no easy way to 'snap' pool elements to arbitrarily distanced gridlines. The flexibility of \parskip and the dozen or more other 'skips' (flexible lengths) in the LaTeX source (latex.ltx) could be removed, and display mathematics set in boxes of an integer number of \baselineskips, and special environments could be written to anchor themselves to a specific corner, but in general, the model of flexibility has proved itself over nearly 40 years, and requirements for grid models should be transferred to the NTS in the care of TUG and the LaTeX development specialists.

4.3 Figures

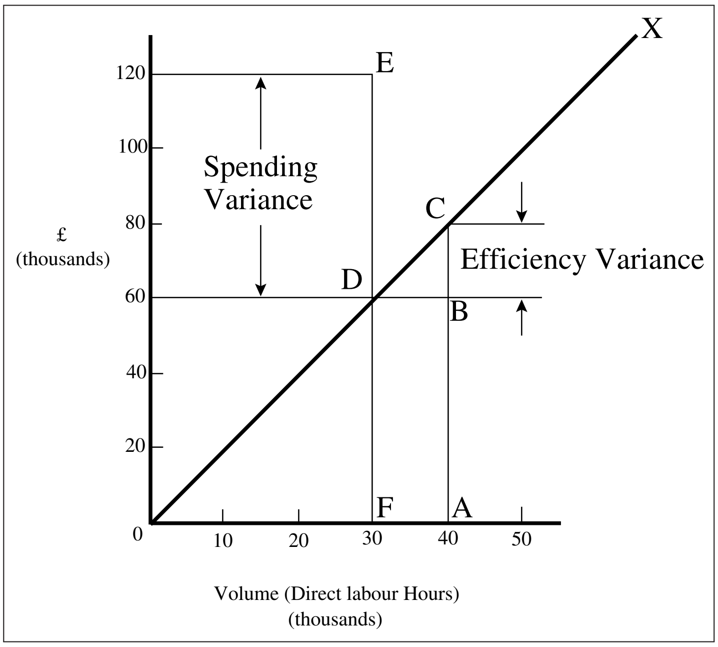
As explained in section 4.2 on page 95, Tables and Figures *float* to a vacant part of the page, as they are not part of your normal text, but illustrative objects that you refer to.

If you haven't already read the panel 'Terminology for tables and figures' on p. 96floatterms and section 4.2.1 on page 96, please read them now before going any further.

To create a figure, use the *figure* environment. Like Tables, they automatically get numbered, and they MUST include a \caption (with a \label after it, if needed for cross-referencing). Like Tables, it is conventional to centre the material, but that is a personal choice.

```
\begin{figure}
\caption{Total variable overhead variance (after
         \textcite[p.191]{bull}}
\label{workeff}
\centering
\fbox{\includegraphics[width=.75\columnwidth]{diagram}}
\end{figure}
```

Figure 4.3 – Total variable overhead variance (after Bull (1972, p. 191))



You can see that the structure is very similar to the *table* environment, but in this case we have a graphic included with the \includegraphics command. Here, it's also enclosed in an \fbox, which places a frame box around it (see section 4.6.2 on page 127). Details of including graphics are in the next section because they can occur in many places, not just Figures: you need the graphicx package. Details of the bibliographic citation mechanism used in the caption are will be covered in section 5.3.2 on page 140.

Figures can contain text, diagrams, pictures, or any other kind of illustration, even a *tabular* environment — LaTeX is agnostic on this point, so Tables can contain an image (of a table, presumably) and Figures can contain a tabulation. What matters is that you describe them properly.

The content of the Figure could of course also be textual, in the form of lists, paragraphs, or other blocks of text. For drawings, LaTeX has a very simple drawing environment called *picture*, which lets you create a very limited set of lines and curves, but for a diagram of any complexity, you can use any normal vector drawing program (see section 4.4.3 on page 120), save the image as a PDF vector image, and include it in your Figure with `\includegraphics` as illustrated.

Much more common is to create your diagrams within the LaTeX document using *TikZ*, which is a TeX interface to the Portable Graphics Format (PGF) graphics language. *TikZ* is much more powerful than the *picture* environment, but it's a whole language of its own, so it needs learning. (I used it to create the labelling on the image in Figure 1.4 on page 29.) The manual is in the tikz and there is a self-help blog at `latexdraw.com` and extensive help via the TeX Stackexchange.

4.4 Images

Images (graphics) can be included anywhere in a LaTeX document, although in most cases of formal documents they will occur in Figures (see section 4.3 on page 113). To use graphics, you need to use the graphicx package in your Preamble: `\usepackage{graphicx}`.[7] This package provides the command

[7] You may find a lot of old files which use a package called epsf. Don't use it: it's obsolete. The graphicx package has an 'x' because it's an extension on the older package called graphics.

\includegraphics which is used to insert an image in a document. The command is followed by the name of your graphics file *usually without the filetype* (we'll see in itemsection 4.4.1 on the facing page why you don't normally need to include the filetype).

```
\includegraphics{myhouse}
```

In most cases you should just make sure the image file is in the same folder (directory) as the document you use it in. This avoids a lot of messing around remembering where you put the files; but you could instead put them all in a single subfolder (for example, called `images`) and include that as part of the filename you use in the command.

```
\includegraphics{images/myhouse}
```

If you have images you want to use in several different documents in different places on your disk, there is a way to tell LaTeX where to look (see below).

4.4.1 Supported image file formats

The type of image file you use depends on LaTeX processor you are using (see section 1.2 on page 5 for how to choose). The common file types are:

- □ Joint Photographic Experts Group (JPG), used for photographs and scanned images;
- □ Portable Network Graphic (PNG), used for photographs and scanned images;
- □ Portable Document Format (PDF), used for vector graphics (drawings, diagrams) and typographic output from other programs;
- □ Encapsulated Postscript (EPS), an old publishing industry standard for many years, and the forerunner of PDF, still used by some older programs that generate diagrammatic or typographic output.

See below and below for other file formats. For more details, see the answers to the question on StackExchange, Which graphics formats can be included in documents processed by latex or pdflatex?. Basically, for modern systems running XƎLATEX, LuaLATEX or *pdflatex* (creating PDF output) graphics files MAY be in PNG, PDF, or JPG (JPEG) format ONLY. (You may come across very old systems running plain (original) LATEX (creating DVI output) in which graphics files MUST be in EPS format ONLY: no other format will work: see below)

□ PNG actually gets converted to the PDF internal format automatically (at a small penalty in terms of speed) so for lots of images, or very large images, use JPG format or preconvert them to PDF;

□ It is also of course possible to convert (repackage) your JPG pictures to PDF, using any of the standard graphics conversion/manipulation programs (see below for details). Preconverting all your images to PDF makes them load into your document slightly faster.

□ LATEX will search for the graphic file by file type, in this order (check for the newest definition in your `pdftex.def`): `.png`, `.pdf`, `.jpg`, `.mps`, `.jpeg`, `.jbig2`, `.jb2`, `.PNG`, `.PDF`, `.JPG`, `.JPEG`, `.JBIG2`, and `.JB2`.[8]

See section 4.4.3 on page 120 for more about how to create and manage your image files.

4.4.1.1 Other file formats

Convert them to one of the supported formats using a graphics editing or conversion tool such as the GNU Image Manipulation Program (GIMP), the *NetPBM* utilities, *ImageMagick*, or a utility like Péter Szabó's *sam2p* (not available with TEX Live but downloadable for Windows and Linux).

Some commercial distributions of TEX systems allow other formats to be used, such as GIF, Microsoft Bitmap (BMP), or Hewlett-Packard's Printer Control Language (PCL) files, and others, by using additional conversion software provided by the supplier;

[8] Thanks to Enrico Gregorio and Philipp Stephani on `comp.text.tex` for locating this for me.

but you cannot send such documents to other LATEX users and expect them to work if they don't have the same distribution installed as you have.

It is in fact possible to tell LATEX to generate the right file format by itself during processing, but this uses an external graphics converter like one of the above, and as it gets done afresh each time, it may slow things down.

4.4.1.2 Postscript

Since TEX 2010, EPS files will be automatically converted to PDF if you include the epstopdf package. This avoids need to keep your graphics in two formats, at the expense of a longer compile time while it converts every EPS image (not recommended).

All good graphics packages (eg *GIMP*, *PhotoShop*, Corel *Draw*, etc) can save images as EPS, but be very careful with other software such as statistics, engineering, mathematical, and numerical analysis packages, because some of them, especially on Microsoft Windows platforms, use a very poor quality driver, which in turn creates very poor quality EPS files. If in doubt, check with an expert. If you find an EPS graphic doesn't print, the chances are it's been badly made by the creating software. Downloading Adobe's own Postscript driver from their Web site and using that instead may improve things, but the only real solution is to use software that creates decent output.

For these reasons, if you create vector EPS graphics, and convert them to PDF format, *do not* keep additional JPG or PNG copies of the same image in the same directory, because they risk being used first by LATEX instead of the PDF file, because of the order in which it searches.

EPS files, especially bitmaps, can be very large indeed, because they are stored in ASCII format. Staszek Wawrykiewicz has drawn my attention to a useful MS-DOS program to overcome this, called *cep* ('Compressed Encapsulated PostScript') available from CTAN archive in the `support/pstools` directory, which can compress EPS files to a fraction of their original size. The original file can be replaced by the new smaller version and still used directly with `\includegraphics`.

One final warning about using EPS files with \includegraphics: never try to specify an absolute path (one beginning with a slash) or one addressing a higher level of directory (one beginning with `../`). The *dvips* driver will not accept these because they pose a security risk to *PostScript* documents. Unlike PDF, *PostScript* is a real programming language, capable of opening and deleting files, and the last thing you want is to create a document able to mess with your filesystem (or someone else's).

4.4.2 Resizing images

The \includegraphics command can take optional arguments within square brackets before the filename to specify the height or width, as in the example below. This will resize the image that prints; whichever dimension you specify (height or width) the other dimension will automatically be scaled in proportion to preserve the aspect ratio.

The file on disk does not get changed in any way, and nor does the copy included inside the PDF: what gets changed is just the size that it displays at in the finished document. So if you include a huge JPG but tell LaTeX to print it at a small size, your PDF will still include the whole image file at full size — all that changes is the way it shows it. This is very inefficient: normally you should create images at the right size for the document.

```
\begin{center}
  \includegraphics[width=5cm]{twithcat}
\end{center}
```

☞

If you specify both height *and* width, the image will be distorted to fit (not really useful except for special effects). You can scale an image by a factor (using the *scaled* option) instead of specifying height or width; clip it to specified coordinates; or rotate it in either direction. Multiple optional arguments are separated with commas.

For details of all the arguments, see the documentation on the graphicx package or a copy of the *Companion*. The package also includes commands to rotate, mirror, and scale text as well as images.

4.4.3 Making images

There are two types of image: bitmaps and vectors.

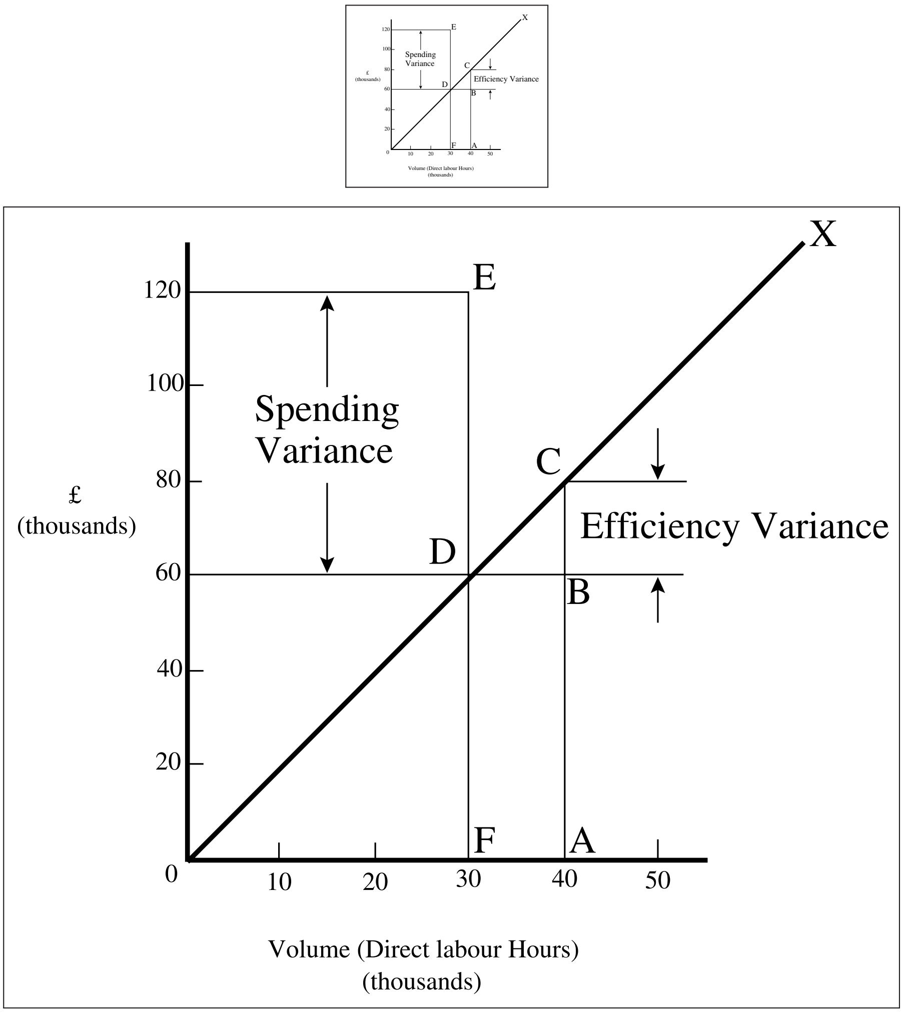
Bitmaps: Bitmap images are made of coloured dots, so if you enlarge them, they go jagged at the edges, and if you shrink them, they may go blurry. Bitmaps are fine for photographs, where every tiny dot is a different colour, and the eye won't notice so long as you don't shrink or enlarge too much. Bitmaps for diagrams and drawings, however, are almost always the wrong choice, and often disastrously bad.

Vectors: Vector drawings are made from instructions, just like LaTeX is, but using a different language (eg 'draw this from here to here, using a line this thick'). They can be enlarged or reduced as much as you like, and never lose accuracy, because they *get redrawn* automatically at any size. You can't do photographs as vectors, but vectors are the only acceptable method for drawings or diagrams.

Vector graphic packages are also better suited for saving your image directly in EPS or PDF format (both of which use their own vector language internally). All the major graphics-generating packages in all disciplines output vector formats: *AutoCAD*, *ChemDraw*, *MathCAD*, *Maple*, *Mathematica*, *ArcInfo*, and so on.

EPS was for decades the universally-accepted format for creating vector graphics for publication, with PDF a close second. PDF is now the most common format, but most of the major graphics (drawing) packages can still save as EPS, such as *PhotoShop*, *PaintShop Pro*, Adobe *Illustrator*, Corel *Draw*, and *GIMP*. There

Figure 4.4 – The vector diagram shrunk and enlarged



are also some free vector plotting and diagramming packages available like *InkScape*, *tkPaint*, and *GNUplot* which do the same. Never, ever (except in the direst necessity) create any *diagram* as a bitmap.

Bitmap formats like JPG and PNG are ideal for photographs, as they are also able to compress the data substantially without too much loss of quality. However, compressed formats are bad for screenshots, if you are documenting computer tasks, because too much compression makes them blurry. The popular Graphics Interchange Format (GIF) is good for screenshots, but is not supported by TEX: use PNG instead, with the compression turned down to minimum.

Avoid uncompressible formats like BMP as they produce enormous and unmanageable files. The Tagged Image File Format (TIFF), popular with graphic designers, should also be avoided if possible, partly because it is even vaster than BMP, and partly because far too many companies have designed and implemented non-standard, conflicting, proprietary extensions to the format, making it virtually useless for transfer between different types of computers (except in faxes, where it's still used in a much stricter version).

Exercise 23 – Adding pictures

Add \usepackage{graphicx} to the Preamble of your document, and copy or download an image you want to include. Make sure it is a JPG, PNG, or PDF image.

Add \includegraphics and the filename in curly braces (without the filetype), and process the document and preview or print it.

Make it into a figure following the example in section 4.3 on page 113.

4.4.4 Graphics storage

I mentioned earlier that there was a way to tell LATEX where to look if you want to store some images centrally for use in many different documents.

If you want to be able to use some images from any of your LATEX documents, regardless of the folder[s], then you should store the images in the subdirectory called `tex/generic` within your Personal TEX Directory.

Otherwise, you can use the command \graphicspath with a list of one or more names of additional directories you want searched when a file uses the \includegraphics command.

Put each path in its own pair of curly braces within the curly braces of the \graphicspath command. I've used the 'safe' (MS-DOS) form of the Windows My Pictures folder in the example here because you should never use directory or file names containing spaces (see the panel 'Picking suitable filenames' on p. 49filenames).

Using \graphicspath does make your file less portable, though, because file paths tend to be specific both to an operating system and to your computer, like the examples above.

```
\graphicspath{{c:/mypict~1/camera}{z:/corp/imagelib}}
\graphicspath{{/var/lib/images}{/home/peter/Pictures}}
```

If you use original LaTeX and *dvips* to print or create PostScript files, be aware that some versions will not by default handle EPS files which are outside the current directory, and will issue the error message saying that it is 'unable to find' the image. As we mentioned above, this is because PostScript is a programming language, and it would theoretically be possible for a maliciously-made image to contain code which might compromise your system. The decision to restrict operation in this way has been widely criticised, but it seems unlikely to be changed. If you are certain that your EPS files are kosher, use the R0 option in your command, eg dvips -R0 ... dvifile

4.5 Quotations

Direct speech and short quotes within a sentence 'like this' are done with simple quotation marks as described in section 1.8 on page 24. Sometimes, however, you may want longer quotations set as a separate paragraph. Typically these are indented from the surrounding text. LaTeX has two environments for doing this.

Such quotations are often set in a smaller size of type, although this is not the default, but you can use one of the size commands like \small (see section 6.2.5 on page 191) as shown in the example.

```
\begin{quotation}\small\noindent
At the turn of the century William Davy, a Devonshire
parson, finding errors in the first edition of his
\citetitle{davy}, asked for a new edition to be printed.
His publisher refused and Davy purchased a press, type,
and paper. He harnessed his gardener to the press and
apprenticed his housemaid to the typesetting. After
twelve years' work, a new edition of fourteen sets of
twenty-six volumes was issued---which surely indicates
that, when typomania is coupled with religious fervour,
anything up to a miracle may be
achieved.\hfill\textcite[p.76]{ryder}
\end{quotation}
```

> ☞ At the turn of the century William Davy, a Devonshire parson, finding errors in the first edition of his *A System of Divinity*, asked for a new edition to be printed. His publisher refused and Davy purchased a press, type, and paper. He harnessed his gardener to the press and apprenticed his housemaid to the typesetting. After twelve years' work, a new edition of fourteen sets of twenty-six volumes was issued—which surely indicates that, when typomania is coupled with religious fervour, anything up to a miracle may be achieved. (Ryder 1976, p. 76)

The inclusion of a bibliographic citation at the end is optional but commonplace, especially in academic or research documents where it may be compulsory because of the need to cite everything you quote. It's also possible in LaTeX for this to be tucked into the space at the end of the last line of the quotation, if there is room (if it's too long, or not predictable [like a web page], it should go on a line by itself, as it does in the web version of this document).

The *quotation* environment sets the whole block of text indented, and each paragraph of it also has its own indentation on the first line, even the first paragraph. This is rather unconventional as a default, so it is common to add a \noindent command at the start of the quotation so that the first paragraph does *not* get indented (others still will).

4.6 Boxes, sidebars, and panels

LaTeX, like most typesetting systems, works by setting text into boxes. Each character is also a box, with a height and a width, just like it was in metal type; characters are assembled into lines, which are also boxes; and lines are assembled into pages, which are also boxes. The page-making mechanism also works like an old compositor's galley (tray) from the days of metal type: the box accumulates lines of typeset text until it's a bit longer than the height of the page. TeX then works out how much of it really will fit on the page, cuts it off and ships it out to the PDF file, and puts the remainder back into the galley (box) at the top, ready to start accumulating more material for the following page.

4.6.1 Boxes of text

Because of this 'box model', LaTeX can typeset any text into a box of any width. The simplest command for small amounts of text is \parbox. This command needs two arguments in curly braces: the first is the width you want the text set to, and the second is the text itself, as in the example shown.

```
\parbox{3in}{Please make sure you send in your
completed forms by January 1st next year, or the
penalty clause in 2(a) will apply.}
```

☞

Please make sure you send in your completed forms by January 1st next year, or the penalty clause in 2(a) will apply.

The text is typeset to the required width, and the box is extended downwards for as long as is required to fit the text. Note that the baseline of a \parbox is set to the midpoint of the box, so if you include a \parbox in mid-sentence, the centre of the box will be lined up with the line of type currently being set. You can specify that the top or bottom should be the alignment point by adding an optional *t* (top) or *b* (bottom) in square brackets before the width. For example, `\parbox[t]{3in}{...}` will produce a box with the baseline aligned with the top line of the text in the box.

Where the contents is more complex, use the *minipage* environment.

```
\begin{minipage}{4in}
  Please make sure you send in your completed forms
  by January 1st next year, or the penalty clause
  2(a) will apply:
  \begin{itemize}[noitemsep]
    \item Incomplete forms will be returned to you
          unprocessed.
    \item Forms must be accompanied by the correct fee.
    \item There is no appeal. The adjudicators'
          decision is final.
  \end{itemize}
\end{minipage}
```

☞ Please make sure you send in your completed forms by January 1st next year, or the penalty clause 2(a) will apply.

- Incomplete forms will be returned to you unprocessed.
- Forms must be accompanied by the correct fee.
- There is no appeal. The adjudicators' decision is final.

Notice that when setting very narrow measures with type that is too large, the words may not fit nicely and the spacing may become uneven or there may be too much hyphenation. Either use `\raggedright` or reduce the type size, or (in extreme cases) reword the text or break each line by hand. Fortunately, it is rare for LaTeX to need this level of attention.

Within a *minipage* environment you can use virtually everything that occurs in normal text (eg lists, paragraphs, tabulations, etc) with the exception of floats like Tables and Figures. The *minipage* environment takes a compulsory argument just like `\parbox` does, and it means the same: the width you want the text set to.

Note that in both *minipage*s and \parboxes, the paragraph indentation (\parindent) is reset to zero. If you need to change it, do so *inside* the *minipage* or \parbox using the \setlength command (see section 2.7 on page 62).

Because a minipage is typeset independently from the rest of your text, any footnotes inside a minipage will be typeset at the end of the minipage, *not* at the foot of the containing page, and they will be done using lowercase letters by default, to keep them separate from the normal footnotes. We haven't done footnotes yet, but they're in section 5.1 on page 135.

There are other ways of typesetting text to widths other than the normal text width: you can use a one-row, one-cell *tabular* environment with the p column type specification; or you can use the technique of setting the material into a special box that remembers it, and then emitting it where you want it (this is implemented by the standard *lrbox* environment or by the *Sbox* environment from the fancybox package, but these are advanced techniques).

4.6.2 Framed boxes

To put a frame round some text, use the \fbox command:

```
\fbox{some text}
```

We already saw this used in the Quick Start document and also to frame an image in Figure 4.3 on page 114. For text, this works for a few words in mid-line, but the framed box and its contents won't break over the end of a line. To typeset multiline text in a box, put it in a \parbox, or use a *minipage* or *tabular* environment as described above, and enclose the whole thing in a \fbox.

```
\fbox{\parbox{3in}{Please make sure you send in your
completed forms by January 1st next year, or the
penalty clause 2(a) will apply.}}
```

The spacing between text and box is controlled by the value of \fboxsep, and the thickness of the line by \fboxrule, both of which can be reset with the \setlength command.

If you are using colour, the xcolor package extends boxing to the \colorbox command, which takes two arguments: a colour name or code for the background colour, and the text (which will need a foreground colour if black would not be suitable):

```
\colorbox{green}{\textcolor{white}{some text}}
```

The package also provides \fcolorbox which puts a frame around a coloured box; in this case the first argument is the frame colour, the second the background colour, and the third the contents.

4.6.3 Panels and sidebars

The fancybox package lets you extend the principle of \fbox with commands to surround text in square, oval (round-cornered), and drop-shadow boxes (eg \ovalbox, \shadowbox, etc: see the documentation for details).

You can create panels of any size with these borders by typesetting your text in a *minipage* environment inside the *Sbox* environment which fancybox provides. The *minipage* formats the text but the *Sbox* 'captures' it, allowing you to delay putting the frame around until it is complete (so it knows the size).

The printed version of this document uses this extensively and there is a worked example shown in section 7.5 on page 210.

Sidebars are blocks of text, usually explaining something too detailed to put in the main text. They sometimes go at the side of the page as their name suggests, but can also be floats like Tables and Figures. The float already mentioned lets you create your own new types of float as environments, so you could define one called *sidebar* and use that to handle your sidebars, putting a boxed block of text in each one.

4.7 Verbatim text

If you are documenting computer procedures, you probably need fixed-width type for examples of programming or data input or output. Even if you are writing about completely non-computer topics, you may often want to quote a URI, filename, an email address, or raw text containing characters which you don’t want interpreted by LaTeX and which needs to be typeset specially, using a fixed-width font.

LaTeX includes two features for handling fixed-format text: inline verbatim and display verbatim. There are many more variations available in other packages.

4.7.1 Inline verbatim

To specify a word or phrase as verbatim text in typewriter type within a sentence, use the special command \verb, followed by the word or phrase surrounded by any suitable character which does *not* occur in the word or phrase itself. This is a very rare exception to the rule that arguments go in curly braces.

For example, you could use the plus sign to show a LaTeX command in a manual like this one:

```
You can typeset a phrase verbatim, even if it includes
\LaTeX\ command characters, for example the command to
insert an image: \verb+\includegraphics[width=3in]{myhouse}+
```

You can typeset a phrase verbatim, even if it includes LaTeX command characters, for example the command to insert an image: \includegraphics[width=3in]{myhouse}

The plus sign is ‘safe’ to use here because it doesn’t appear in the code you want to typeset but you could use the grave accent or backtick key ‘ or the vertical bar | if the phrase already had a plus sign in it:

```
for example \verb|\(a=b+c\)| when illustrating the
\LaTeX\ equation \(a=b+c\).
```

for example \verb|\(a=b+c\)| when illustrating the LaTeX equation $a = b + c$.

The \verb command has the advantage that it turns off all special characters (see section 1.7 on page 22) except the one you use as the delimiter, so you can easily quote sequences of characters in any computer syntax — including TEX. However, LATEX will never break the argument of `\verb` at a line-end when formatting a paragraph, even if it contains spaces, so if it happens to be long, and falls towards the end of a line, it *will* stick out into the margin. See section 1.10.2 on page 32 for more information on line-ends and hyphenation. The argument to `\verb` MUST NOT contain a linebreak in your editor: this will cause it to fail. See also the warning about using `\verb` inside `\footnote`.

4.7.1.1 Typesetting URIs

Typesetting URIs causes two problems which LATEX overcomes with the url package (and the hyperref package which also handles URIs).[9]

Visible accuracy: It is important for them to be visibly accurate, so they can be copied and retyped from print.

It is therefore essential to use a typeface which distinguishes well between 1 (digit one), l (lowercase ell) and I (uppercase eye), and between 0 (zero) and O (uppercase oh). Monospaced 'typewriter' type usually makes this clear, but many sans-serif fonts do not. It is a common error by designers not to distinguish URIs in this way.

Intrusive hyphens: URIs (especially long ones) must never have a hyphen added if they have to be broken over a line-end, because it might be mistaken (and retyped) by the user for a real hyphen needed as a part of the address. Conversely, any existing hyphen must never be used at a line-end.

Handling a URI therefore means being able to perform a hyphenless line-break, but only at some existing punctuation characters — principally the slash and the dot — but not usually the hyphen.

[9] The original term Uniform Resource Locator (URL) is now deprecated in favour of the more accurate Uniform Resource Identifier (URI). For details see `www.w3.org/Addressing/`. The older term still persists, especially in this LATEX package and its command, and in some XML markup vocabularies.

The url package provides the command \url which works in the same way as \verb, but uses the standard curly braces to enclose the address, eg \url{http://latex.silmaril.ie} — the command understands the syntax of a URI (Berners-Lee, Fielding and Masinter 2005) and will never break mid-way through an unpunctuated word, only at slashes and full points (and never at embedded hyphens unless in exceptionally rare circumstances when the *hyphen* package option can be used).

Bear in mind, however, that spaces and non-ASCII characters are (currently) invalid in URIs, so using spaces in a \url argument will cause it to fail, as will using other non-URI-valid characters.

The hyperref package make links for cross-references, citations, and URIs clickable in LaTeX PDFs (it's used in the PDF version of this document). It can also make footnote links clickable, although as they are usually on the same page, this is probably not needed in most documents.

You can load it with footnote links turned off, and the different classes of link (cross-references, citations, and URIs) in different colours:

```
\usepackage[hyperfootnotes=false]{hyperref}
\hypersetup{linkcolor=DarkGreen,citecolor=DarkRed,
            urlcolor=Blue,pdfborder=0 0 0,
            allbordercolors=0 0 0,colorlinks=true}
\AtBeginDocument{\renewcommand{\UrlFont}{\ttfamily}}
```

In some systems, links get borders drawn around them, but in the example above they are turned off by making their colour invisible (0 0 0). The hyperref package likes to use the roman font for URIs (normally A Bad Idea as we saw above in the list item 'Visible accuracy' above) but this can be turned off by changing the vaue of \UrlFont back to ttfamily.

4.7.1.2 Enhanced inline verbatim

The listings package, which we look at more below for display verbatim, also has an inline form. This can use colour to highlight your examples based on the language you are documenting — I am using it extensively in the PDF of this book.

The command \lstinline uses the same syntax as \verb (two matching but otherwise unused characters) to enclose the argument, but it provides for very extensive options to specify the language, font, size, style, and formatting. The most useful is the language, of which about 100 are predefined, from *ADA* to *Verilog*, and you can add new keywords and even whole new languages.

This is probably the most effective way to show computer-language examples inline, because it handles the syntax-based enhancement for you. It is, however, still subject to the same limitations as \verb, in that the code must fit on the space available in the line, or it will stick out into the margin.

```
For example, you could use the plus sign to show a \LaTeX\ command:
\lstlisting[language={[LaTeX]TeX}]`\verb+\includegraphics[width=3in]{myhouse}+`
in order to display
\lstlisting{[LaTeX]TeX}]`\includegraphics[width=3in]{myhouse}`, because the
plus sign does not occur in the command, and is therefore free to be used.
```

☞ For example, you could use the plus sign to show a LaTeX command: \verb+\includegraphics[width=3in]{myhouse}+ in order to display \includegraphics[width=3in]{myhouse}, because the plus sign does not occur in the command, and is therefore free to be used.

4.7.2 Display verbatim

For longer (multiline) chunks of fixed-format text like examples of programming, use the `verbatim` environment. Like \verb, this turns off all special characters, so you can include anything at all in the verbatim text except the exact line \end{verbatim}, which MUST occur on a line by itself, starting at the beginning of the line (*not* indented)[10].

```
\begin{verbatim}
\documentclass[11pt,a4paper,oneside]{report}
```

[10] A number of environments which do complicated things with the category codes of characters have this requirement to end the environment with the \end {...} command on a line by itself, at the start of the line. These include the `Verbatim` environment from the fancyvrb package, and any of the \end {...} commands taken over by the endfloat package.

```
\begin{document}

\title{Practical Typesetting}
\author{Peter Flynn\\Silmaril Consultants}
\date{December 2004}
\maketitle

\end{document}
\end{verbatim}
```

For more control over formatting there are two useful packages: the verbatim package, which overcomes a few of the limitations of the built-in *verbatim* environment; and the fancyvrb package, which provides much greater flexibility with a *Verbatim* environment (note the capital letter).

Exercise 24 – Try some fixed-format text

1. Add your email address and home page URI using the \verb and \url commands. You'll need to add \usepackage{url} to your Preamble for the latter.
2. Load the listings package and try the \lstinline command to do the same.

However, as I mentioned above, for a much more powerful verbatim environment, I use the listings package for its ability to colour the keywords of a program according to the language used. It can also add rules, interpret internal formatting, and include external files, and let you add your own language definitions for new languages. The penalty is a slightly more complex configuration, but if you are documenting any kind of computer code in significant quantities, the quality and usability of the result is well worth it.

```
\documentclass[11pt,a4paper,oneside]{report}
\begin{document}

\title{Practical Typesetting}
```

```
\author{Peter Flynn\\Silmaril Consultants}
\date{December 2004}
\maketitle

\end{document}
```