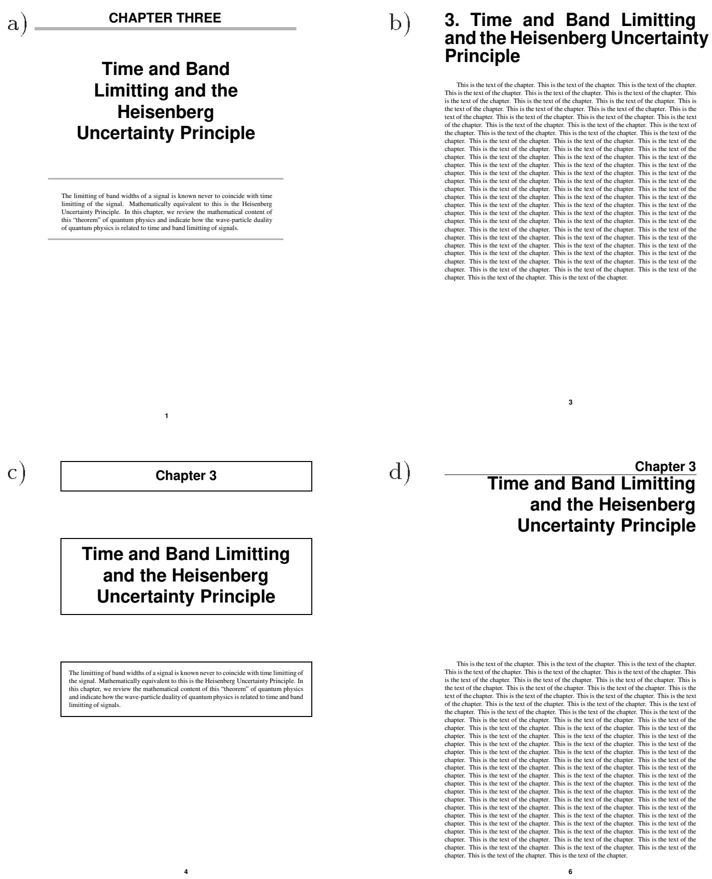


Figure 7.6: These are `\chapheada`-`\chapheadd`. In addition, `\chapheade` looks like the cover page of these instructions.

and the other "Review Problems for Chapter 7". You name and define your headings in the format file with the command `\newexerhead`. In particular, you say

```
\newexerhead{TYPE}{DEFINITION}
```

where `TYPE` is your name for the type of head you are describing and `DEFINITION` are the commands that will create the heading you want. You will probably want to use commands like `\thesection`, `\arabic{section}`, or `\thechapter` in your definition so that the heading depends on the position in the book. The heading is automatically put on the exercises by the `\exercises` command.

New in version 3.0: In addition to exercise headings, you may want to call some commands to affect how the different types of exercises look. For example, suppose you have two different types of exercise headings: `DEF` and `MISC`. (Here I imagine that most of the exercises are of type `DEF` but the ones at the end of a chapter are different, they are `MISC`. And suppose that you want the exercise numbers in the `MISC` problems to be printed in bold. Then you need to have access to a command that gets called for `MISC` exercises but not for other ones! This is what is provided by the new command `\exerxtra`. If you define the command `\exerxtraMISC` in your defs file, then it will be called at the beginning of any `MISC` exercises. So, for instance, if you say

```
\def\exerxtraMISC{\def\labelenumi{{\bf \arabic{enumi}.}}}
```

then the problem numbers of only `MISC` problems will be in bold.

The Structure of a Chapter File

The chapter file itself only needs to contain the `\chapter` command and `\secinput` commands to input the sections. In particular, it will likely look like

```
\documentstyle{astex}
\asformat{WHATEVER}
\begin{document}
\chapter{9}{The Title of the Chapter}
\secinput{mv-sec}{The Title of the First Section}
.
.
.
\secinput{Bleh-sec}{The Title of the Last Section}
\end{document}
```

`\chapter` (❖) is the command for beginning a chapter. As in LaTeX, this command takes the title of the chapter as an argument and makes the chapter heading. However, since each chapter here is expected to be in a separate file, in this case you must also indicate the chapter number. The syntax of the command is `\chapter{NUM}{TITLE}` where `NUM` is the number of the chapter and `TITLE` is its title.

`\secinput` (❖) is a command for inputting each section of the chapter as an individual file. It takes two arguments, the name of the file and the title of the section. It will use the name of the section as a label. Thus, if you want to refer to a section number, you do not need to come up with a new label. The syntax, then, is simply

```
\secinput{FILE}{TITLE OF THE SECTION}
```

If you use the command `\justsec` to specify certain sections to load, then not every section will be input.

This command actually prints the heading of the section (using the `\section` command just as you would) and so you do not need to have a separate `\section` command.

`\justsec` (❖) takes one argument, which is the name of a section file that you want to input. If this command does not appear, then the command `\secinput` will actually input every file it is given. However, when at least one `\justsec` appears then `\secinput` will only input files whose names appear as arguments of a `\justsec` command. So, for instance, in the following example:

```
\documentstyle{astex}
\asformat{OURBOOK}
\justsec{sec-cows}
\justsec{sec-chickens}
\begin{document}
\chapter{3}{Farm Animals in Modern Society}

\secinput{sec-goats}{The Goat in the Workplace}
\secinput{sec-cows}{The Cow that Jumped over the Moon}
\secinput{sec-horses}{The True Meaning of Horse Power}
```

```
\secinput{sec-chickens}{Which came first...?}
\secinput{sec-conclusion}{Conclusion}

\end{document}
```

only the second and fourth sections will be included. Note that it doesn't really matter where the `\justsec` command occurs as long as it occurs before the `\secinput` command. However, it is a good idea to keep them all together so that you know where they are. (It could lead to missing sections if you forget that you have some `\justsec` commands around.) Note also that the names of the files above (`sec-goats.tex`, `sec-cows.tex`, etc.) are merely examples. There is no fixed prefix which you must use for section file names.

Making Appendices Instead of Chapters

`\appendices` (❖) is a flag which throws a switch (called `@chapapp`) to indicate that appendices will be made instead of chapters and also sets the counter command `\thechapter` to print the chapter number as a capital letter. Furthermore, all of the supplied `\chapheads` are set to recognize the `@chapapp` switch. The point of all of this is,

```
\appendices
\chapter{3}{This is the title of an Appendix}
```

will create a heading for **Appendix C** rather than for **Chapter 3**. If you write your own `\chaphead` command rather than using one of the supplied samples, you may want to have it recognize this switch as well.

7.3 Commands for Exercises

In general, one would expect a set of exercises to occur at the end of each section of the book. The way this is done in ASTEX is that a single command, the `\exercises` command, appears in the chapter file itself. This loads a separate file, the *exercise list*, that lists the names of the exercises in order and also "formats" them. A separate file is then needed for each exercise. This file contains not only the question which is to appear in the book, but also the solution which will appear in the student (or teacher's) solution manual. It was done this way because exercises are frequently changed and moved when working on a textbook and this method minimizes the risk of a mistake occuring during such changes.

`\exercises` (❖) This command loads the exercises and should be placed at the position in the chapter where you want the exercises to appear. In general, it is placed at the end of each section of the chapter. The syntax for the command is

```
\exercises<FILE>[HEADTYPE](NUMCOLS)
```

where `ex-FILE.tex` is the name of the exercise list, `HEADTYPE` is the type of exercise heading and `NUMCOLS` is the number of columns in which the exercises will be typeset. All of these arguments are optional; in any case a default is chosen if no option is specified. The default for `FILE` is the number of the current section. The default for `HEADTYPE` is `DEF`. The default for `NUMCOLS` is the value of the variable `\excols` (set in your format file) which has a "factory setting" of 1. Thus, if you always use the same number of columns in exercises, your standard exercise heading is of type `DEF` and your exercise lists are named `ex-1.tex`, `ex-2.tex`, etc. where the number is the section number, then the `\exercises` command with no argument can be used in almost all cases.

Note: The choice of "`ex-`" as the prefix used for the exercise lists is the default, as well as my suggestion. However, you are not required to use this. Merely define `\exlistprefix` to be whatever you wish to use as a specifier for these filenames.

Note: When the number of columns is specified to be greater than one, the command `\Ncolumns` is used to set the exercises in columns.

`\excols` (❖) is a variable (set in the format file) which determines the number of columns in which the exercises are to be set by default. It should be a positive integer.

`\qinput` (❖) This command is the most basic command for use in an exercise list. It is an "input" command which loads question file. It automatically numbers the question, loads the question and labels the question. Its syntax is simply

```
\qinput{QUESTION1}
\qinput{QUESTION2}
.
.
.
```

where `QUESTION1.TEX` etc. are files containing the questions. A question can be referred to merely by saying `\ref{QUESTION2}`, for example, since `\qinput` automatically labels the question.

This command is smart enough to know when the question being input is "empty" (i.e., prints nothing). In that case, it will make a note of this and skip this question the next time through. This is particularly useful in the solutions where you may want to have ASTEXignore the questions which have not or cannot be answered.

`\instructions` (❖) is the command for inserting instructions for a set of questions. It appears in the exercise list file exactly where you intend the instructions to appear. Within this command, it is useful to refer to questions by their file name. The syntax is demonstrated by the following example:

```
\qinput{question1}
\qinput{question2}

\instructions{In problems \ref{question3}-\question{10}, perform the
given integration by applying the Fundamental Theorem of Calculus}
\qinput{question3}
.
.
.
\qinput{question10}
.
.
.
```

There is no restriction on where nor how many instructions can appear in an exercise list. Most frequently, this command is used with the environment `qcolumns` when the questions which follow the instructions are short. (Note that you may want to use `qset` instead of the `\instructions` command. That alternative method of including instructions warns the user if questions are out of place.)

There are four commands you can use to alter the appearance of the instructions: `\inspullback`, `\pre@instr`, `\post@instr` and `\@instr`.

`\inspullback` (⟼) is the amount by which the left margin of the `\instructions` is "pulled back" beyond the left margin of the questions. By default, it will pull back by exactly the correct amount to compensate for the indentation caused by the enclosing enumerate environment. (That is, by default, it will make the instructions go out to the normal left margin.) However, you can reset it in your format file if you want something different.

`\pre@instr` (⟼) is the vertical space inserted before the instructions, similarly

`\post@instr` (⟼) is the vertical space inserted after the instructions

`\@instr` (❖) is a command which does nothing (by default). However, if you know something about LaTeX programming and want to change the appearance of the `instructions`, you can most probably do it merely by defining this command. (Look in `astex.sty` for more information.)

The `qset` environment is useful for keeping instructions and their exercises together. You may consider it as a more cautious method of including instructions than using the `\instructions` command described above.

`qset` (`\begin`-`\end`) is an environment for grouping exercises with their instructions. The syntax is very simple. You merely put the instructions in a TeX file (which I will refer to here as `ins1-2.tex` but you can name it anything) and say

```
\begin{qset}[ins1-2]
\qinput{3p2a9}
\qinput{3p2a10}
.
.
.
\qinput{3p2a25}
\end{qset}
```

As far as they typesetting goes, this would be exactly the same as using the `\instructions` command followed by the `\qinput`s. Furthermore, you can put any command which you would normally put in an exercise file inside the `qset` environment. (For example, you can use `qcolumns` – and most frequently would do so.) The only thing which makes the `qset` environment different is its relationship to the commands `\needsinstr` and `\qsetrange` which are described below.

`\qsetrange` (❖) is a command which prints the range of question numbers associated with the present `qset`. That is, a `qset` is a set of questions, and if anywhere in that environment you say `\qsetrange`, this will print the number of the first problem in the set followed by a long dash followed by the number of the last problem. Most likely, this command will be used in the instructions themselves as in this example:

```
In problems \qsetrange, evaluate the integrals and indicate...
```

Note that outside of a `qset` environment, the command `\qsetrange` will simply print question marks and complain about an undefined reference.

`\needsinstr` (❖) is really the command which makes the `qset` environment useful. Each question in a `qset` environment should contain a `\needsinstr` command to indicate which `qset` it should be a part of. Then, a warning message is printed if either (a) the question appears without its instructions or (b) a question which should not be included in the `qset` is put there by mistake. This is particularly useful in the case that the exercises are frequently being moved during revisions of the book since it prevents these two types of common errors. The command takes a single argument which is the name of the TEX file in which the instructions are saved. That is, if the instructions are saved in the file `ins1-2.tex` (as in the example above) then the command `\needsinstr{ins1-2}` should appear in each question file. For safety, a `%` should be put at the end of the command line to prevent extra space. So, for example, a proper usage of this command might be a file which looks like:

```
\needsinstr{ins1-2}%
\Question{$\int_0^{2\pi}\sin x\,dx$}
\Answer{$0$}
```

`qcolumns` (`\begin`-`\end`) is an environment for inside an exercise list. While in the environment, questions are not printed on separate lines, but in columns. You specify the number of columns you want as an argument.

```
\begin{qcolumns}{3}
\qinput{3p9-3}
\qinput{3p9-4}
.
.
.
\qinput{3p9-30}
\end{qcolumns}
```

The items then run *across* the page in the desired number of columns. TEX can break the page inside a `qcolumns` environment.

There are two OPTIONAL arguments for `qcolumns` which will be the number of columns which appear in some other context. For example, suppose you want the solutions to a particular set to be displayed in 4 columns in the solutions manual. You would then use the first optional argument in square brackets:

```
\begin{qcolumns}{3}[4]
```

Similarly, there is another optional argument in parentheses (which must follow the square brackets) which could be used for another context altogether:

```
\begin{qcolumns}{3}[4](2)
```

To make use of these optional argument, you should understand the following flags.

`\qcolarga` (❖) This flag tells `qcolumns` that it should use the *first* argument to set the number of columns. This is the default, so you will probably never have to use it.

`\qcolargb` (❖) This flag tells `qcolumns` that it should use the *second* argument (the one in square brackets) to set the number of columns. If this flag is called but the optional argument in square brackets is not present, the `qcolumns` command will be ignored and it will be typeset in a single column.

Note: The flag `\qcolargb` is called by the flag`\printanswers`. Consequently, when you are printing the solutions manual, the optional argument with square brackets will be used to determine how many columns a set will be printed in. Usually, there will probably not be an optional argument present and so the solutions will usually be printed without `qcolumns`.

`\qcolargc` (❖) This flag tells `qcolumns` that it should use the *third* argument (the one in parentheses) to set the number of columns. It does not get called by any official ASTEX command. It is merely included in the case that you need to have a third context for printing questions/answers. For example, if you are printing two different solutions manuals and you want a different number of columns in each, then when you define a flag you might want to use `\qcolargc`.

`\noqcolumns` (❖) This flag tells `qcolumns` to ignore all arguments (optional or otherwise). It effectively turns off the `qcolumns` command. It is called by the `\printboth` command that prints answers and questions together, so the `qcolumns`command is ineffectual when `\printboth` is called.

Inside of a question file, you put whatever commands you need to generate the question as you wish it to appear *as well as* the commands necessary to produce the solution for that question. (Both the question and the solution should be in the same file.) The only special commands for inside of a question file are `\Question` and `\Answer`, which take the question and answer as arguments.

`\Question` (❖) takes one argument, which is the question in a question file. It normally merely returns its argument (that is, `\Question{stuff}` is the same as typing `{stuff}`. However, certain flags such as `\printanswers` turn this command "off" so that it does nothing. That is, after this flag has been called, `\Question{stuff}` is the same as typing nothing at all.

`\Answer` (❖) takes one argument, which is the answer in a question file. It normally returns nothing, which means it will look as if the command did not exist in the file at all. However, certain flags such as `\printanswers` turn this command "on" so that it returns its argument. That is, after this flag has been called, `\Answer{stuff}` is the same as typing `{stuff}`.

7.4 Generating A Solutions Manual

In general, it is expected that the solutions will be formatted in a style separate from the rest of the book. Often the page sizes are different, the margins are different, etc. For this reason, it may be convenient to have a separate format for the solutions.

In addition, certain things will *need* to be different. That is, since the same exercise lists and question files will be used to generate the solution manual, the commands must be interpretted differently in order for it to look different than the questions in the book.

For example, use the command `\newexerhead` to make alternate definitions of your exercise headings which are appropriate. If the exercise heading of type `DEF` in the book reads "Exercises for Section 2.3", then you will probably want the heading in the solution manual to say "Solutions to Exercises for Section 2.3".

Note: By using the commands `\Question` and `\Answer`, you could avoid redefining the exercise heading for the solution manual by making the definitions of the headings "smart" enough to know where they are. For example, the definition below would produce the correct results in both the solution manual and the book:

```
\newexerhead{DEF}{\Answer{Solutions to }Exercises for Section \thesection}
```

Most of the other things which you will want to be different in the solution manual are done automatically by the flags described below.

`\printanswers` (❖) is a flag which exchanges the roles of the commands `\Question` and `\Answer`. By default, `\Question{something}` is equivalent to `something` whereas `\Answer{something}` expands to nothing. This way, if you write the question files as described above, the question will appear but the answer will not. Since `\printanswers` exchanges their roles, the question will now be ignored and the question will be printed. In addition, this command turns off the `qcolumns` environment. Furthermore, since `\instructions` is defined using `\Question`, instructions will not be printed.

`\printboth` (❖) is a flag very similar to `\printanswers`, but instead of exchanging the roles of `\Question` and `\Answer`, it lets them both do the same thing, so that both questions and answers are printed. It turns off `qcolumns` but does not affect `\instructions`. This is useful for checking the questions and answers, but probably will not be used in any published material.

`\printeven` (❖) is a flag (which can be used in conjunction with `\printanswers`) that will cause `\qinput` to skip any odd numbered question. When `\qinput` comes upon an odd numbered question after this flag has been called, it will add one to the counter called `enumi`, which is used by LaTeX to count exercises, and it will label by the file name as usual, but it will not load the exercise file.

`\printodd` (❖) does almost the same thing as `\printeven`, with one obvious difference.

7.5 Making an Index

The commands used to generate and print an index have not changed much in ASTEX, so you would probably do well to read a LaTeX manual on the subject. The only differences are that you can use the command `\indexhere` to put the index nicely into any number of columns, and that you will need to "collect" the indices of the different chapters if you are printing a textbook.

`\indexhere` (❖) is the ASTEX command for loading and printing the index. It has two forms, with an without an asterisk ("*"). The form without the asterisk is used to print the index of the file it appears in. So, if you want the index to appear at the end of a file, you need only put the command `\indexhere` in the correct position. In this form, the index is preceded by a heading which is created by `\indexhead`. By default, `\indexhead` creates a section heading entitled "Index", but you can redefine `\indexhead` to make any sort of heading you want.

Alternatively, if you are producing a book, you will want to collect and print an index for the entire book. The command `\collectindex` described below collects the indices from all of the chapters into one index called "book.idx". When `makeindex` is run on this file, it creates the file "book.ind". When `\indexhere` is followed by an asterisk, it will load the file "book.ind" rather than the index for the file itself. In this form, `\indexhere` does *not* put a heading before the index. It is assumed that you will put a heading on the index with a chapter, section or other command.

An optional argument in square brackets (after the "star" if there is one) determines the number of columns which the index will be set in. By default, it will be set in three columns. So, for instance, all of the following examples are reasonable uses of `\indexhere`.

```
\indexhere  % Prints the index of this file in 3 columns
\indexhere* % Prints the index of the book in 3 columns
\indexhere[2]  % Prints the index of this file in 2 columns
\indexhere*[4] % Prints the index of the book in 4 columns
```

`\indexhead` (❖) creates the heading of the index when the command `\indexhere` is used to create it and the "star" option is not requested.

Note: If you merely input the ".ind" file with the command `\input`, rather than `\indexhere`, it will be printed in two columns using the usual LaTeX `theindex` environment.

`\collectindex` (❖) will collect all of the indices from the chapters together into one file. However, it will not print the index, it will merely put it together in a file called `book.idx`. You must use a program such as `makeindex` (which is generally distributed with TeX) to compile it into a TeXable index file called `book.ind`. Then, to print the index, you need only input this file. It is likely that you will want to print the index in two or three columns. You may do that with the ASTeX `\Ncolumns` command or the LaTeX two-column commands. For example, an index such as the one at the end of this document is produced by the statement `\Ncolumn{3}{\input{book.ind}}{\indexhead}`, where `\indexhead` is a command for producing the heading of the index.

Note: Of course, ASTeX must know what chapters make up your book in order to collect the index: see `\nextchap`.

Review of LaTeX Index Features

As stated above, I have not modified the LaTeX commands for creating the `.idx` index file. The following facts are stated as a brief review:

- The command `\index` takes one argument. Its basic syntax is `\index{word}` where `word` is what you wish to appear in the index. (More complicated formats depend on the program used to generate the index from the `.idx` file. The examples below are in the case of the program `makeindex`.) The command `\index` does **nothing** unless the flag `\makeindex` appears at the beginning of the file. If this flag appears, the `.idx` file is generated. If not, the `\index` command is ignored.
- The command `\index{main!sub}` would result in the sub-heading `sub` under the main heading `main`.

- The command \index{alph@print} will alphabetize the item as if it were alph, but will actually print print.
- The command \index{word|cmd} will index word and apply the command \cmd to the page number. For example, \cmd could be defined to make the number bold or italic, as in the TEX manual.

8 General ASTEX Commands

8.1 Making a Table of Contents

\tableofcontents (❖) will load and print the table of contents for the file which it is contained in. This is what you would probably want to use in a journal or book of conference proceedings. If "auto-page by chapter" has not been selected, the page number will automatically be reset after the table of contents. It will be set to one unless "auto-page by issue" has been selected, in which case it will be set to the next appropriate page number. In this form, the command automatically calls the command \maketoc as well, so you do not need to call that separately. However, since a textbook is going to be made up of several different chapter files, the "star" option will most often be used with textbooks.

When the \tableofcontents command is followed by and asterisk ("*") it will collect and print the table of contents for each file in the chapter list. You can put this command in a file which is part of the "front matter" of the book, including the preface and other stuff, and it will then print a table of contents for the entire book. (It does not call \maketoc in the "star" form, so you will have to call this flag separately if you want the file containing the table of contents to save its own table of contents file.

In both versions of the table of contents, the folio is determined by the command \thetocpage. By default, it is defined to be \roman{page}, so that the page numbers in the table of contents will be lower case Roman numerals.

\tochead (❖) is the command which generates the heading for the table of contents. You may define it to be anything you wish, but it merely does \title{Table of Contents} by default.

\maketoc (❖) is a flag which tells ASTEX to generate a table of contents file for the present TEX file. This command does not cause the table of contents to be printed, but merely stored in the file JOBNAME.toc. It is automatically called by ASTeX in any file which is in the chapter list or any file with a \tableofcontents command (without an asterisk).

Note: The new table of contents is not created unless the \end{document} is reached. If a file is only "partially TEXed", then the table of contents will remain unchanged.

\extratoc (❖) is a command which allows you to add extra lines to the table of contents. For example, suppose the chapters in your book or articles in your journal are broken down into **Part I** and **Part II**. To include these lines in the table of contents just use the command \extratoc{PART I}, etc.

To determine how your table of contents looks, you will need to define a "\tcl@" type for each kind of entry in the table of contents, and a font ("\tcf@") for each type of entry. Default values for \tcl@'s are provided. For example, you must have a value set for \tcl@section, \tcl@subsection, \tcl@article, \tcl@chapter, \tcl@extra etc.

(To see what I mean, look in the `.toc` file generated by your ASTEX file. It contains commands of the form

```
\contentsline{section}{\numberline{2} Bleh blah bleh}{33}
```

This command will, in turn, call the command `\tcl@section`, because the first argument is "section".) The sample `\tcl@`'s that are provided for your use are `\tcl@a` – `\tcl@h` as well as `\tcl@ignore` and `\tcl@center`. To use them, you may include lines such as:

```
\let\tcl@section\tcl@ignore
\let\tcl@article\tcl@b
\let\tcl@extra\tcl@center
```

in your format file. This would mean that section headings should not be included in your table of contents, that article headings should be printed in type "b", and that comments included with the `\extratoc` command will be centered.

In addition, you must specify a font for each type, where the font name is something like `\tcf@section`, `\tcf@extra`, etc. Also, the commands beginning `\tcs@` will be executed immediately after the `\tcl@` command. (In other words, `\tcs@section` will be executed after the printing of each section in the table of contents. The intended purpose of this command is merely to create the vertical space. This is why the default, if no `\tcs@` command is defined, is merely to execute the LaTeX command `\smallskip`. However, you could define it to do more complicated things, such as loading a picture or something.

Finally, it is a little more complicated in the case of articles, because you also have to say what to do with the author and title. This introduces two new fonts and one new command: `\authtitl`. There are 4 predefined possibilities for `\authtitl`, any one of `\authtitla` – `\authtitld`. The associated fonts are `\tcf@auth` and `\tcf@titl`.

8.2 Auto-Page: Getting the Right Page Numbers

Both in producing a textbook and in producing a journal with ASTEX, it will be useful to have the page number of the first page of a file set automatically. In a textbook, for instance, each chapter should automatically start after the previous chapter. In a journal, on the other hand, each issue in a volume starts after the previous issue. This is handled automatically by the "autopage" routine of ASTEX.

`\noautopage` (❖) is a flag that can be used in your format file or in an individual file to turn off both of the auto-page routines.

`\startoddpage` (❖) is a flag that sets both auto-page by chapter and auto-page by issue to begin each chapter/issue on an odd page number. Thus, if the next page after the previous issue/chapter is even, it will be skipped. Books and journals generally start on an odd page number.

`\startanypage` (❖) turns off `\startoddpage`, so that the book or journal can start on an odd page *or* an even page.

Auto-Page by Chapter

`\autopagebychapter` (❖) is the flag which indicates that each chapter should automatically begin after the previous one. This command is called automatically by `\textbookdefaults`, which should probably be in your format file anyway.

It is assumed that you have a chapter list created by the command `\nextchap` in your format file. This list informs ASTEX of the files which make up the book and the order in which they appear. Consequently, it is able to determine if the file you are TEXing is one of the chapters, and what page it should start on. When you first TEX a file, the name of the file will be compared to the names of all of the files in the chapter list. (Note: the directory is not checked – only the name of the file.) If it finds the name in the list, it calls `\autopage` with the previous file in the list. In addition, if it finds the name of the current file in the list, it automatically saves the last page number of the file so that other files can `\autopage` it.

Note: To use `\autopagebychapter`, you need to have a chapter list and a home directory (saved in the macro `\homedir`). If you have another file which has the same name as one of the files in the chapter list, ASTEX will assume that you want it to start on the same page. (If you don't, you can turn off autopage in that file with the flag `\noautopage`.)

Auto-Page by Issue

`\autopagebyissue` (❖) is a flag which indicates that the first page number in each file should follow the last page number in the previous issue. This flag is called automatically by `\articledefaults`, which you should have in your format file anyway. In addition, if you use this feature, you must define a variable called `\issuenumdir`, which is the directory in which this information will be stored. For example, you might define it with the line

```
\def\issuenumdir{/usr2/pub/ISSUES}
```

The directory can be one in which you keep nothing else, or a directory that you use for other things.

Furthermore, you must use the command `\issuenum` to inform ASTEX of the issue number of the present issue. After you do that, the command `\tableofcontents` or `\setpagenumber` will set the page appropriately.

By default, the information is saved in files called `issue-1.lstpg`, `issue-2.lstpg`, etc. You may change the prefix of those files (from `issue-`) to anything you might want. In particular, if you produce more than one journal, you might want the name of the journal to be included to differentiate the journals. You can set this in the macro `\ispref`.

`\startartodd` (❖) is even stronger than `\startoddpage`. This command insists that each individual article begin on an odd page number. Although less common than the practice of starting a book or journal on an odd page, this is certainly done.

`\startartany` (❖) turns off `\startartodd` so that an article can start on an odd page *or* on an even page.

Your Own Auto-Page

The basic commands which ASTEX uses for auto-page by chapter or by issue are available for your use. You can use this to have some "auto-page" features in files which are not part of the textbook or in a series of journals.

`\autopage` (❖) takes one command, that is the file (including directory) which the current one should follow. It then looks at that file's ".lstpg" file and starts on the following page. (If `\startoddpage` is set, it starts on the next odd page.) This only works if the file in question has had `\savepagenum` called.

`\savepagenum` (❖) is a flag which causes the current file to save its last page number in a file so that other files can refer to it with `\autopage`. This command is automatically called in files which are part of auto-page by chapter or issue.

8.3 Labels and References

LaTeX provides the commands `\ref` and `\label` for labelling and referencing counters. However, these commands only work within a single TeX file. Since each chapter in the book is a separate TeX file, this means that you will not be able to use these commands to reference between chapters. Most references will be contained within a chapter and can be done with the usual `\ref` command. The "global reference" commands `\gref` and `\glabel` will be used for those labels which must be referred to by other files.

Regular References

When a counter such as the section number or figure number is stepped (i.e., increased by one) the current value of the counter is placed in a special location.[1] When you say `\label{SOMETHING}`, LaTeX sends the value from this special location along with the word `SOMETHING` to the aux file. Then, when you TeX the file again, the aux file gets read in at the beginning of the job and `\ref{SOMETHING}` is defined to print the value of that counter. There are two important points here:

- The `\label` command must immediately follow the stepping of the counter which you wish to label.
- The `\ref` command will not work until the *second* time TeXing the file since it must first be written to the aux file.

In most cases, the thing which you will want to refer to is *automatically* labeled by ASTEX. For example, any question which is read in by `\qinput` is automatically labelled by the name of the file. Thus, to refer to it by number, you need only say

```
Question~\ref{FILENAME}
```

Similarly, figures are automatically labelled by the `\finput` command.

However, other counters which you might want to refer to (such as the section number) are *not* automatically labelled. If you want to refer to the number of a section, merely put a `\label` command immediately after the `\section` command and then refer to it with a `\ref` command.

Note: Although the tilde (`~`) is frequently used in conjunction with the `\ref` command, it is not actually related to it. (I have encountered TeX files in which it is clear that the author believes that the `\ref` command will not work without a tilde in front of it.) In fact, tilde is just a space at which TeX will not try to put a line break.

[1] This is not true for *every* counter; it is true if the counter is stepped with the command `\refstepcounter` rather than just `\stepcounter`.

Note: These commands automatically record page numbers as well. If you wish to refer to the *page* on which a particular label occurs, rather than the last counter which was stepped, use the `\pageref` command.

References in Articles

Since it is likely that the articles which you are putting together into a journal or book were not created by the same authors, there is no reason to expect that the references will be distinct. Consequently, ASTEX will *localize* labels and references to an article. That is, if you `\label` something "`bleh`" in one article, you will not be able to refer to it in another article with `\ref`. (The advantage, on the other hand, is that the authors of both articles could have used the label "`bleh`" without causing any problems. A label which is made in something other than an article (i.e., something not loaded with the article command, as you might do for a preface or something) can be referred to anywhere. In addition, it is certainly possible that you will *want* to refer to one article from another. This can be done simply by using the command `\Label` rather than `\label`. When the capital "L" label is used, it makes the label as if it were not just defined in that article. Less common is the possiblility that you will want to refer to something which was TEXed in another file all together. Although this is not very common in cases other than textbook production with ASTEX, the commands `\glabel` and `\gref` can be used, as described in the next section.

Global References

Since the chapters of a textbook are going to be separate TEX files when produced with ASTEX, I need to provide "global" equivalents of the label and reference commands that work throughout the book. The commands `\gref` and `\glabel` are provided by ASTEX to play a role similar to `\ref` and `\label`. The command `\glabel{SOMETHING}` will save the name and value of the last counter to a file called `JOBNAME.grf` (rather than `JOBNAME.aux`). So, for instance, when you LATEX `chap1.tex`, it produces a file called `chap1.grf` in addition to `chap1.aux`. The difference is that although `chap1.aux` is read only by the file `chap1.tex`, the global reference file can be read by any other file.

This is another routine which makes use of the chapter list, which is a list of the files which make up your book and their directories. Assuming you have set up a chapter list using the `\nextchap` command in your format file, you can use `\glabel` and `\gref` exactly as you would `\ref` and `\label` for any files in the list. For example, you could save the number of a figure in `chap3.tex` with the command `\glabel{foofig}` and refer to it in `chap2.tex` or `chap9.tex` with the command `\gref{foofig}`. This is because the grf files of each file in the chapter list are read into every file in the chapter list, just as they read their own aux file.

In addition, this command works for other files as well. For any file which is *not* in the chapter list, you can use the `\glabel` command in exactly the same manner. However, for another file to refer to it, the command `\greferstofile` must be placed at the beginning of the file (along with the flags) to indicate the name of the file to which it is referring. For example, suppose the file `FRED/foofile.tex` is *not* in the chapter list, but some other file has a `\gref` to a `\glabel` in it. Then the other file should have contain the line

```
\greferstofile{FRED/foofile}
```

(Here, the directory `FRED` is given with respect to the home directory: `\homedir`.)

Similarly, a file which is not in the chapter list can refer to a file which is in the chapter list by using the commands `\greferstofile` and `\gref`.

Note: A page number reference version of the global reference command is provided as well: `\gpageref`.

Note: The global references are stored separately from the usual LaTeX references. Thus, you can use the label `foo` as both a global and local reference for different things without any conflict. However, just as you cannot have two labels with the same name in the same file (since LaTeX will not know which to refer to), you cannot have two global references with the same name in the same book. (A warning will appear if you do.)

`\glabel` (❖) is similar to `\label`, however the information is stored in a ".grf" file rather than a ".aux" file. Other files may read in the .grf file in order that they may be able to refer to a label made with this command. Global labels within the same book should be kept distinct in order to avoid confusion.

`\gref` (❖) is similar to `\ref` however it refers to "global" labels made with the `\glabel` command.

`\greferstofile` (❖) is used as a flag to indicate that the present file contains global references to another file. This is not necessary when both files in question are contained in the chapter list, but is necessary in all other cases. The command takes one argument, which is the name of the file including the path or directory relative to the home directory given by `\homedir`.

8.4 Paginating Commands

One common complaint of authors is that TeX is very *stubborn* about pagination. It can be quite difficult to get it to break the page in a good place, and even harder to get it to break the page exactly where you want! In fact, it is possible to convince TeX to automatically break the page in reasonable places. These features were built in by the author of TeX, and I merely utilize them in ASTeX to try to get a pretty good pagination automatically. Although I hope you will generally be happy with the pagination which ASTeX selects, there will certainly be times that the pagination is *not* what you really want. The commands in this section are "hacks" which help you solve this problem.

`\runover` (❖) is the biggest hack of all, but it is very useful. This command *prevents* TeX from breaking the page. If you use this command without `\breakhere`, it will produce a very overfull page. (Text will run off the bottom of the page.) However, it is intended that you will also use the command `\breakhere` to break the page at a reasonable location. In this way the command merely indicates that the present page should be broken by manual pagination, rather than automatic pagination. (I like to think of it as being like the "margin release" button on an old typewriter.)

Note: The command has no effect if it is the first thing on the page. It must be atleast on the second line for it too have any effect. The way it actually works is that it tricks TeX into thinking that it still has twice `\textheight` of empty space left on the page.

`\breakhere` (❖) is just another page breaking command. (There are many different ones.) This command will cause the page to break at the position in which it is placed.

`\ignore` (❖) takes one argument, but it might be a LONG argument, and ignores it. For example:

```
This is a test \ignore{and this will be ignored} of the ignore command.
```

will simply print "This is a test of the ignore command." Anything can be in the argument, including input commands and environments, but they will all be ignored.

`\pareject` (❖) is another page breaking command, but it is used to break the page in the middle of a paragraph. For example, if you want a page break to occur *exactly* between the words "I" and "am" in a particular sentence, then you put `\pareject` between them and "I" will be the last word on the page. (Note: It is a hack. What it does is tell TEX not to add space and the end of the paragraph, end the paragraph, break the page and tell TEX not to indent the next paragraph!)

The commands in the form above are always "on". You can also make many different sets of those commands that are only turned on by a flag. To generate these sets of commands, use the `\newpagination` command.

`\newpagination` (❖) takes one argument and then defines a set of paginating commands with that argument as prefix and a flag to turn them on. For example,

```
\newpagination{FINAL}
\newpagination{B}
```

will create the commands: `\FINALrunover`, `\FINALvspace`, `\FINALbreakhere`,...`\Bpareject`. These commands can be placed in your document and will have no effect. The flag `\FINALpagination` will turn on all of the paginating commands with the prefix `FINAL`, but leave the others alone. Similarly, the flag `\Bpagination` will turn on all of the `B` commands. This is particularly useful when you are maintaining several versions of the document. (See `\suffix` below.)

Finally, the command `\nostretch` can be considered a paginating command, but it is not included in the commands generated by `\newpagination`. It is part of many other ASTEX macros, such as `\article` (to leave the last page of each article short) and `\enddocument`.

`\nostretch` (❖) prevents the current page from being "stretched" to `\textheight`. In general, ASTEX will stretch every page (if possible) to be the same height. This command will turn off that stretching for a single page. This is generally only done at the end of something (like the end of a list, section or table of contents, etc.).

8.5 Maintaining Different Versions of a Document: `\suffix`

There are many occasions on which you might need to make more than one version of a document. For a book or journal, there might be a long and a short version, or an American version and a European version. For an individual article, you might have a version prepared for submission to a journal and another one which you will distribute as a preprint. In each of these cases, the differences between the versions will be relatively small. It doesn't make sense to have a completely separate copy of the file for this other version, because you would then have to remember to modify them both if the need arose for changes. (Not to mention the fact that it would take up more space.) ASTEX provides a feature which allows you to maintain separate versions of a document (as many as you want) without having separate copies of the file. The only restriction is that all of the differences must be made by a flag. (Remember, a flag is a single command which goes at the beginning of the file.) I will use the case of an American and British version of a textbook to illustrate this point.

Suppose you had a command called `\AmBr` which takes two arguments: an American spelling and a British spelling. (For example, `\AmBr{color}{colour}`.) This command, by default, ignores the second argument and merely expands to the first argument. In this way, the book will print out without the British spellings at all. However, the flag `\UseBrit` could *switch* the command around, so that now the British spellings are used and the American spellings ignored. Below, I will explain how you can maintain both the British and American versions of the book at the same time.

Suppose the TEX file in question is called `chap3.tex`. When you TEX this file, the American version is printed. If, on the other hand, you include the flag `\UseBrit`, it prints with British spelling (however, this flag is *not* in the file...I was just speaking hypothetically). Now, if you create a file called `chap3BRIT.tex` which says the following:

```
\def\suffix{BRIT}
\def\suffixflag{UseBrit}
\input{chap3}
```

it will look like `chap3` with the flag `\UseBrit` on when it is LATEXed. The difference is, since it is a different file, the table of contents, index and stuff are saved separately! So, you don't erase the information from the American version. The commands `\gref`, `\tableofcontents*` and `\collectindex` all are "aware" of the current suffix and will load their information from the other files with the same suffix.

`\suffix` (❖) is a variable which holds the name of the "suffix" of the current file. Usually it is defined to be nothing. This suffix is appended automatically to filenames of "information" files which are included when appropriate so that an entirely separate book can be maintained. The suffix (and suffix flag) should be set in a file whose name is the same as another file but with the suffix attached, and then the other file should be input. That is, if you want a different version of the file `file` with the flag `dobleh` called, make a file called `fileBLEH` which says:

```
\def\suffix{BLEH}
\def\suffixflag{dobleh}
\input{file}
```

`\suffixflag` (❖) is automatically called at the end of every `\asformat` command. By default, it does nothing. However, if it is defined before the `\documentstyle` command (in the manner described above) it can be used to maintain separate versions of a document.

8.6 Flags

`\verbose` (❖) will cause ASTEX to be very "talkative" during LATEXing. In particular, it will print out information about what command it is executing and what it is "thinking". This can be very useful when you are trying to debug a document which is doing something wrong, or when you simply want to have a better understanding of how ASTEX works.

`\showfilename` (❖) is a flag which you might want to have on during processing of your document in all of its "draft" stages. This command causes the names of files which are input with `\exercises`, `\finput`, `\tinput`, `\qinput` and `\article` to be printed on the page.

8.7 AMS Compatability

ASTEX can be used as an AMS-LATEX style file. If you are doing this, include the flag `\amscompatability` in your format file and use `amslatex` as the command to process your document. This will give you all of the AMS-LATEX commands and fonts. However, you should note that there are differences between the font specifications in LATEX and AMS-LATEX, so you may have to alter your as-fonts file slightly to use `amslatex`. Furthermore, you should be warned that AMS-LATEX uses much more memory than just ASTEX alone. Try not to load too many font families other than those loaded by AMS-LATEX so as not to use up more memory than you have.

8.8 Miscellaneous Commands and Hacks

`\note` (❖) is used for a comment such as

Note: This is a comment. It takes one optional argument, which is the word printed in bold. By default, it prints the word "note", but `\note[Case 1]` will print "Case 1".

`\Ncolumns` (❖) is a command for putting text into columns. It takes three arguments: the number of columns, the stuff to put into columns and a heading for it.

```
\Ncolumns{N}{STUFF}{HEAD}
```

The number `N` should be an integer greater than 1. The only real limitation on `STUFF` is that it can't be *too* long. More than a few pages worth may be too much! Finally, `HEAD` should be commands that generate the heading, and not just some words. For example, it can be a `\section` command or a `title` command.

`boxeqnarray` (`\begin`-`\end`) is a slight modification of the usual `eqnarray` environment of LATEX. Recall that the `eqnarray` environment is the standard method of typesetting a set of equations in LATEX. (It is similar to the `\align` command in AMSTEX.) However, one problem which my "clients" often ran into with this environment was that it automatically starts a new line. There are certain situations, such as when the environment is the first thing after an `\item` in an `enumerate` environment or the first thing in a "solution", in which you want the `eqnarray` to line up with something else.

Hence, `boxeqnarray`, which can be thought of as an `eqnarray` in a box. It lines up appropriately next to anything like an "item label" which hangs in the margin.

Just like the `eqnarray` environment, a "star option" given by an asterisk ("`*`") turns off the numbering of equations in the array. See a LATEX manual for more information.

`\defbox` (❖) takes one long argument (a paragraph or two) which it sets in a large, centered box with a shaded background. An optional argument in square brackets determines the width of the box. For example,

```
\defbox[2in]{This is an example of a \defbox of width 2 inches.
By default, the box will be the full width of the page.  Displayed
math, like
$$
```

```
\sin^2\theta+\cos^2\theta=1
$$
looks fine in a \defbox.

So does a new paragraph.  You do have control over the graylevel of
the background.  It is determined by the variable \defboxshade.}
```

This is an example of a `\defbox` of width 2 inches. By default, the box will be the full width of the page. Displayed math, like

$$\sin^2\theta + \cos^2\theta = 1$$

looks fine in a `\defbox`.
So does a new paragraph. You do have control over the graylevel of the background. It is determined by the variable `\defboxshade`.

`\rulebox` (❖) is basically the same as `\defbox`, but does not have the shading behind it. Both `\rulebox` and `\defbox` color separate as "color" not "black".

`fullpagesize` (`\begin`-`\end`) is an environment which temporarily "stretches" the page size out to the left (by exactly the amount that the section headings hang out). If `\cutpgwid` and `\fullpgwid` are set to the same value, this environment will have no effect. Since the results are put in a `\vbox`, there cannot be a page break within the environment.

`\gray` (❖) inserts the postscript commands which indicate to print in some gray scale rather than in black. It takes one optional argument which is a number between 0 and 1. For instance,

```
{\gray This is gray} This is not.
```

Looks like: This is gray This is not. Printing will be in white if the optional argument is 1, and black if it is zero (and linear in between):

```
{\gray[.5] This is gray \gray[.25] this is gray \gray[.75] this is gray.}
```

produces This is gray this is gray this is gray.

Note: Although the command is "contained" by TEX braces, it is not really a TEX command. It is a postscript command and makes use of the color stack. It is essentially the same thing as the command `\pushblack` except that its argument is "opposite" (e.g. 1 is white, 0 is black, not the other way around) and that it automatically calls `\restorecolor` at the end of the present group.

`across` (`\begin-\end`) is an environment like the enumerate environment, but it runs *across* the page rather than vertically. It takes two optional arguments: the type of labels and the number of columns. The syntax is:

```
\begin{across}<N>[A]
\item OBJ1
\item OBJ2
.
.
.
\item OBJN
\end{across}
```

where `N` is the number of columns and `A` is an indicator of the label type. If the number of columns is not specified, `across` will just put as many objects on a line as it can fit. If the label indicator is not present, it will label in lower-case letters (a, b, c,...). Possible indicators are: `a` for lower-case letters, `i` for lower-case Roman numerals and `I` for upper-case Roman numerals. (It is easy to add new types. Look at `\ac@a` in the style file if you want to add more.)

For example, the result of the following code:

```
Which of the following is not a British television show?
\begin{across}<3>
\item Red Dwarf
\item Fawlty Towers
\item Connections${}^2$
\item Hee Haw
\item Black Adder
\item Doctor Who
\end{across}
```

is:

Which of the following is not a British television show?

(a) Red Dwarf	(b) Fawlty Towers	(c) Connections^2
(d) Hee Haw	(e) Black Adder	(f) Doctor Who

Arrays

One programming tool which I always found very useful (and I was disappointed not to find in TEX, was an array. An array of variables is a set of variables indexed by positive integers. So, for example, I would not only be able to have a single variable called `\fudge`, but a whole series of them: `\fudge{1}`, `\fudge{2}`, etc. I have introduced some of the power of arrays with the following commands.

`\newarray` (❖) takes one argument and sets up a new array with that name. (E.g. `\newarray{fudge}`.)

`\nextelt` (❖) takes two arguments: the name of an array and the value to set the next element to. It then adds a new element to the array which has the value deteremined by the second argument. So, for example

```
\nextelt{fudge}{3}
\nextelt{fudge}{extra large}
\nextelt{fudge}{\thesection}
```

adds three elements to the array `fudge`. You can then call an element by merely saying `\fudge{1}` (which is the number 3 in our example) or `\fudge{3}` which in our example is the section number in which the element was defined (and not necessarily the current section number).

`\forarray` (❖) is a loop command which runs through an array. It takes two arguments, the name of the array and what to do. In the second argument, the array can be referred to without an element number to mean the current element in the loop. For example

```
\forarray{fudge}{I really like \fudge.\par}
```

will produce:
I really like 3.
I really like extra large.
I really like 8.

`\breakarrayloop` (❖) is a useful command to use in an array loop. If it is encountered in the second argument of `\forarray`, the loop will halt after completing that round. So, if you include this in an `\if` statement of some sort, you can cause the loop to stop when a certain condition is met.

`\fornum` (❖) is another loop, although not associated with arrays. This loop merely performs some task a given number of times. This is useful when you want to print the letter "a" exactly 345 times. It takes two arguments, the number and the thing to do. (Note: the counter involved is called "array", so you can refer to the number explicitly.)

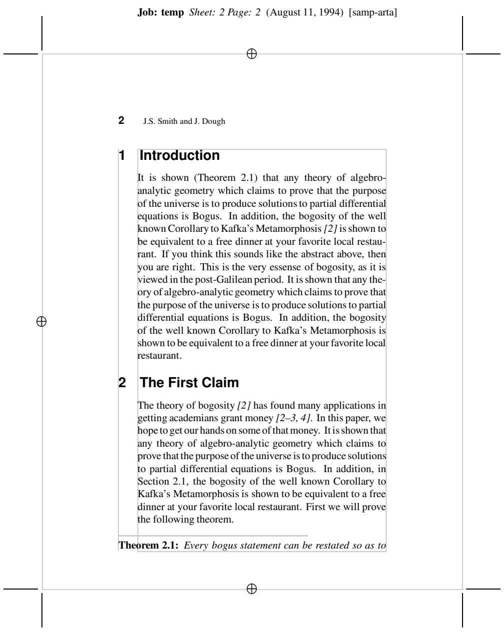
```
\fornum{4}{This is a test.  I'm up to step number \thearray.\par}
```

This is a test. I'm up to step number 4.
This is a test. I'm up to step number 4.
This is a test. I'm up to step number 4.
This is a test. I'm up to step number 4.

9 Paper Marks

The paper marks are symbols which (hopefully) will not appear in the published version of your document, but will be helpful in producing it. The paper marks are illustrated in Figure 9.7 and include:

- Crop marks, that indicate the size of the paper as determined by `\paperht` and `\paperwd`. These marks are used to trim the final printing of the document to the correct size. These marks are the pairs of lines which appear at the corners.
- Registration marks, which look like little "crosshairs" (or direct sums, for those mathematicians out there). These are used to align separations when printing color separations.



Job: temp *Sheet: 2 Page: 2* (August 11, 1994) [samp-arta]

2 J.S. Smith and J. Dough

1 Introduction

It is shown (Theorem 2.1) that any theory of algebro-analytic geometry which claims to prove that the purpose of the universe is to produce solutions to partial differential equations is Bogus. In addition, the bogosity of the well known Corollary to Kafka's Metamorphosis *[2]* is shown to be equivalent to a free dinner at your favorite local restaurant. If you think this sounds like the abstract above, then you are right. This is the very essense of bogosity, as it is viewed in the post-Galilean period. It is shown that any theory of algebro-analytic geometry which claims to prove that the purpose of the universe is to produce solutions to partial differential equations is Bogus. In addition, the bogosity of the well known Corollary to Kafka's Metamorphosis is shown to be equivalent to a free dinner at your favorite local restaurant.

2 The First Claim

The theory of bogosity *[2]* has found many applications in getting academians grant money *[2–3, 4]*. In this paper, we hope to get our hands on some of that money. It is shown that any theory of algebro-analytic geometry which claims to prove that the purpose of the universe is to produce solutions to partial differential equations is Bogus. In addition, in Section 2.1, the bogosity of the well known Corollary to Kafka's Metamorphosis is shown to be equivalent to a free dinner at your favorite local restaurant. First we will prove the following theorem.

Theorem 2.1: *Every bogus statement can be restated so as to*

Figure 9.7: The extra lines on this page (and the text at the top) are paper marks. You can turn them on or off individually with flags.

- The textbox, which is a box surrounding the region in which text is supposed to appear. It can be used to help check that the text does not "stick out" or "stick down" too far.
- The paper box, which is a box surrounding the region which will comprise the actual page in the published verion of your document. It is useful to have this box printed so that you can get a good idea of what the layout will actually look like on the correct page size.
- The top stamp, which is text printed just above the crop marks with information about the printing.

The paper marks can be controlled with the following commands and parameters:

`\nocropmarks` (❖) is a flag which turns off the printing of crop marks. The flag must be called before the `\begin{document}`.

`\showcropmarks` (❖) is a flag which turns *on* the printing of crop marks. This is the default.

Similarly, there are commands `\noregmarks` and `\showregmarks` for registration marks (`\noregmarks` is the default), `\showtextbox` and `\notextbox`, and finally `\showpaperbox` and `\nopaperbox`.

There is also a command called `\notopstamp` for turning off the printing of the text above the crop marks, but there is no command for turning it back on. To turn it back on, you need to redefine `\topstamp`.

`\topstamp` (❖) determines what gets printed in the text above the crop marks. By default, it prints the jobname, sheet number, page number, date and article name. The command

`\notopstamp` defines it to be nothing. You can get it to do anything else by simply redefining it.

`\regmarksize` (❖) is a number which determines the "diameter"[2] of the registration marks in inches. By default, it is set to .5, making the registration marks one inch across. You can define this to be another value in your format file with a line such as

```
\def\regmarksize{.2}
```

`\dim@regmark` (↔) is the distance from the registration mark to the edge of the page. By default, it is set to zero (zero inches or points or whatever...its all the same) which means that it is centered on the edge of the page. If you set it to some positive distance, then it will be shifted off the page slightly. This is desirable if you are doing color separation and are including things printed arbitrarily close to the edge of the page.

10 Displaying Figures, Tables and Things

10.1 ASTEX Float Environment: `floatfig`

The `floatfig` environment was designed to replace the `figure` and `table` environments of LaTeX because we found that we were rarely happy with the locations that those environments chose to put things. Although you could put anything inside of it, it is expected that you will either put commands for loading figures (i.e., `\finput`) or creating tables in it. Below, I will refer to these commands simply as `OBJECT`.

`floatfig` (`\begin`-`\end`) will take any object put inside the environment and: center it, put space around it and float it to a good location. The syntax is simply

```
\begin{floatfig}
OBJECT
\end{floatfig}%
```

without any arguments or options. The percent sign (`%`) after the `\end` is not really *necessary*, but it is highly desirable. If this is not there, extra space might be left by `floatfig` in the position which it has "floated away from". It will attempt to float the object to the top or bottom of the page. If that location is either already taken up by another `floatfig` or is blocked by a `\dropem` command, it will float the object to the top of the next page. `floatfig` will never float an object farther than the top of the next page. If necessary, it will put several figures at the top of the next page.

There are two commands which you can use in your format file to control the behavior of `floatfig`. One, `\floatscanstay` is a flag; the other, `\bonly`, is a variable.

`\bonly` (❖) (stands for "bottom only") is a variable whose value lies between 0 and 1 that determines at what point a float is going to go only to the bottom without trying to go to the top. Notice that the float routine *generally* always tries to go to the top of the page first, unless it is occupied by another picture. However, using this variable you can have it try the bottom first if the float is called more than a certain percentage of the way down the page. If you define `\bonly` to be .5, then any picture called more than halfway down the page will go to the bottom first. Setting it to .25 will cause any picture

[2] The distance from the center of the cross to the tip of the lines, not to the arc of the circle.

more than one quarter of the way down the page to go to the bottom. Alternatively, if you set it to be 1 or greater, it will always try the top first.

If another float tries to move to the bottom after a float has been put there because of `\bonly`, the first float will be moved to the top of the page. So, in these rare instances, a float which is called more than `\bonly` of the way down the page can still end up at the top.

Note: One interesting way to use this variable is to set it to be negative. In this situation, the picture will always try to float down first, even if it is near the top. This is good for those authors who hate for figures to float to a point earlier than where they were called.

`\floatscanstay` (❖) is a flag which you can include in your format file. This command will cause `floatfig` environments to *not* float whenever they could possibly fit right where they are. When this flag is on, `floatfigs` will hardly ever float, but will still not mess up pagination as much as if you always use non-floating environments to display figures and tables.

As stated above, the `floatfig` environment "watches" for the occurrence of a command called `\dropem`. The idea is that this command is a barrier past which `floatfig` cannot carry a floating object.

`\dropem` (❖) is a command with no arguments or options. If a `\dropem` occurs on a page, then no `floatfig` which follows will be able to float to the top of the page. Similarly, if a `floatfig` preceeds a `\dropem` on a page and it intends to carry the object to the bottom of the page or the next page, this action will be "blocked" and it will "drop" the object at the location of the `\dropem` instead. It is not actually intended that you will use this command frequenlty. Insteady, it is expected that this command will be used *within* other commands to build natural boundaries for `floatfig`. For example, it is included in the definition of `\section` so that figures will not float out of one section and into another.

Sometimes you do not want an object to be able to float; sometimes you know exactly where you want it to appear. However, in order to make it look the same as if it had floated to that location in a `floatfig`, the environment `stayfig` is provided.

`stayfig` (`\begin`-`\end`) looks very much like `floatfig` in that it centers an object with some space around it. However, it does not float and will appear exactly where you put it. The syntax is simply

```
\begin{stayfig}
OBJECT
\end{stayfig}
```

There are some situations (for example, within some environments of your own creation) that you do not want to allow any floating. The flag `\stayfigonly` is provided for this purpose.

`\stayfigonly` (❖) is a command, which can be used as a flag or locally to turn off the floating ability of the `floatfig` environment. It is included in the definition of many other environments in which floating would not be possible or would not make sense.

Note: Since the "decision" processes of `floatfig` can be rather complicated, it may be useful to use the flag `\verbose` to see a description of these decisions in the log file.

BIG Floatfigs or Stayfigs

ASTeX is designed to allow for an extra wide margin on the left side of the page. Often, if a figure or table is to wide to fit within the textbox of width `cutpgwid`, it is desirable to have it stick out the extra distance to the `fullpgwid`. The environments `fullfloatfig` and `fullstayfig` are created for this purpose. These environments behave *exactly* like `floatfig` and `stayfig`, but they work with the entire `fullpgwid`.

10.2 Putting things in the margin

In many books, the difference between `\fullpgwid` and `\cutpgwid` is large enough that figures or supporting text are placed in that margin. The command `\putinmargin` can be used for placing objects in the left margin. It should be noted, however, that this command is not at all "smart". It will not make sure that there is room for the object (i.e., that it will not run off the page or crash into other elements). The output of the command is illustrated in Figure 10.8.

`\putinmargin` (❖) takes one argument and one optional argument. The argument is an object which will be placed into the left margin of the page. The optional argument, in square brackets, is either the letter `t` or the letter `b` to determine the position of the object relative to the line from which it was called.

```
\putinmargin[t OR b]{OBJECT}
```

The command may occur anywhere in a line of text and the object will appear to the left of that line. The amount of space left for an object placed by this command is the space in the margin minus the dimension `\pimgap`.

`\pimgap` (⟷) is the amount of space left between the main text and an object which is

1 Outline

A brief review of previous results which will be necessary to understand the results of this paper follows in Section 6. For details, please refer to *[KrN,Pr]* for a review of the theory of spectral curves, *[Mu]* for the algebraic theory of the KP hierarchy and to *[SW]* for the analytic theory of the grassmannian which will be used in this paper.

This is in the margin

In Section 7, the technique of the dual grassmannian is utilized to study bispectrality of the wave function. Using results of Wilson *[W]* which completely identified the bispectral wave functions in Gr_1, a set of conditions on W or τ_W which are equivalent to bispectrality of ψ_W are listed (Theorem 8). In particular, it is shown that for $W \in Gr_1$, the wave function ψ_W is bispectral if and only if τ_W is of *bounded degree*. As a consequence, it can be shown that any τ which is polynomial in x with constant leading coefficient is necessarily a polynomial in each time variable t_i. A curious corollary of this fact is that one may use the KP hierarchy to generate systems of non-linear PDE's with *only* polynomial solutions.

The bispectral involution introduced by Wilson *[W]* is studied in Section 9. It is noted that the composition of the bispectral involution with the KP flows results in a non-isospectral flow under which the singular points of the spectral curve move as a cmps. The action of the involution is determined explicitly (Theorem 10) in terms of dual grassmannian parameters in the generic case that the image is one of Krichever's rational solutions *[Kr]*. Using the correspondence between rational solutions and particle systems, the bispectral involution is determined as an involution on cmpss (Theorem 11), and it is shown (Theorem 12) that it acts as a linearizing map. The bispectral involution acting on particle systems, after simple rescaling, is seen (Theorem 13) to restrict to the linearizing map σ introduced in *[AMcM]* and thus explains the surprising involutionary nature of σ in terms of bispectrality. The paper concludes with two examples. The first example demonstrates how the bispectral involution can be used to determine the τ-function of a rational KP solution. Finally, the second example demonstrates Theorem 14 by determining the condition space of the image under the bispectral involution of a particular rational solution of the form given by Wilson *[W]* in Example 10.3.

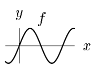


Figure 1: This is the caption

Figure 10.8: The command `\putinmargin` is used to place graphic or textual objects in the left margin.