placed in the margin by the `\putinmargin` command. This dimension must be smaller than the amount of space in the margin itself (which is `\fullpgwid` minus `\cutpgwid`). Set this variable in your format file. (Changing its value elsewhere will not be effective since it is used in calculations performed immediately after reading the format file.)

10.3 `\finput`*,* `\tinput` *and Captions*

The commands `\finput` and `\tinput` are used for loading and captioning figures and tables respectively. There are two things which make this really different than the LaTeX routine for captioning:

- The type of caption which appears does *not* depend on the environment. On the other hand, in LaTeX, the `table` environment or `figure` environment (which are not really supposed to be used with ASTEX) determined the type of caption. In ASTEX, you can use either `\finput` or `\tinput` in a `stayfig`, `floatfig` or `minipage` environment or inside of a `\putinmargin` command. You can also put a table and figure next to eachother, which would not have been easy in LaTeX!
- Since the command takes both the caption and the commands which generate the figure or table as arguments[3], it can perform more complicated interactions. For example, by default, the width of the caption is the same as the width of the figure or table. Furthermore, by using the commands `\figontop` or `\figonbot` (and their table counterparts `\tabontop` and `\tabonbot`) you can globally determine whether the caption appears above or below the figure.

In addition, `\finput` and `\tinput` automatically label the object, so that you can refer to it with a `\ref` command, and they allow greater control of the appearance of the caption.

`\finput` (❖) is used for loading figures and putting captions on them. In general, it takes two arguments, the name of a file which contains commands for generating the figure and a caption. In this case, the syntax is simply:

```
\finput{FILENAME}{CAPTION}
```

The figure number is automatically labeled by the `FILENAME`, so that you can refer to that figure in the text with the expression `Figure~\ref{FILENAME}`. The figure and the caption together are automatically placed in a single `\vbox`. Consequently, TeX will treat them as a *single* object. For example, putting two `\finput` commands next to eachother will result in both pictures being placed next to eachother (when they fit on the same line).

An asterisk ("`*`") indicates that no caption will appear on the figure, only a *figure number*. In this case, the expression "Figure 1.3" (or whatever) will appear without any caption. The basic syntax here is:

```
\finput*{FILENAME}
```

Otherwise, everything remains as above.

If the commands which are going to generate the figure are *not* in a separate file, then you can use an alternate syntax indicated by square brackets:

```
\finput[LABEL]{COMMANDS FOR GENERATING FIGURE}{CAPTION}
```

[3]The particular commands which you use to generate the figure or table are not important. ASTEX commands for generating figures are discussed in Section 11.1 of this document, and commands for generating tables are discussed in LaTeX manuals.

Here the figure number will be labelled by `LABEL`, and the specified commands will generate the figure. To make sure you understand this, note that:

```
\finput{FILE}{CAPTION} = \finput[FILE]{\input{FILE}}{CAPTION}
```

If an asterisk is placed before the option in the square brackets then no caption will be expected.

Note: If the argument in square brackets is simply a hyphen, then no `\label` will be made for that figure. For example

```
\finput[-]{\input{a}\qquad\input{b}}{This is the caption}
```

cannot be referred to with a `\ref` command.

By default, the width of the caption will be the same as the width of the picture. However, an optional argument allows you to specify the width of the caption explicitly. (You might want to do this when the caption is particularly long or the figure is particularly narrow.) Merely place the dimension in angular brackets in front of the caption in either of the cases above:

```
\finput{FILE}<DIM>{CAPTION}
\finput[LABEL]{COMMANDS FOR GENERATING FIGURE}<DIM>{CAPTION}
```

There are two fonts associated with the caption in an finput command.

`\figlegfont` (✡) is the font in which the actual words "Figure 1.3" will be printed.

`\figcapfont` (✡) is the font in which the caption will be printed.

Furthermore, there is one dimension associated to the command `\finput` and one "switch" which must be set in your format file. To determine whether the figure will appear over the caption or the other way around, use either `\figontop` or `\figonbot`.

`\figontop` (❖) is a flag which indicates that when an `\finput` command is called that the figure will appear over the caption. Although you *could* use this in the middle of a document to alter the appearance of the output of `\finput`, it is expected that you will use the command only in the format file.

`\figonbot` (❖) is similar to `\figontop`, but it puts the figure below the caption.

`\fcapskip` (↦) is the space between a figure and its caption when generated with the `\finput` command (regardless of which is on top.

`\fcaptype` (❖) determines the appearance of figure captions. By default it is set to `\centering`, which will cause each line of the caption to be centered below the figure. Other reasonable settings for this command would be `\relax` (to stretch out each line to the margins), `\flushleft` (to push each line to the left) or `\flushright` (to push each line to the right).

Everything is basically the same for `\tinput`. The only reason the two commands are separate is that many books have very different specifications for figures and captions. If you set all of the parameters the same, there is no reason that your figure and table captions would not look the same.

`\tinput` (❖) is essentially the same as `\finput`. It will load and caption a table. The commands analogous to those described above are, `\tablegfont`, `\tabcapfont`, `\tcapskip`, `\tabontop`, `\tabonbot` and `\tcapstyle`.

Note: If, for some reason, you need to use the captions without the input commands, you can use the commands `\fcaption`, `\tcaption`, `\fnumber` and `\tnumber` (which are not documented).

`\nofnumber` (❖) is a flag which temporarily turns of the assignment and printing of a figure number. That is, although a caption will be put under a picture, the word "Figure" and the figure number will not be printed. If you put it at the top of a file, then this will be true for the entire file. Alternatively, you can include it in the definition of your "exercise heading" to turn it off only in exercises. You can use it just for an individual picture by including it in the `stayfig` or `floatfig` environment. For example,

```
\begin{stayfig}
\nofnumber
\finput{FILE}{CAPTION}
\end{stayfig}
```

In addition, you can restrict its effect to any region of a file by enclosing it in braces ("{}").

Note: When there is no figure number, there is no way to reference the figure. So, although the labelling feature is not disabled by `\nofnumber`, you should not refer to that label.

Note: There is also a similar command called `\notnumber` which does the same for tables`\table number`.

10.4 Aligning Pictures and Tables

Although you can merely put `\finput` and `\tinput` commands next to eachother to get many combinations of pictures and tables on a line, this is not always satisfactory. The command `\valign` should help with this problem.

`\valign` (❖) takes a set of objects (pictures, tables, letters, words, whatever) and aligns them according to your specification on a single line. The basic use of the command is:

```
\valign{{obj1}{obj2}{obj3}....{objn}}
```

However there are two options which you have. You may specify "top" or "bottom" alignment (whereas the default is baseline alignment) with an argument in square brackets:

```
\valign[t]{{obj1}{obj2}{obj3}....{objn}}
or
\valign[b]{{obj1}{obj2}{obj3}....{objn}}
```

In addition, you can specify an amount to raise or lower each individual object in the list by putting the amount in square brackets before the object:

```
\valign{{obj1}[10pt]{obj2}{obj3}....[-3in]{objn}}
```

Special Features: The command `\valign` now has special features which allow you to force line breaks or alter the horizontal spacing.

- Instead of using several `\valign` commands in a row to get several different lines of objects, you can use the character / to indicate a newline. For example,
  `\valign{{obj1}{obj2}{obj3}/{obj4}{obj5}{obj6}/.../{obj?}{obj?+1}{obj?+2}}`

would result in a single vbox of lines with three objects aligned on each line. The `/` character can be used anyplace and any number of times. A vertical space of height `\vavskip` is added at each `/`.

- By default, `\valign` puts an `\hfil` around each object on a line, thereby "centering" them in their own space. However, you may want to group some objects together or force objects apart. To do this, you can use the symbol `*` to indicate where you want space. A line with at least one `*` on it will only have space where there are `*`'s. So for instance,
  `\valign{{obj1}*{obj2}}`
  will result in the two objects forced apart and
  `\valign{*{obj1}{obj2}*}`
  will result in the two objects forced together.

Examples: A common use of the command might be something like:

```
\valign[t]{{\finput*{fig1}}{\finput*{fig2}}}
```

or

```
\valign{{\finput{fig1}{caption}}[.5pc]{\tinput{table1}{caption}}}
```

However, here I will demonstrate the command with the following example:

```
\valign[t]{{a}[-3pt]{b}{c}{d}[5pt]{e}{f}{g}}
```

which produces:

a b c d e f g

If the optional "t" is replaced with a "b", we get:

a b c d e f g

New lines can be inserted with the / character, for example:

a b

c d e

f g

which is generated by the command

```
\valign[b]{{a}[-3pt]{b}/{c}{d}{e}/{f}{g}}
```

and space can be controlled with the character `*`:

```
\valign{{a}{b}{c}/{cow}{dog}/{e}{f}{g}/*{a}*{b}{c}*/
{cow}*{dog}/*{e}*{f}*{g}*}
```

produces

a b c

cow dog

e f g

a b c

cow dog

e f g

Note: The objects must be things which can be put in `\hbox`es.

`\vavskip` (⟷) is the vertical distance added by the / character for newline in `\valign` commands. By default it is .5 pica.

When you use the "corner label" argument of the `aspic` environment, you generally have many different picture which you wish to display together. However, they may be of drastically different sizes, and this makes it difficult to display them together in an attractive way. The command `\picarray` (picture array) is provided to help spread arrays of pictures into a "clean" looking lattice.

`\picarray` (❖) takes one argument, which is a list of objects (probably pictures, but they could be anything really). The entire list should be surrounded by a single set of "curly" brackets, and each element of the list is surrounded by square brackets, as shown:

```
\picarray{[obj1][obj2]....[objn]}
```

The objects are then displayed in a grid with as many objects on a line as possible (ASTEX determines this automatically) and spread out evenly. Most frequently, this command will probably be used to display sets of pictures over a single caption. An example of the syntax in this case is provided, along with its output which appears in Figure 10.9. (The TEX code for the preceeding reference says "`\ref{pa}`" because that is how the `\finput` command below labels it.)

```
\finput[pa]{\picarray{[\input{fig31a}][\input{fig31b}]
```

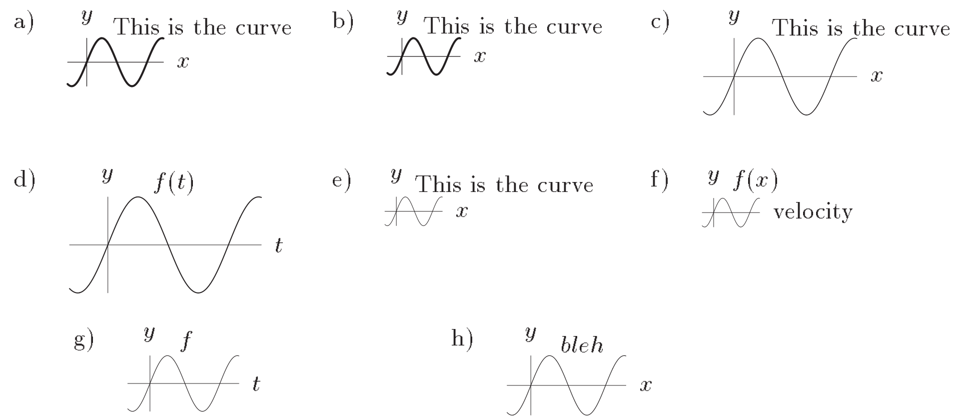


Figure 10.9: This is the caption under the big picture thing.

```
[\input{fig31c}][\input{fig31d}][\input{fig31e}]
[\input{fig31f}][\input{fig31g}][\input{fig31h}]}}{This
is the caption under the big picture thing.}
```

Alternatively, you can specify the number of pictures for `\picarray` to put on a line with an optional argument in square brackets. The syntax then becomes

```
\picarray[N]{[obj1][obj2]....[objn]}
```

where `N` is a positive integer. The advantage of *not* explicitly specifying the number of pictures per line is that the algorithm which `\picarray` usually uses to determine the number of picture per line will be affected by a change in page size or in the size of the pictures, whereas the specified value will not. On the other hand, `\picarray` may not make a choice which you consider aesthetic!

Note: Since the objects in the list are specified by square brackets, the command might get confused if the objects themselves contain square brackets. For instance, if one of the arguments is `[\cmd[op]{thing}]`, `\picarray` will take "`\cmd[op`" to be the first object! To solve this problem, merely enclose each object in an extra set of curly brackets:

```
\picarray{[{obj1}][{obj2}]...[{objn}]}
```

`\paskip` (↤→) is a dimension which determines the vertical space between lines in a `\picarray`. (The horizontal space is determined by an algorithm involving the width of the page and the width of the pictures.)

11 Graphics Commands

The graphics commands of ASTEX are designed to allow for *consistent* and high quality figures for a book, journal or article. Consistency, in fact, was a major factor in the creation of these commands. If you spend some time to learn to use them, you will see that the output is worth the extra effort.

The idea is the following: the picture will be a postscript file and the labels will be overlaid by TEX. The postscript files can be designed so that shading and line weights are consistent throughout the book. (This will be discussed further in the next section.) Furthermore, correctly using the commands below for labelling will guarantee consistent size and spacing of labels and figures.

The basic environment here is the `aspic` environtment (short for ASTEX picture). It is *basically* the same as the LATEX picture environment. That is, it is a two-dimensional coordinate system allowing you to specify an exact location to put an object. In fact, all commands which work within a LATEX picture environment will work here. (These commands include `\put`, `\line`, and the drawing commands from `epic` and `eepic`.) However, unlike the LATEX picture whose size is specified explicitly in the TEX code, the ASTEX picture environment will expand to include labels placed outside of the specified boundaries. This guarantees consistent positioning of the figure.

11.1 The ASTEX Picture Environment

`aspic` (`\begin`-`\end`) is essentially the same as the LATEX picture environment. It takes one two arguments, which are the width and height of the picture in `\unitlengths`. In addition, there are three optional arguments. The first scales the picture (to increase or decrease the size withouth affecting the line weights in the picture or the distances from labels to the object which they refer to. The other two relate to putting a label in the upperleft hand corner. The basic syntax is:

```
\begin{aspic}|S|(X,Y)

% commands here for generating picture and labels , such as \put,
% \putlabel, \putarrow, \hlength,....

\end{aspic}
```

where `X` and `Y` are positive numbers which represent the width and height (respectively) of the picture. The `|S|` is the optional argument determining the scale. If this argument is not present, the value 1 is used by default for `S`. (The role of this scaling constant will be discussed further below.) The main difference between this definition of `picture` and the LaTeX version is that the width and the height of the picture will automatically be extended.

Whereas in the LaTeX definition of `picture`, the size of the box is determined solely by the arguments `X` and `Y`, in ASTEX, it is more reasonable to think of `X` and `Y` as the *minimum* picture size. The commands inside the picture environment will put labels and arrows over the picture and may extend outside of the `X` $\times$ `Y` box which you have specified. If this occurs, the box will expand to include these objects. (New in version 3.0: You can specify that a picture should *not* resize by including the flag `\dontresize`. If this occurs in the preamble of the file then no pictures will be resized. Alternatively, you could localize it for a single picture environment by enclosing it and the picture environment in braces.

The optional argument identified by vertical bars ("`|`") is used to scale a picture. The argument between the bars is a positive number, which determines the `\unitlength`. In particular, the argument `|N|` sets the `\unitlength` equal to `N \basicunit`s. (See `\basicunit` below.) In general, it is expected that the argument will be 1 – meaning that the "coordinates" of the picture environment are based on the `\basicunit`. Consequently, this is the default value for the case that this argument is not specified.[4] However, `|.5|` would result in a picture which is half the size (actually, one quarter the area) of the same picture if the argument had been one. This kind of scaling can be done for several reasons: to increase or decrease the size of a picture which was generated at a size which is too large or small, to allow a figure to fit on a page for better pagination, to decrease the size of a picture so that it will fit next to another object, etc.

Another important feature of the scaling option is that it sends its argument to the postscript file in the form of a variable called `UnitScale`. In particular, it will define the postscript variable `UnitScale` to be the argument of this option. If you "massage" your postscript files appropriately (see section below) you will be able to use this in order to ensure that line weights do not change when a picture is scaled. That is, when an ordinary postscript picture is printed at twice its normal size, the line thicknesses

[4] In fact, earlier versions of this environment did not have this feature that the scale was determined by an optional argument. It is still possible to use this command without having the scale determined by an optional argument: merely use `nsuaspic` instead. (Stands for "not scaling unitlength".) This version will not understand an optional argument with vertical bars. In addition, if you do not set `\unitlength` manually when using `nsuaspic`, it will merely remain whatever it had been set to previously. This situation is a little "dangerous", which is what prompted me (at the suggestion of P. Blanchard) to make scaling an optional argument.

will double as well. A publication will not look professional if the line weights are not consistent from figure to figure. Therefore, it is useful to have postscript files in which the lineweights automatically scale relative to the variable `UnitScale` so that a picture scaled with the optional argument of `aspic` will maintain constant line weights.

Now, onto the other two optional arguments. Often, when several picture environments are placed together, you wish to have a label in the top left corner of the picture to indicate a specific picture. For example, you might label them (a) and (b). An option on the `picture` environment will automatically put such a label in the top left corner, so that it will be in the correct position even if labels (which might change the size of the picture) are added or removed. The syntax when using the options is

```
\begin{aspic}|S|<SPACE>[LABEL](X,Y)
% other commands
\end{aspic}
```

where `|S|` is optional as always, `SPACE` is a dimension specifying the amount of horizontal space between the label and the upperleft corner of the picture and `LABEL` is a label such as `a)` or `(ii)` or whatever you want to appear. If the `LABEL` is present without the `SPACE` then the dimension `\crnrlbldist` is used.

`\crnrlbldist` (⟷) is the horizonal space between a "corner label" and the top left corner in an `aspic` environment. A small distance such as 1 pica should be correct nearly all of the time, and so you can set this in your format and then only give the `SPACE` argument on the `aspic` environment on rare occasions.

`\unitlength` (⟷) is a LaTeX dimension which is used by the `aspic` (and `picture` environments as the length of the basis of the coordinate system. In ASTEX it is expected that this dimension will change from picture to picture (using the scaling option of the `aspic` environment) and is therefore not a dimension which you need to set in your format or defs file.

`\basicunit` (⟷) , on the other hand, will be constant throughout your files and should therefore be set in your format file. It is the unit which the `aspic` environment (and its scaling argument) is based on. In the United States (backwards country that it is), most people will probably use 1 inch as their "basic unit", so that they can label their figures in inches. Consequently, this is the default. However, setting this value to any other TeX dimension in your format file would allow you to label figures in your favorite unit.

11.2 Commands for Inside an `aspic` *Environment: Labelling a Figure*

`\loadps` (❖) is the command which is used to load an encapsulated postscript picture into ASTEX.[5] It takes one arguments (the name of the postscript file) and two optional arguments (the width and height to draw the picture). The syntax is

```
\loadps{FILENAME.eps}[WIDTH,HEIGHT]
```

where `FILENAME.eps` is the name of the postscript file (it doesn't *have* to end in `.eps`, it is merely an example). `WIDTH` and `HEIGHT` are positive numbers which are the width

[5] The command `\loadps` is actually calls the command `\psfig` by Trevor Darrell. It would be relatively simple to modify it to call another postscript loading command if you prefer.

and height of the figure in `\unitlength`s. Since the `\unitlength` is used automatically, this command will scale according to the scaling argument of an `aspic` environment. If the width and height are not given with the optional argument, they will be assumed to be the width and height of the aspic environment which the command is contained in (prior to any stretching). [If this command is used in a `picture` environment rather than an `aspic` environment, the size must be specified explicitly.]

In general, the width and height specified for the `\loadps` command will be the same as the width and height of the picture. In these cases, you will simply put the picture at the coordinate `(0,0)` in the picture using the LaTeX command `\put`:

```
\begin{aspic}(3,2)
\put(0,0){\loadps{sinecurve.ps}}
%
% other labelling commands
%
\end{aspic}
```

However, there are times at which you would want to specify the size at which the file should be loaded. For example, you might be including two postscript files in the same `aspic` environment:

```
\begin{aspic}(7,3)
\put(0,0){\loadps{pic1.eps}[3,3]}
\put(4,0){\loadps{pic2.eps}[3,3]}
\end{aspic}
```

Or, as is often the case, the postscript file itself might have lots of "blank space" around it, which you can cut off by positioning it appropriately and lowering the size of the `aspic` environment. Consider the following example of a picture which has one extra half inch of white space all around:

```
\begin{aspic}(3,2)
\put(-.5,-.5){\loadps{thepicture.ps}[4,3]}
\end{aspic}
```

Note: If you are doing this kind of "trimming" of white space, the flag `\showlabelbox` can be very useful. Furthermore, note that the shell scripts associated to the `PS` package automatically do this sort of white space trimming and use a method similar to the example above.

`\putlabel` (❖) is the basic command used for labelling a figure. It is assumed that a label will generally be referring to some element of a picture with a specific location. The command takes the location of this picture element (and *not* the location at which you want the label to appear). This ensures consistent spacing between the object being referred to and the label. The syntax is:

```
\putlabel(X,Y)[DIR]{LABEL}
```

where `X` and `Y` are the coordinates of the point to which the label refers, `DIR` indicates the direction of the point from the label and `LABEL` is the label itself.

The direction is specified by one or two letters: `c` for center, `t` for top, `b` for bottom, `l` for left, or `r` for right, and two letter combinations (such as `tr` with the `t` or `b` first) for the corners. The distance between the point of reference and the label is determined by the variables `\labspace` and `\slabspace`.

`\labspace` (|↔|) determines the space between a label and the point to which it is referring when the direction specifier is only one character long. (That is, when the direction is not diagonal.) A box around each label whose sides are exactly 2 `\labspace`s longer than the smallest box containing the label will just touch the reference point. To see this box (and better understand the role of `\labspace`, use the flag `\showlabelbox`.

`\slabspace` (|↔|) determines the space between a label and the point to which it is referring when the direction specifier is two characters long. (That is, when the direction is diagonal.) The corner of a box around each label whose sides are exaxtly 2 `\slabspace`s longer than the smallest box containing the label will just touch the reference point. To see this box, use the flag `\showlabelbox`.

`\putarrow` (❖) is a command similar to `\putlabel`, however in addition to putting a label in a position relative to a reference point, it puts an arrow between then. In particular,

```
\putarrow(X,Y)[DIR]{LABEL}
```

puts the `LABEL` and the tail end of an arrow pointing in direction `DIR` at the point `(X,Y)`. The length of the arrow is determined by the dimension `\@arrowlength`. An optional argument (in angular brackets between the word "putarrow" and the coordinates of the point) specifies any other length for the arrow.

`\hlength` (❖) is used for labelling the horizontal length between two points in a picture. The syntax is:

```
\hlength(X,Y)(X',Y)[DIR]{LABEL}
```

where (X,Y) is the coordinate of the leftmost point, and `(X',Y)` is the coordinate of the other point. `DIR` can be one of `t`, `T`, `c`, `b` or `B`. The direction and label can both be left off in order to have a length represented with no label. In this case, or if the direction specifier is `c`, the ends of the arrow occur exactly at the points specified. In any other case, the arrows are shifted so as to refer to the points from the given direction. By default, a small vertical line is placed at the end of each arrow. This line can be left off of either or both ends by using angular brackets rather than parentheses to specify the points.

An option (given by a star immediately following the command) exists which results in arrows pointing *in* from outside rather than out from inside. This is useful when the length you are representing is about the same size as the label. So, if the length is so short that the label covers the arrows, try using the "star" option.

`\vlength` (❖) is almost the same as `\hlength` above except that it deals with vertical lengths. The syntax is the same, except that the acceptable direction specifiers in this case are `l`, `L`, `c`, `r` or `R`.

`\rlength` (❖) is similar to `\vlength` and `\hlength`, except that it puts the bar on at arbitrary angles. Note that it does not take a label as an argument and only handles the bars and arrow. The syntax is:

```
\rlength[n](X,Y)(X',Y')<V,W>
```

where `<V,W>` is an optional argument for specifying the direction of the small end bars and `[n]` is an optional argument which determines whether the end bars are printed at all. If the optional argument in angular brackets is not used, the end bars are drawn perpendicular to the arrows, as is done in the case of `\vlength` and `\hlength`. The parameter `n` can be either 0, 1, 2 or nothing (the latter being the case where the optional argument is not present). If the optional argument is not present, then both end bars are drawn. If `n` = 0, then no end bars are drawn. If `n` = 1 then just the end bar at the point `(X,Y)` is drawn and if `n` = 2 then just the end bar at `(X',Y')` is drawn.

Note: If you want to put a label on an `\rlength`, merely use `\whiteout` and `\putlabel`.

`\rotatelabel` (❖) is used for putting labels on pictures at arbitrary angles. It has two different forms for its syntax:

```
\rotatelabel(X,Y)(X',Y')[DIR]{LABEL}
```

puts the `DIR` side of the `LABEL` at the point `(X,Y)` and then rotates it so that it heads towards the point `(X',Y')`. (Note that it may not reach that point or overshoot that point depending on the length of the label). If the optional argument `DIR` is not present, the left side of the label is used. An example of the use of this optional argument is that if `DIR` is `c`, then it is the center of the `LABEL` which is at the point `(X,Y)`.

Alternatively, one may specify a vector in place of the second point.

```
\rotatelabel(X,Y)<V,W>[DIR]{LABEL}
```

If a vector syntax is chosen, then the `LABEL` points in the direction given by the vector regardless of its location. (For example, the vector `<1,1>` means that the label should be rotate up by forty-five degrees regardless of its position. Whereas, if the label starts at the point `(0,1)` then no rotation at all is done to head to the point `(1,1)`.)

Note: The `aspic` environment does *not* expand to include rotate labels. This would have been too calculation intensive and slowed down the TEXing too much.

`\mline` (❖) is used for multiple-line labels. In each of the commands above, if the label is going to be more than one line long, you should use the command `\mline`. It has an optional argument (in square brackets) to specify left, center or right justification, and an argument which is the label using `\\` as a line break indicator. For example,

```
\putlabel(2,3)[br]{\mline[r]{This is the first line \\ This is the
second \\ ... \\ This is the last line of the label.}}
```

If the optional argument is not present, the label is left justified.

`\whiteout` (❖) is used for erasing part of the picture to make room for a label. You merely give it the label as an argument and it will print the label with a white box behind it. For example,

```
\putlabel(2,3)[br]{\whiteout{This is the label!}}
```

As with `\mline`, this can be used along with the label in *any* of the commands above. A small amount of space is added around the label. This space is determined by the dimension `\@whitespace`. An optional "star" after the command will eliminate this extra space (e.g. `\whiteout*{label}`), resulting in a white box just big enough for the label.

11.3 Flags to Help with Labelling

\showlabelbox (❖) is a flag which causes the box around labels and aspic environments to be visible. It is useful for ensuring that labels were placed correctly and that the box containing an aspic environment is the right size (but is hard to explain).

\showgrid (❖) is a flag which causes a grid (one tenth of a \unitlength square) to be laid over the picture. It is expected that this is how you will do your labelling. In particular, it you use \showgrid and then view a picture with a postscript previewer, you can determine the coorinates at which to place your labels. The graylevel of the grid is determined by postscript variables which are set in your format file. The amount which the grid extends beyond the aspic environment is determined by the variable \gridstretch (which must be a positive integer...larger integers lead to more grid). A symbol is placed at the origin (the coordinate $(0, 0)$) in the picture. You can change what symbol it is by defining the command \@originsym.

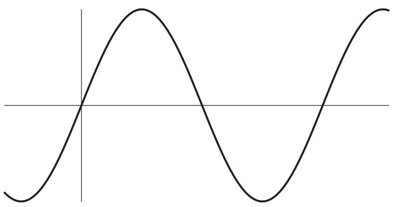
11.4 Examples of Graphics Commands

The aspic *Environment with a PostScript File*

Here is a very simple file which loads a postscript file and puts no labels on it:

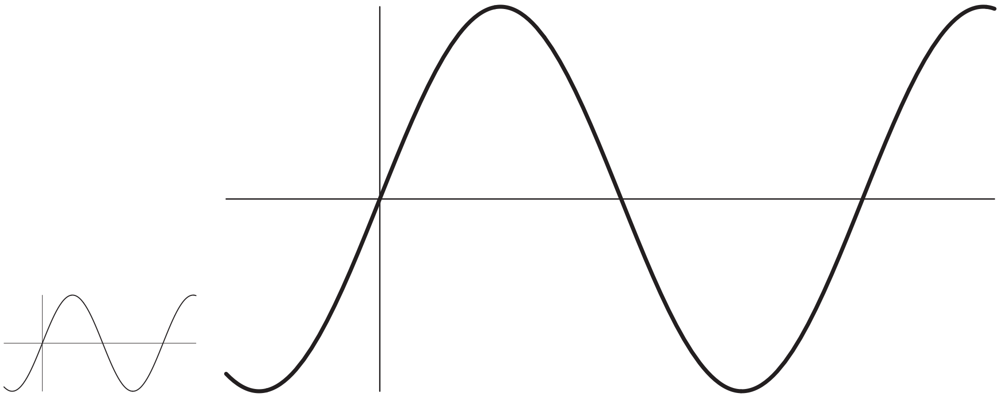
```
\begin{aspic}(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\end{aspic}
```

and this is what it produces:



If we include the optional *scaling* argument on the aspic environment, we can change the size of the picture but the line weights are kept the same. The preservation of line weights is something that I have done with *postscript* and not just TEX, although it is controlled by TEX commands. Below is the same picture with scaling constant .5 in the first case and 2 in the second.

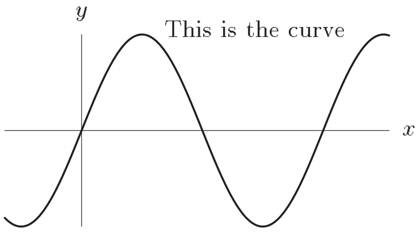
```
\begin{aspic}|.5|(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\end{aspic}

\begin{aspic}|2|(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\end{aspic}
```



Simple Labels

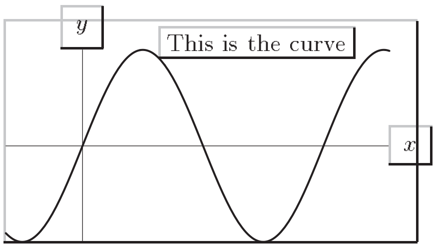
The basic labelling command is `\putlabel`. Here are some examples of labels on the previous picture, the commands which produce it are

```
\begin{aspic}(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\putlabel(2,.5)[l]{$x$}
\putlabel(.4,1)[b]{$y$}
\putlabel(.8,.95)[bl]{This is the curve}
\end{aspic}
```



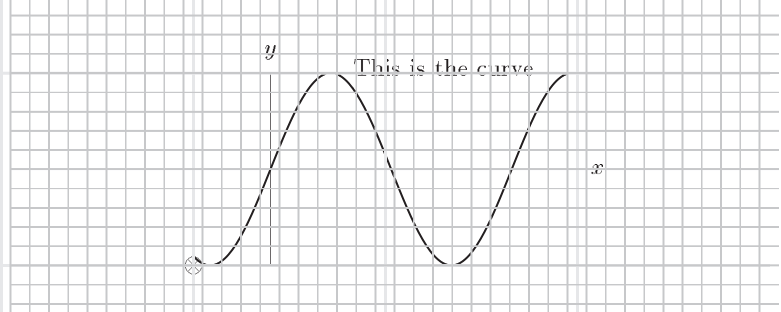
Two Flags

There are two *flags*, commands intended to go at the beginning of the file, which can help you label. Here I will show the results of each. First, the picture above with the flag `\showlabelbox`:



Notice that the box containing the whole picture stretches to include the x and y. Notice that the edges of the boxes just touch the point to which the label refers.

As you can see from the next figure, the edge or corner of the box exactly coincides with the coordinate which I gave the `\putlabel` command. Here, we see the way the picture would look with the command `\showgrid`. Notice that the grid extends beyond the boundary of the picture. You can control how much it sticks out with the variable `\gridstretch`.

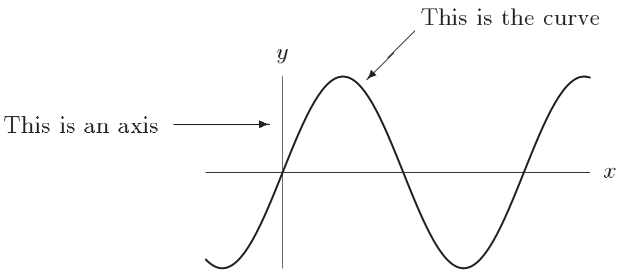


In addition to being able to control how much the grid sticks out, you can alter the darkness of the grid with the postscript variables `showgridbig` and `showgridsmall`, which are defined in set in the format file, and you can change the symbol which is placed at the origin in the grid (in this case it is $\otimes$) by defining `\@originsym` in your defs file.

Arrows and Lengths

The command `\putarrow` is used to point to things and label them. Here are some examples:

```
\begin{aspic}(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\putlabel(2,.5)[l]{$x$}
\putlabel(.4,1)[b]{$y$}
\putarrow(.8,.95)[bl]{This is the curve}
\putarrow<.5in>(.4,.75)[r]{This is an axis}
\end{aspic}
```



Notice that in one case I merely changed a `\putlabel` in to a `\putarrow`. In that case, I do not specify a length for the arrow and the *default* length is taken. Just as the size of the box expanded to fit the `\putlabels`, it expands for these `\putarrows`.

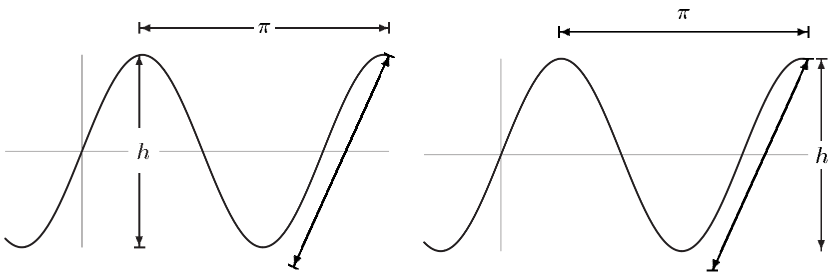
To represent lengths in a picture, you use length bars. Here, they are given by the commands `\vlength`, `\hlength` and `\rlength`. Consider the following two examples:

```
\begin{aspic}(2.00,1.00)
```

```
\put(0,0){\loadps{laser.ps}}
\vlength(.7,0)<.7,1>[c]{\whiteout{$h$}}
\hlength(.7,1)(2,1)[b]{$\pi$}
\rlength(1.5,-.1)(2,1)
\end{aspic}
```

and

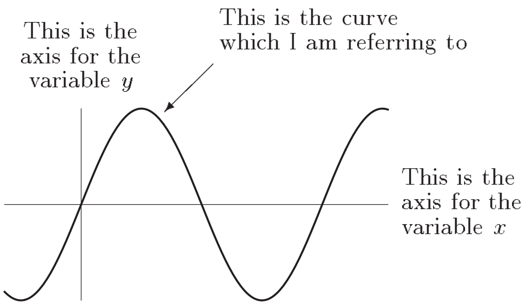
```
\begin{aspic}(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\vlength<2,0>(2,1)[l]{$h$}
\hlength(.7,1)(2,1)[B]{$\pi$}
\rlength<1,0>(1.5,-.1)(2,1)
\end{aspic}
```



Multi-Line Labels

In any of the commands described here, you can use a *multi-line* label, if you use the `\mline` command where you put the label. Here are examples:

```
\begin{aspic}(2.00,1.00)
\put(0,0){\loadps{laser.ps}}
\putlabel(2,.5)[l]{\mline{This is the\\axis for the\\variable $x$}}
\putlabel(.4,1)[b]{\mline[c]{This is the\\axis for the\\variable $y$}}
\putarrow(.8,.95)[bl]{\mline{This is the curve\\which I am referring to}}
\end{aspic}
```



The Input Command

If you put the TEX commands above into a file called `fig1.tex`, then you merely need to say `\finput{fig1}{Notice that the width of the caption is automatically determined by the width of the picture}` to get the following output

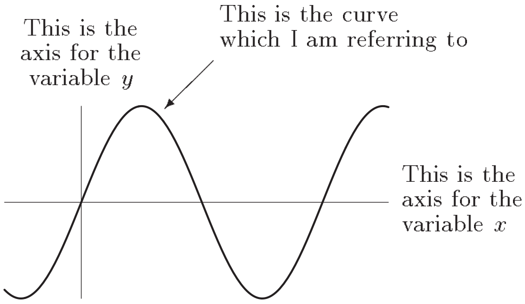


Figure 11.10: Notice that the width of the caption is automatically determined by the width of the picture

Now I can refer to that figure using the command `\ref{fig1}`, as shown here: Figure 11.10.

I can put two figures next to each other, if they fit, merely by using two sequential `\finput` commands:

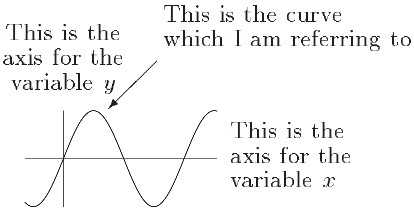


Figure 11.11: These figures have are half sized.

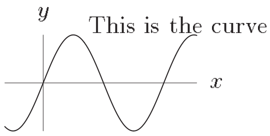


Figure 11.12: But it was very easy to put them next to each other like this.

12 Postscript

Most computer users are familiar with the term "postscript" from such familiar phrases as "postscript printer" or "postscript font". However, they may not be aware that postscript is actually just another computer language. Just like TEX, it is a language which is designed for the production of printed matter, so most of its commands relate to printing words or graphics. A postscript file can be read and modified with any reasonable editor, and you can write "macros" and commands in postscript just as you can in TEX.

The direct output of TEX is in the form of a "dvi" file ("device independent"). However, I am not aware of any printers which can print a dvi file if you send it directly. There certainly are some previewers which allow you to view a dvi file. However, in most situations, a driver is used to convert the dvi file to some other format. One popular driver is *dvips*, which converts a dvi file into a postscript file. In fact, dvips is more versatile than merely that, since it allows you to add your own commands to the postscript by using the `\special` TEX command. It is quite likely that even if you are not aware of it, your computer system runs dvips on dvi files before printing or viewing them. (Often, a simple program is used which performs some combination of running TEX, running dvips and sending the file to the printer, so users are unaware that this is happening.) However, it is certainly possible that you do not use dvips, or even postscript, when printing a TEX file.

ASTEX makes use of the `\special` command and dvips in order to achieve some of the things which would not be possible with TEX alone. In particular, the graphics commands and color separation commands frequently use postscript definitions.[6] If you are not using postscript (or dvips) you will probably want to use the flag `\nopostscript` to prevent the postscript commands from being used.

12.1 Using ASTEX Postscript Commands in Your Graphics

There are many programs which can be used to produce postscript figures which you might want to use in your document. Using the commands described in Section 11.1 you can load these graphics into your file and label them. However, two features of ASTEX will require you to actually modify the output of those programs. In particular, by using ASTEX and "massaging" the postscript files appropriately (i.e., making small changes) you can guaranty consistency in line weight and shading throughout your document and use the ASTEX color commands. (The color commands will be discussed further in Section 12.3.)

The postscript commands which ASTEX will have to have slightly different behavior in different types of files. That is, the command for "color" or "line of axisweight" will not behave the same in a file produced by Fig as a file produced by Mathematica. Consequently, each type of postscript file which you intend to load as graphics into ASTEX will need to "set the swithes" appropriately. For example, the command `Mswitches` should be put at the beginning of every Mathematica postscript file to indicate that the file is a Mathematica file. The way to define and set switches will be discussed later so that I may first introduce the commands themselves.

12.2 Fixed Line Weights and Shades

When a postscript file draws a line, the command `setlinewidth` is called, and its argument is a number which will determine the width of the line. Similarly, the command `setgray` is used to determine how dark an object which is drawn will be and its argument is just a number. The idea, then, is to use a *variable* as the argument of these commands. The variable will be set in your format file, and so, not only will it guarantee consistency in all documents printed with the same format, but it provides a simple way to change all of the figures without redrawing them. For instance, it might turn out that your light shades are too light to be seen. With this method, you do not need to redraw all of the figures or edit any of the postscript files, merely change the variable in the format file.

In ASTEX, there are three line weights which you must set: `\gridweight`, `\axisweight` and `\curveweight`. You set these in your format file by using `\def` to define them to

[6] It *might* be possible for you to use some of the graphics commands even if you are not using postscript, but the color separation commands *certainly* will not work.

be the weight which you want in points. So, for instance, reasonable settings might be:

```
\def\gridweight{.1}
\def\axisweight{.5}
\def\curveweight{1}
```

These, in turn, define postscript variables of the same names which are not simply constants, but are defined to scale appropriately with the scaling constant of the `aspic` environment. For example, if you define `\curveweight` to be 1 as above, then when you use the *postscript* variable `curveweight` to specify a line weight, it will be 1 if the picture is printed at normal size, but 2 if the picture is printed at half size so that it still appears to be 1 point wide!

If you are using Mathematica to generate figures, then it is highly recommended that you install the `PS` package. This is a set of shell scripts (running on Unix systems) which take the postscript output of Mathematica and automatically insert the ASTEX postscript variables in the correct places. The idea is relatively simple. I describe it here both so that you understand how `PS` works and so that you can write scripts of your own. Further information can be found in the README files which are supplied with `PS` on the World Wide Web or FTP (see "Getting the Files" above).

If you choose "code" line weights, such as 1 for gridweight, 2 for axisweight and 3 for curveweight, then you can use the code line weight in the program which generates the postscript and "massage" it by replacing the code with the correct variable. To do this, you need to figure out how the postscript from your program changes the line weight. Simple experiments, in which you draw the same picture but vary the line weights and compare the postscript, should help you determine this. Many programs use the command `setlineweight`. In that case, you need only write a script which replaces "`1 setlineweight`" with "`gridweight setlineweight`" and so on. Mathematica uses the command `Mabswid` to set line weights (when the weight is specified in points). So, your shell script to modify Mathematica postscript would replace "`1 Mabswid`" with "`gridweight Mabswid`".

Similarly, there are three "gray levels" which are stored as variables: `lightshade`, `medshade` and `darkshade`. Set these in your format file and then come up with a code in the postscript so that you can specify these shades by some "code" shade.

Please get a copy of `PS` and its documentation for further information and suggestions.

12.3 Colors and Color Separation

This is the part of ASTEX which has changed the most in this new version and it is why I am only calling it a β version at the moment. The color routines in ASTEX are now completely different than they were before. In particular, they now use standard postscript color commands. The advantages of this are: (a) it is what other people are more used to (b) you can see the colors on a previewer or printer (c) since this new routine uses the "color stack" it is compatible with packages like `graphicx` and more versatile, and (d) you can use graphics with color commands made by most standard programs. The disadvantages are (a) you can no longer produce real color separations by yourself [I do not know how color separations are produced by these new commands!] and (b) in order to continue to be compatible with OLD files I've had to include some hacks. In particular, Dave Richards has written a marvelous patch for allowing old postscript files to be used with the new ASTEX and I include this as part of the package in `\psmathdefs`. So, I'm really not happy with this and don't have time to write about it, but this is how it stands!

Color Commands in TEX

Textual elements, such as certain headings or TEX lines, will be printed in color in additions to graphics objects in figures. Two simple commands can be used to "colorize" a TEX object.

`\colorize` (❖) is the basic command which you will use to make colored text in your book. The command takes one argument and prints it in color rather than black. E.g:

```
\colorize{This is in color.}
```

Unlike the old versions of these commands, due to the color stack this command now *does* work "across" page breaks. So if a page break occurs before the words "in color" in the example above, these words will be printed in color as one would hope.

`\color` (❖) is a command like `\bf` or \, which is contained by TEX braces. For example,

```
This is in black.  {\color This is in color.}  This is in black.
```

Moreover, if you are doing some complicated programming and want to use color, then you should be aware of the *color stack*. There are three commands you need to know for this: `\pushblack`, `\pushcolor` and `\restorecolor`.

`\pushblack` (❖) This command switches the printing color to black and takes an optional argument of a shade as a number between 0 and 1. It does not respect TEX groups or environments.

`\pushcolor` (❖) This command switches the printing color to "color" and takes an optional argument of a shade as a number between 0 and 1. It does not respect TEX groups or environments.

`\restorecolor` (❖) This command should always be called after one of the previous two in order to switch the color back to what it was previously.

You use the first two commands to change to a new color and the last to return to a previous color. The idea is that the program `dvips` will keep track of what colors you have been using and so you can always go back to the last color with `\restorecolor`. For example,

```
This is in black \pushcolor this is in color \restorecolor and this is
in black again.
```

(In fact the `\color` command is nothing other than `\pushcolor` which automatically puts a `\restorecoor` when you close the present group!) Finally, you should also know that `\pushblack` and `\pushcolor` take an optional argument of a number from 0 to 1 and also control shading:

```
This is in black.
\pushcolor[.5] This is in a shaded sort of color.
\pushblack[.2] This is a pale gray.
\restorecolor So this is back in the shaded color and
\restorecolor this is back in black!
```

If you say `\pushcolor[0]` this is the same as saying "print in white" and if you say `\pushcolor[1]` this is the same as `\pushcolor`. (See also `\grey` and `\white` which do similar things.)

Note: Be sure to use `\colorrestore`...you don't want to overload the stack.

Note: If you want some font to always be in color, for example the section heading font `\secfont`, you merely put `\color` in its definition. For example, the following two lines demonstrate how to have the figure legend (`\figlegfont`) and section headings (`\secfont`) in color:

```
\nameasfont*{figlegfont}{\bf\color}
\nameasfont{secfont}{rphvr}{24}<\color>
```

`\cfbox` (❖) is a box (just like the LaTeX command `\fbox`), however the frame appears in color rather than black.

The following two commands allow you to colorize the lines in tables and over footnotes. By default, they are defined to be `\relax`. However, if you include a line such as:

```
\def\tablecolor{\pushcolor}
```

in your format file then the table lines will be in color.

`\tablecolor` (❖) if this command is set to `\pushcolor` then the lines in a LaTeX `tabular` environment will be printed in color.

`\footnotecolor` (❖) if this command is set to `\pushcolor` then the line associated to a footnote will be printed in color.

The "Ink Color"

You can set the "ink color" with the command `\colorink` in your format file or at the beginning of a file. This means that when you print or preview the file, you can have the color ink look however you want.

`\colorink` (❖) sets the values of Cyan, Magenta, Yellow and Black in the color ink. The command is used like:

```
\colorink(a,b,c,d)
```

where a, b, c and d are numbers between 0 and 1. The larger the number the more of the corresponding ink you will get.

This even be used for a cheap kind of color separation. For example, if you want to only see the black ink you can say `\colorink(0,0,0,0)` and similarly you could set the black ink color with the undocumented command `\blackink`.

13 Appendices

A External Shell Scripts

Although none are provided with ASTEX, shell scripts can greatly increase your efficiency. By a shell script, I mean a simple program written in your computer's operating system to manipulate the files you produce. Below is a description of shell scripts which

I wrote in Unix for use with this package (in earlier incarnations). You should be able to write scripts on your own system which perform similar functions. In addition, at a later point, I intend to "generalize" some of the scripts which I wrote previously and distribute them with ASTEX.

PS

Note: If you are using `PS` to allow for the use of Mathematica files in ASTEX then you should call the command `\psmathdefs` in your format file. This command inserts all of the definitions for mathematica postscript commands *and* inserts Dave Richards' patch for the old color separation commands which are obsolete but have not really been replaced.

At this point, the only one of the scripts I am talking about here which I am making publicly available is `PS`. The reason is that it is by far the most valuable time saving tool of these scripts. This package takes the postscript output of Mathematica and turns it into suitable input to ASTEX. In particular, the program:

1. Determines the smallest `aspic` environment which will contain the picture. (Eliminating excess "white space".)
2. Inserts ASTEX color commands `ascolor` and `asblack` in appropriate places in the file so that it automatically color separates.
3. Replaces explicitly states line widths and shades witht the postscript variables which are set in the format file. (In this way, line weights scale according to the `aspic` environment and are consistent throughout the document.)
4. Makes a TEX file which contains an `aspic` environment of the correct size which loads the picture and puts all of the labels (which Mathematica had included in the postscript) in `\putlabel` commands.
5. Puts labels on the axes in two dimensional graphs.

   and
6. Removes all of the labels from the postscript file, so that only the ASTEX labels remain.

This, in fact, is only the "basic" operation. It was also programmed to perform more complicated tasks such as labelling axes in three dimensional graphics and putting TEX arrowheads on the ends of curves or vectors.

The idea behind the operation of the program is quite simple. "Coded" information is passed to the script in terms of the line weights and shades which are set in the postscript file. Using this information, the program uses the editor `ed` to insert the correct commands and remove the correct labels.

Note that there is nothing particularly special about Mathematica that makes this script work. Similar scripts could be written for the postscript output of any program which creates graphics that you might want to use.

As stated above, this package should be publicly available and comes with its own documentation.

Naming Files

It is very important to come up with consistent naming schemes for your files. If there is a simple rule which determines the name of a picture or exercise file, then shell scripts can do a lot of work which would otherwise take up your time. One useful shell script, then, is one to help you name files.

If you are going to have a lot of figure or exercise files, write a script that automatically names the next one appropriately. Then, when all of the files are named appropriately, you can write more complicated scripts like those described in the next section.

Renaming Files

When producing a textbook, you generally print out several versions of the entire book (working copies) before it ever actually gets published. I suggest that after each of these printings, you rename all of the files (picture and exercise) which are used in the book so as to immediately identify their position in the latest version. For example, `fig3f7` would be the seventh figure in chapter 3! This serves two important purposes: it helps you find files quickly without needing to think to hard about its name and it helps with "cleaning", since any file which is not in the book will not be renamed. These "lost" files should be deleted, moved to a storage directory or put into the book.

In support to this renaming script, you should have a program which stores the name of files before and after renaming. You may find that in some instances you have to find a file by its *old* name. In these situations it helps to have a program (called something like "changed") which will give you the new name from an old name or vice versa.

TEXing Files

A useful shell script for anyone using LATEX would be one that runs LATEX on a file, re-runs it if the cross-references are not right, and then offers a number of choices including: printing, previewing, rerunning LATEX, viewing the log file, etc. One nice thing about such a script is that it can check first to make sure that a file exists with the name you requested and thereby avoid getting a lot of "`texput`" files created by TEX.

Either within that program or as a separate program, it is useful to search the log file for the word "warning". In Unix, this can be done simply with the command `grep -i warning FILE.log`. Both ASTEX and LATEX often send warnings to the log file and to the screen which go by too quickly to be noticed during TEXing.

Checking Correctness

There are many instances in which you might want a shell script to make sure that you are doing things correctly. One such instance is a shell script that would check to make sure that each exercise file is input once and only once into an exercise list. Or, you might need a modified spelling checker that suits your particular situation. For example, if you are using the `\suffix` to produce different versions of your text book in different parts of the world where spelling is different, your spelling checker could check each version "separately" even though they are part of the same file. (I actually did this for one book in which both British and American spellings of words were included for two different editions; a spelling checker written for this occasion checked each version separately. It would complain about spellings which were uniquely American if they appeared in the British version and vice versa.)

B *For Users of* `newcalcstyle` *and* `intlpress`

Here, briefly, are some of the major differences between ASTEX and the earlier incarnations of this macro package:

- Perhaps the most significant thing is that all of the people who are using my macros will have the same package now. So, if I fix any bugs or improve anything, you can all get a new copy of `astex.sty` but keep your individual format files and such.

- Although this may not be very useful to those users whose "specs" are already set, it is much easier to change the layout and format with ASTEX. In many cases, you have "multiple choices" of behaviour or appearance to choose from. For example, the appearance of section headings, whether the figure is above or below the caption, how the running head is positioned, and more are all determined by these "multiple choice" questions. Furthermore, all of the things you are likely to want to change are in a *format file*, separate from all of the actual program. So, a casual user is more likely to be able to make changes and does not have to look at my "code".
- Many "appendices" have been removed. That is, a lot of things which no longer had a useful purpose but still existed have been eliminated. For example, `\putcircle`, `\bb`, `\ee`, `\setunits` and `\iohead` are gone. Similarly, the arguments of many commands have become *optional*. For example, the size command on a `\loadps` command is now optional, since it is almost always the same size as the picture environment.
- The `picture` environment has been replaced with the `aspic` environment. (Now the `picture` environment is the usual LaTeX one.) The `aspic` environment is mostly the same as our old `picture` environment. The big difference is that instead of the `\setunits` command, there is an optional scaling argument on the environment itself. If the argument is *not* there, it assumes "`\setunits{1}`". Within the environment itself, `\newputlabel` has been replaced with `\putlabel`.
- The `\finput` command now has a "table" equivalent called `\tinput`. (I know that such a command already existed in Arizona.) Both of these commands have an alternate form in which the commands to create the figure/table can be given explicitly rather than loaded from a separate file. There is an argument to control the width of the caption, although it is still the width of the figure/table by default.
- The color separation commands have been "consolidated" into only two commands: `ascolor` and `asblack`. These replace the variety of color separation commands that used to exist. Instead of having lots of different commands for each type of postscript file (Mathematica, Fig, TEX, etc.) there is a TEX command (`\PSswitches`) which controls "switches" in these two color commands. In this way, a user who doesn't know how to program postscript can make new postscript color commands for use with ASTEX.
- All of the variables which used to be controlled from postscript files (such as `figstyle.ps`) are now set from TEX in the format file.
- I have severed some necessary ties to other programs. In particular, we no longer need to use a modified version of `psfig` or of the Mathematica postscript commands. The good point is that as these programs change (which they seem to do often) you can just update to the new one without worrying about having to change ASTEX. (However, the package `PS` which modifies postscript will still have to be updated for new versions of Mathematica.) As an example, it *used* to be the case that the Mathematica command for setting line weights was modified (by me) to keep line weights constant under scaling. This is no longer the case – now it is the variables `curveweight` and so on that scale. Furthermore, ASTEX is not dependent on `pageframe` as were the other packages.

- `\exercises` instead of `\io`.
- No more shell scripts for making table of contents, index, all iofiles, etc. This is all done from TEX. Also, page numbers get set automatically (without using `\autopage`).
- Display objects (i.e. tables and figures) next to eachother or in an array with `\valign` and `\picarray`.
- Compatablility with LATEX2e and AMS-LATEX.
- Use `boxeqnarray` to put `eqnarray`'s next to item numbers or solution heads.
- `floatfig` instead of `fakefig` is almost the same, but will not float above section headings the way that `fakefig` did. There are a few extra parameters which you can set to control the behavior of `floatfig`.
- I have replaced `pageframe` with standard ASTEX commands. This is more versatile (because I can make it do anything we want) and more safe. In particular, `pageframe` modified part of the output routine and was therefore not compatible with AMS-LATEX or LATEX2e.
- Sections should (or atleast *can*) be input into the chapter file with the `\secinput` command. The advantage of this is that two different users can edit sections in the same chapter at the same time without "stepping on each other's toes." Furthermore, you can easily TEX just a few sections in a chapter by using the `\justsec` command.
- The command `asnewlabel` is used in the `.aux` file to indicate not only what a label is but what kind of file it is associated with. This is useful for writing shell scripts that do things (such as rename) figures, problems, sections, etc., since they can just `grep` the `.aux` file to find them.
- A command, `\retrodefs`, exists for use in your defs file which will "undo" a lot (but not all) of the changes, so that you can use old files. IF YOU WANT TO USE SOME OLD COMMANDS (such as `\chaplabel`, `\widefinput`, etc.) PUT THE COMMAND `\retrodefs` IN YOUR DEFS FILE!
- The sectioning commands can be put into uppercase automatically using things like `\section@uc`, and lowercase letters are specified by `\lowercase`.
- The "claim" environment is much smarter now. It doesn't mind if there are blank lines before the proof starts and it allows you to use something other than the word "proof" if you want.
- Bibliography references work better now and allow references to ranges of entries with `--`.

C Instructions for Authors of Journal and Proceedings Submissions

This section is intended for those authors who wish to submit an article in TEX form to a publisher who uses ASTEX. ASTEX is a LATEX style file. To write a document in ASTEX, you will need (at least) the file `astex.sty`, which should be made available to you by the publisher or is available by anonymous FTP.

Here is a sample of the basic structure of the file which you should create:

```
\documentstyle{astex}

\asformat{DEFAULT}
```

```
\begin{document}

\title{***The title of your paper here***}

\author{***author1 name here***}[***address here***][***and here***]

\author{***author2 name here***}[***address here***][***and here***]

\abstract{***The abstract of the paper***}

\section{***Section title***}

***Text here***
.
.
.
\begin{bibliography}
\bibitem[***]{***}
\bibitem[***]{***}
.
.
.
\end{bibliography}
\end{document}
```

If you have the style file `astex.sty` in the same directory as a file of the form above, you should be able to process the document with LaTeX.

If you want to write your own definitions, please add them between the `\begin{document}` and the `\title`. Use the LaTeX command `\newcommand` to define things so that you don't accidentally erase a preexisting command that is important.

If you want to use AMS-LaTeX, put the command `\amscompatability` after the `\documentstyle` command and before the `\asformat` command. Then, use the command `amslatex` rather than simply `latex` to process the document.

Following is a list of the commands you will probably want to use in your article. Find them in the index and read about their usage, or look at how they are used in the supplied examples.

- `\title`, `\abstract` and `\author` to start the article off.
- `\stateclaim`, `\statetheorem`, `\statecorollary`, etc. to state a claim without a proof.
- The `claim`, `theorem`, `corollary`, etc. environment to make a claim and follow it with a proof.
- `\bibref` to refer to items in the bibliography and `\bibitem` to make items in the bibliography.
- `\section`, `\subsection` and `\subsubsection` to divide your article into sections.

  In addition, the following LaTeX commands, which are not described in these instructions but can be found in any LaTeX manual, will be of use: