UNIX Writing Tools Primer for CS Graduate Students: Ispell, LaTeX, BibTex, Xfig, GNUPLOT, XV, etc.

Dr. David W. Juedes

January 1, 2001

1 Introduction

As graduate students in computer science at Ohio University, you will be expected to write reports for classes, conference papers, journal papers, theses or dissertations. It is expected that such documents will be well written and carefully prepared. Writing well is a difficult task, even for those of us with many years of experience. It can be even more difficult to prepare professional looking documents because technical writing in computer science (and other disciplines) often involves extensive references, complicated technical diagrams, and complicated formulae. Fortunately, there are a number of relatively standard (and free!) software tools that can help produce professional looking documents. In this primer, a brief introduction is given to each of these software tools. While some of these tools can be used in a stand alone fashion, this document is primarily concerned with the use of these tools for the creation of professionally prepared reports.

Because writing well is difficult, you may want to invest in a good dictionary and other books about writing before you start. There are many excellent books on writing, grammar, and how to avoid common mistakes. My favorites are Nicholas Higham's *Handbook of Writing for the Mathematical Sciences* [3] and Brusaw *et al. Handbook of Technical Writing* [1]. You may also wish to use some of the classics, such Strunk and White's *The Elements of Style* [4]. In addition, there are a number of expository articles about writing that you may want to read. A somewhat recent article by Hemaspaandra and Selman [2] provides excellent advice on writing and producing large technical documents. Their article provides a number of additional references on writing that may be useful to some of you.

This document is a primer on six common software tools for producing technical documents: `ispell`, LaTeX, `bibtex`, `xfig`, `gnuplot`, and `xv`. All of these tools are relatively complex, and hence this primer cannot possibly cover all the many ways of using them. For additional help, you will almost certainly want to purchase a user's guide, such as the classic book *LaTeX: A Documentation Preparation System User's Guide and Reference Manual* by Leslie Lamport [5].

2 Using ispell

As we will discuss below, most UNIX tools for document preparation use flat text files as input. Hence these files can be checked for correct spelling using a standard spelling checker. The spelling checker that I prefer is called `ispell`. The common usage of this command is `ispell filename`. This command scans the file for misspelled words and flags any words that it believes are misspelled.

If `ispell` does not find a word in its dictionary, it looks for "near misses" in its dictionary. For example, consider the file `test` that contains one sentence.

```
ther were two men on the bridge.
```

Upon issuing the command `ispell test`, `ispell` detects that "ther" is not in its dictionary. It then displays

```
ther were two men on the bridge.

00: ether
01: her
02: other
03: the
04: Thea
05: their
06: them
07: then
08: there
09: they
10: Thor
11: tier
```

```
[SP] <number> R)epl A)ccept I)nsert L)ookup U)ncap Q)uit e(X)it or ? for help
```

In this case, the correct replacement is "there," selection number 8. Simply typing 08 will cause ispell to replace "ther" with "there." After this change, `ispell` will exit since all of the remaining words in the file are spelled correctly.

Using `ispell` can be a bit confusing at first. `Ispell` may flag correctly spelled words as misspelled. In these cases, simply type `<space>`, and `ispell` will move on to the next word without changing the flagged word. If `ispell` flags a word as misspelled and does not display the correct replacement, simply type "r" and provide the correct replacement.

`Ispell` will often flag proper names as being misspelled. For example, the proper name "Kolmogorov" is not in the common `ispell` dictionary. However, you can add this name to your private dictionary by typing "i" when `ispell` flags this word as misspelled. `Ispell` places these addition words into a file called `.ispell_english` in your home directory.

Get into the habit of using `ispell` on all of your written documents. It is one of the easiest ways to prevent misspelled word from appearing in your papers. However, note that there is no substitute for carefully proofreading your work.

3 A Primer on LaTeX

LaTeX is a very powerful document preparation system. LaTeX takes as input a text description of a document, and it produces a device independent (dvi) binary description of the document. This device independent binary description can be used to create files that can be printed.

It is common practice to place the text description of the document in a file with a `.tex` extension. Once you have created a text description of your document in a file `filename.tex`, issue the command `latex filename`. This command will compile the description and create the file `filename.dvi`. The device independent description of the document can be viewed on the screen by issuing the command `xdvi filename`. This device independent description can also be used to print the document. To do this, you will want to create a Postscript version of the document. This can be achieved by issuing the command `dvips filename -o filename.ps`. The file `filename.ps` can be printed directly to any Postscript compatible printer via the command `lpr filename.ps`. The file `filename.ps` can be converted to PDF format by issuing the command `ps2pdf filename.ps filename.pdf`.

3.1 Getting Started

LaTeX processes (compiles) a text description of a document into a printable format. This text description must begin with a command that describes the type of document to be created. Typically, the first line of a LaTeX file is the command

```
\documentclass[11pt]{article}
```

that indicates that the document should be processed as an "article" (most conference and journal papers should be formated that way) and that the document should used 11 point fonts. After this command, it is often desirable to include standard packages of commands for use within the document. This can be done with the `\usepackage` command. For example, I often use the commands

```
\usepackage{epsf,amsfonts,amssymb}
```

to include packages for encapsulated postscript figures (epsf), AMS fonts (mathematical fonts), and AMS symbols (common mathematical symbols).

After describing the type of document that you want to create, LaTeX expects that you tell it where a description of the document begins and ends. This is done via the commands `\begin{document}` and `\end{document}`. For example, the following basic document is a valid LaTeX file.

```
\documentclass[11pt]{article}
\usepackage{epsf,amsfonts,amssymb}
\begin{document}
\title{This is my first paper}
\author{Your Name}
\maketitle

\section{Introduction}
Hi Mom.  How does this look?

\end{document}
```

3.2 Organizing your document

LaTeX provides a number of commands for organizing your paper into sections, chapters, etc. Since the `article` style only permits sections, I will not discuss chapters, etc. To create a new section in your document, it suffices to place the command `\section{Section Title}` in the appropriate place. Subsections can be created using the command `\subsection{Subsection Title}`. You can even create sub-subsections using the command `\subsubsection`. By default, LaTeX numbers all sections and subsections. To disable this feature, issue the commands `\section*`, `\subsection*`, and `\subsubsection*` instead of the usual commands.

In addition to sections, you may also wish to label paragraphs and to create lists. Paragraphs can be labeled using the `\paragraph` and `\paragraph*` commands. Lists can be created using the `\begin{enumerate}` `\end{enumerate}` and `\begin{itemize}` `\end{itemize}` commands. As one might expect, these commands can be nested to create outlines, etc. In either environment, the command `\item` denotes each new item. For example, the LaTeX fragment

```
\begin{enumerate}
\item one
\item two
\item three
\end{enumerate}
```

creates the following enumerated list.

1. one
2. two
3. three

Similarly, the LaTeX fragment

```
\begin{itemize}
\item one
\item two
\item three
\end{itemize}
```

creates the following itemized list.

- one
- two
- three

Finally, you may want to include an abstract to your paper. This can be done by using the `\begin{abstract}` `\end{abstract}` commands prior to the main body of the paper.

3.3 Using multiple source files

It is possible to break up a large LaTeX source file into smaller components. Indeed, I strongly recommend this approach for conference papers and theses because it is much easier to manage several smaller files than one large file. LaTeX provides two basic commands for including additional source material. The first command is `\input{filename}`. This command compiles the LaTeX files as if the file `filename.tex` was copied directly into the original LaTeX source file. The other command is `\include{filename}`. This command creates a new page (as if the `\newpage` command were issued) and then includes the contents of the file `filename.tex`.

3.4 Fonts and Underlining

It is possible change the default font size and style within a LaTeX document. To create boldfaced text, use the command `\textbf`. For example, the command `\textbf{This is a bold sentence.}` creates the next sentence. **This is a bold sentence.** To create italic text, use the command `\emph`. To create large text, use one of commands `\large`, `\Large`, `\LARGE`, `huge`, `\Huge`, or `\HUGE`. To create smaller text, use `\small` or `\tiny`. Note that all of the commands that change size need to be placed within `{ }`, or the size of all following text will be changed. Finally, you can underline text using the command `\underline{Text to be underlined}`.

3.5 Centering Text, etc.

To center text (and other objects) on the page, use the `\begin{center}` `\end{center}` environment. For example, the commands

```
\begin{center}
This centers some text.
\end{center}
```

produce the following result.

This centers some text.

3.6 Typesetting mathematics

One of the nicest features of LaTeX is it ability to cleanly typeset mathematics. LaTeX generally assumes that everything in a document is text. Hence, you need to "tell" LaTeX to typeset mathematics. There are several ways to do this. To include mathematics within a line of text, use the `$` delimiters. For example, the commands `$y=x^2$` produce $y = x^2$. To include mathematics as an equation on a line by itself, use the `\[ \]` environment. For example, the fragment `\[ y=x^2 \]` produces

$$y = x^2.$$

The same effect can be attained via the `$$ $$` and `\begin{equation} \end{equation}` environments. For example, `$$ y=x^{x^2} $$` produces

$$y = x^{x^2}.$$

In addition to the basic mathematical environments, LaTeX includes features for arrays of equations. This feature is particularly useful when displaying a sequence of inequalities. To include arrays of equations, use the `\begin{eqnarray} \end{eqnarray}` environment. For example, the following LaTeX fragment

```
\begin{eqnarray}
(a+b)(c+d) &=& a(c+d) + b(c+d) \\
           &=& ac + ad + bc +bd
\end{eqnarray}
```

produces the equation array

$$\begin{aligned} (a+b)(c+d) &= a(c+d) + b(c+d) && (1)\\ &= ac + ad + bc + bd. && (2) \end{aligned}$$

Notice that each equation in the previous equation array is given an equation number. To disable this feature, use the `\begin{eqnarray*} \end{eqnarray*}` equation environment instead of the `\begin{eqnarray} \end{eqnarray}` environment.

Almost anything found in modern mathematics textbooks can be typeset using LaTeX. Unfortunately, there is no simple way to describe all of the possible ways to write mathematics using LaTeX. Anyone writing mathematics using LaTeX needs to use a good reference manual. However, there are few simple rules and some tricks that I have found to be useful.

1. Subscripts and superscripts can be added to any mathematical symbol by using `_` and `^`. In addition, both subscripts and superscripts can be nested. For example, the LaTeX fragment `$x^2_{i_1}$` produces $x^2_{i_1}$.

2. Summations, unions, products, and other mathematical equations place subscripts and superscripts in the correct positions. For example, the fragment `\[ \sum_{i=1}^{\infty} x_i \]` produces

   $$\sum_{i=1}^{\infty} x_i.$$

   The same fragment inside of `$ $` delimiters produces $\sum_{i=1}^{\infty} x_i$. To force the same format inside of a line of text, use the `\limits` command. For example, `$\sum\limits_{i=1}^{\infty} x_i$` produces $\sum\limits_{i=1}^{\infty} x_i$.

   The following list gives some summation like commands.

   (a) Union — `\[ \bigcup_{i=1}^{\infty} A_i \]` produces

   $$\bigcup_{i=1}^{\infty} A_i.$$

   (b) Intersection — `\[ \bigcap_{i=1}^{\infty} A_i \]` produces

   $$\bigcap_{i=1}^{\infty} A_i.$$

(c) Product — `\[ \prod_{i=1}^{\infty} x_i \]` produces

$$\prod_{i=1}^{\infty} x_i.$$

(d) Limit — `\[ \lim_{n \rightarrow \infty} x_n \]` produces

$$\lim_{n \to \infty} x_n.$$

(Notice that the regular union and intersection symbols are `\cup` and `\cap`, respectively.)

3. Occasionally, it is useful to describe a function that is defined conditionally. For example, consider the step function $f : \mathbb{R} \to \mathbb{R}$ that is 0 when $x < 0$ and 1 when $x \geq 0$. This can be written in LaTeX with the following fragment.

```
\[ f(x) = \left \{ \begin{array}{ll}
0 & \mbox{ if $x < 0$} \\
1 & \mbox{ if $x \geq 0$} \\
\end{array}
\right .
```

This fragment produces the equation

$$f(x) = \left \{ \begin{array}{ll} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{array} \right .$$

The commands `\left` and `\right` enlarge the next delimiter (`\{` and `.`) to match the size of the text between `\left` and `\right`. In this case, the period denotes an empty delimiter. The `\mbox{ }` environment allows normal text within math mode. Finally, the `{ll}` in the `\begin{array}` environments denotes two fields in the array, each being left justified. The `&` denotes the end of a field and the `\\` denotes the end of a line.

4. There are a number of common mathematical symbols with their own commands. Note that any command can be negated by placing the `\not` command in front.

- `\leq` == $\leq$
- `\geq` == $\geq$
- `\neq` == `\not =` == $\neq$
- `\subseteq` == $\subseteq$
- `\alpha` == α, `\beta` == β, `\gamma` == γ, etc.
- `\infty` == ∞
- `\ldots` == $\ldots$
- `\cdots` == $\cdots$
- `\cdot` == $\cdot$

Please refer to manual on LaTeX for further information on typesetting mathematics.

3.7 Organizing Mathematics

LaTeX provides a straightforward means to organize mathematics into the theorem – corollary – lemma – proof format found in many journals and textbooks. When done properly, LaTeX can automatically handle the drudgery of numbering, labeling, and crossreferencing such mathematical results. To begin, you must "tell" LaTeX the types of theorem-like environments that you wish to use. This usually happens before the `\begin{document}` command. For example, if you wish to you labeled theorems, corollaries, lemmas, and claims in the text of your paper, include the fragment

```
\newtheorem{theorem}{Theorem}[section]
\newtheorem{corollary}{Corollary}[section]
\newtheorem{lemma}{Lemma}[section]
\newtheorem{claim}{Claim}[section]
```

prior to the `\begin{document}` command. With these definitions, commands such as `\begin{theorem}` can be issued to indicate the statement of a theorem. LaTeX will automatically number such a theorem, and that value can be crossreferenced. In this case, the theorem will be numbered with respect to the current section. For example, the fragment

```
\begin{theorem} \label{Theorem1}
If $a \in {\mathbb R}$, then $a^2 \geq 0$.
\end{theorem}
```

produces the following theorem.

Theorem 3.1 *If $a \in \mathbb{R}$, then $a^2 \geq 0$.*

The `\label` command was included so that Theorem 3.1 could be crossreferenced using the `\ref` command. In this case, the command `\ref{Theorem1}` will create 3.1. As mentioned in the paper by Hemaspaandra and Selman [2], this is the preferred way to reference theorems, etc. since it makes modifying the document later much easier.

Unlike theorems-like environments, creating an environment to give a proof is a bit more complicated. I prefer the following approach. Include the following commands at the beginning of your document.

```
\newenvironment{proof}{\par\noindent \textbf{Proof:}}{\hfill $\Box$}
```

If this definition is included at the beginning of the document, then it can be used as follows.

```
\begin{proof}
If $a \in {\mathbb R}$ then we have two cases.  Either $a \geq 0$, in which
case $a^2 = a \cdot a \geq 0$, or $a = -b <0$ for some $b >0$.  In the
second case, $a^2 = (-b)^2 = b^2 \geq 0$.
\end{proof}
```

This produces the following text.
Proof: If $a \in \mathbb{R}$ then we have two cases. Either $a \geq 0$, in which case $a^2 = a \cdot a \geq 0$, or $a = -b < 0$ for some $b > 0$. In the second case, $a^2 = (-b)^2 = b^2 \geq 0$. □

3.8 Displaying Programs

A simple way to display programs inside of a LaTeX document is to use the `\begin{verbatim} \end{verbatim}` environment. For example, the fragment

```
\begin{verbatim}


main() {
    int i;
    i++;
}

\end{verbatim}
```

displays the following code fragment.

```
main() {
    int i;
    i++;
}
```

3.9 Providing Citations and References

In any scholarly work, you will want to provide references to prior work. In LaTeX, there are many ways to do this. However, the way that I will describe is perhaps the most widely accepted approach. To provide citations to work mentioned in references section of your document, use the command `\cite{ref1,ref2,ref3,....}`, where `ref1`, `ref2`, and `ref3` are labels that refer to the appropriate references. For example, the command `\cite[etc.]{Brusaw,Higham,Lamport94}` produces the citation [1, 3, 5, etc.] in this document. In general, the labels used inside of the `\cite` command should come from a `bibtex` database. For this document, I used a `bibtex` database called `notes.bib`.

The command `bibtex` is used to automatically create bibliographies. To automatically create a bibliography for your document via `bibtex`, place the commands

```
\bibliography{notes}
\bibliographystyle{plain}
```

just prior to the `\end{document}` command at the end of your document. After using the command `latex filename`, LaTeX will create a `.aux` file that contains references to all citation labels. In this document, all of these citation labels correspond to database entries in the `bibtex` database `notes.bib`. Issuing the command `bibtex filename` will cause `bibtex` to search the `bibtex` database `notes.bib` for the referenced labels. Then, `bibtex` will create a bibliography (`filename.bbl`) containing the appropriate bibliographic entries. Issuing the command

```
latex filename
```

again (and again, if necessary) will add the bibliography and build the correct citations into the document. Note that more than one `bibtex` database can be searched. To denote that more than one `bibtex` database should be searched, simply provide additional filenames in the `\bibliography{}` command.

If you wish to include references in your bibliography that are not cited in the text of your document, simply use the `\nocite{label}` command.

4 Using and Creating BibTeX Databases

First, let me mention that one of the best ways to use `bibtex` is to make use of existing `bibtex` databases. The web-site `http://liinwww.ira.uka.de/bibliography/index.html` contains a huge (searchable) repository of `bibtex` databases. Make use of this resource! The `bibtex` databases on this web-site can be download directly, or individual entries can be added your own `bibtex` database.

While the aforementioned web-site contains a huge number of `bibtex` database entries, it is certainly not comprehensive. So, you will almost certainly want to create your own personal `bibtex` databases. As with LaTeX documents, `bibtex` databases are stored as flat files. The following list contains examples of some of the most prevalent `bibtex` database entry types.

1. **Journal articles** — The `bibtex` database record for a journal article usually has the following format.

   ```
   @article{LABEL,
   author = "author1 and author2 and author3",
   title = "Full article title",
   journal = "Journal Title",
   volume = "Volume Number",
   number = "Number",
   year = "Year",
   pages = "page numbers"
   }
   ```

   For example, the `bibtex` entry for the paper by Hemaspaaandra and Selman [2] is as follows.

   ```
   @article{Lane98,
   author = "L.~A.~Hemaspaandra and A.~L.~ Selman",
   title = "Writing and Editing Complexity Theory: Tales and Tools",
   journal = "SIGACT News",
   volume = 29,
   number = 4,
   year = "1998",
   pages = "20--27"
   }
   ```

2. **Books** — The `bibtex` database record for a book usually has the following format.

   ```
   @book{LABEL,
   author = "author1 and author2 and author3",
   title = "Book Title",
   year = "Year",
   publisher = "Publisher",
   edition = "edition",
   address = "address of publisher"
   }
   ```

For example, my database entry for the book by Strunk and White [4] is as follows.

```
@book{Strunk,
author = "W.~Strunk Jr. and E.~White",
title = "The Elements of Style",
publisher = "Macmillan Publishing Company",
edition = "Third",
address = "New York",
year = "1979"
}
```

3. **Papers in Conference Proceedings** — The `bibtex` entry for a paper that appeared in a published conference proceedings has the following format.

```
@inproceedings{LABEL,
author = "author1 and author2 and author3",
title = "Full article title",
booktitle = "Title of Proceedings",
publisher = "Publisher",
address = "Address of Publisher",
year = "Year",
pages = "Page Numbers"
}
```

For example, the `bibtex` entry for a conference paper that I wrote with Lutz and Lathrop in 1993 has the following format.

```
@InProceedings{ICALP::JuedesLL1993,
  title =        "Computational Depth and Reducibility (Extended
                 Abstract)",
  author =       "David W. Juedes and James I. Lathrop and Jack H.
                 Lutz",
  editor =       "Svante Carlsson {Andrzej Lingas, Rolf G. Karlsson}",
  booktitle =    "Automata, Languages and Programming, 20th
                 International Colloquium",
  address =      "Lund, Sweden",
  month =        "5--9~" # jul,
  year =         "1993",
  series =       "Lecture Notes in Computer Science",
  volume =       "700",
  publisher =    "Springer-Verlag",
  pages =        "277--288",
}
```

5 Using Xfig to create figures in LaTeX

Often, it useful to include drawn figures inside of LaTeX documents. Again, there are many ways to do this. The approach described in the following paragraphs uses the drawing program `xfig` to create encapsulated postscript drawings that are included into LaTeX documents.

5.1 Creating EPS files in xfig

To run `xfig`, simply execute the command `xfig filename.fig`. The `xfig` program is a WYSIWYG, and hence it is relatively straightforward to use. Once you've drawn a figure in `xfig`, you can include this picture into your LaTeX document as follows. First, click on the bottom labeled "export" in `xfig` and select "Encapsulated Postscript" as the output language. Then, click on the button labeled "export." This should create a file called `filename.eps`. You can view this file via the command `ghostview filename.eps`.

5.2 Including EPS files in LaTeX

To include the encapsulated postscript figure `filename.eps` into your document, first make sure the package `epsf` is in use. (Include the command `\usepackage{epsf}` at the beginning of your document, if necessary.) Then, create a figure inside of your LaTeX using the following commands.

```
\begin{figure}
\begin{center}
\leavevmode
\epsfxsize 5.0 in
\epsfysize 4.0 in
\epsffile{filename.eps}
\end{center}
\caption{Caption for Figure} \label{figlabel}
\end{figure}
```

The commands `\epsfxsize` and `\epsfysize` scale the encapsulated postscript figure to fit the given dimensions. It is common to omit one of `\epsfxsize` and `\epsfysize` since using both may scale the figure in unintended ways. The command `\epsffile{filename.eps}` simply provides the filename of the included encapsulated postscript figure.

5.3 Creating Xfig drawings with LaTeX labels

The `xfig` program can draw text labels inside of figures. However, the text features of `xfig` are relatively primitive in comparison to LaTeX. Often, it is desirable to use the power of LaTeX for labels inside of figures. This can be done using `xfig`; however, it requires some work.

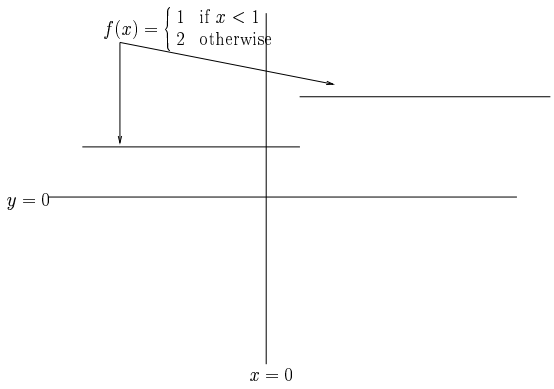

Figure 1: A Figure that combines LaTeX and EPS

Consider Figure 1. This figure was created by first creating text labels in `xfig` with the appropriate LaTeX commands. For example, the top label was written as

```
$f(x) = \left \{ \begin{array}{ll} 1 & \mbox{ if $x<1$ } \\
2 & \mbox{ otherwise} \end{array} \right . $
```

All labels were created with the "text flag" **special** set to "special" instead of "normal." This causes the labeled to be handled specially during the export process.

Once the appropriate LaTeX labels were created, the figure was exported using the language "Combined PS/LaTeX (both parts)." This creates two files: `figure.pstex_t` and `figure.pstex`. To create an encapsulated postscript file, a latex file `figure.tex` was created. The file `figure.tex` contained the following commands.

```
\documentclass{article}
\usepackage{epsfig}
\setlength{\textwidth}{100cm}
\setlength{\textheight}{100cm}
\begin{document}
\pagestyle{empty}
\begin{center}
\input{figure.pstex_t}
\end{center}
\end{document}
```

This file was then compiled using `latex filename.tex`. Finally, an encapsulated postscript version of this file was created using the command `dvips -E filename -o filename.eps`.

6 Using GNUPLOT and LaTeX

A simple and very useful plotting program is called `gnuplot`. You can use `gnuplot` to create figures suitable for inclusion in LaTeX documents. This section provides a brief introduction to `gnuplot` and

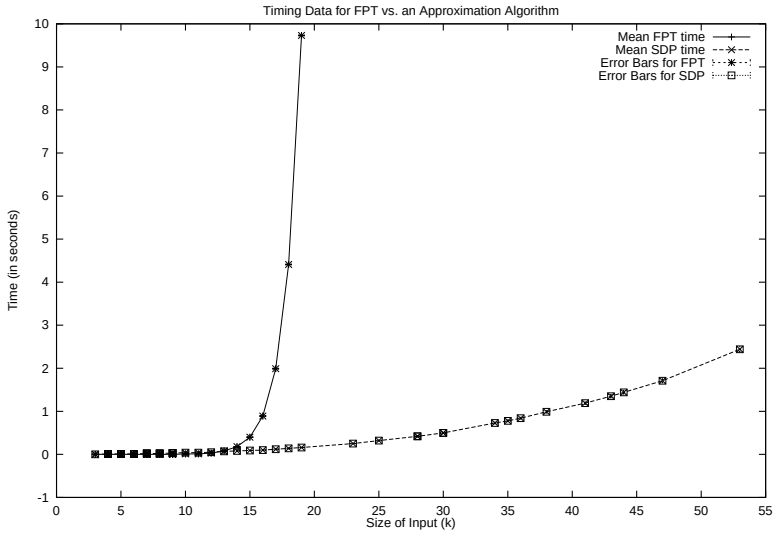

Figure 2: A Figure created by GNUPLOT

shows how to use `gnuplot` to create encapsulated postscript figures. These encapsulated postscript figures can be included directly into LaTeX documents as described in the previous section.

Like LaTeX, `gnuplot` uses flat text files to control the way it generates its output. In the case of `gnuplot`, there are usually several files used as input. The most important file is the file that describes the plot. The following is a relatively straightforward example.

```
#
# These are where comments go
#
set term postscript eps 12
set title "Timing Data for FPT vs. an Approximation Algorithm"
set autoscale xy
set ylabel "Time (in seconds)"
set xlabel "Size of Input (k)"
plot  'meanfpttime.dat' title "Mean FPT time" with linespoints,
'meansdptime.dat' title "Mean SDP time" with linespoints,
'meanfpttime.dat' title "Error Bars for FPT" with yerrorbars,
'meansdptime.dat' title "Error Bars for SDP" with yerrorbars
```

For the sake of argument, assume this file is called `controller.gpt`. This file tells `gnuplot` to produce encapsulated postscript (using 12pt fonts) as output. To create the appropriate eps file, issue the command `gnuplot controller.gpt >plotfile.eps`. In this case, `gnuplot` produces the plot given in Figure 2.

The `gnuplot` file `controller.gpt` requires a bit of explanation. There are two data files associated with this plot:`meanfpttime.dat` and `meansdptime.dat`. Each datafile contains three values per line. For example, `meanfpttime.dat` contains the following values

```
 3       0.00       0.00
 4       0.00       0.00
 4       0.00       0.00
 4       0.00       0.00
 4       0.00       0.00
 5       0.00       0.00
 5       0.00       0.00
 5       0.00       0.00
 5       0.00       0.00
 6       0.00       0.00
 6       0.00       0.00
 6       0.00       0.00
 6       0.00       0.00
 7       0.00       0.00
 7       0.00       0.00
 7       0.00       0.00
 7       0.00       0.00
 8       0.00       0.00
 8       0.00       0.00
 8       0.00       0.00
 8       0.00       0.00
 9       0.00       0.00
 9       0.00       0.00
 9       0.00       0.00
 9       0.00       0.00
10       0.01       0.01
11       0.01       0.01
12       0.03       0.00
13       0.08       0.00
14       0.18       0.00
15       0.40       0.00
16       0.89       0.00
17       1.99       0.01
18       4.41       0.01
19       9.73       0.01
```

Each line contains a value that corresponds to the size of input for some test case, the mean time required by the program on that test case, and the standard deviation of the amount of time taken by the program on that test case. This data can be plotted in several ways. First, it can be plotted using "linespoints" as in the following line.

```
plot  'meanfpttime.dat' title "Mean FPT time" with linespoints,
```

The "linespoints" style connects each point in the data file with the previous point in the data file via a straight line. Similarly, the file can be plotted using the line

```
plot  'meanfpttime.dat' title "Error Bars" with yerrorbars,
```

This creates errorbars about each point of size given by the third element per line.

The remaining commands in the file `controller.gpt` provide the labels for the title, labels for the x and y axes, and scaling information. For additional help on `gnuplot`, simply type `gnuplot` and enter `help` at the prompt.

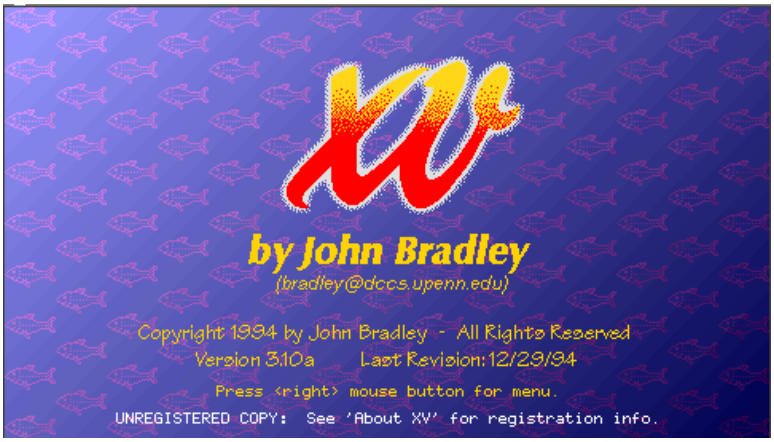

Figure 3: A Figure created by XV

7 Using XV to grab windows

Finally, it is occasionally useful to include screen shots inside of a LaTeX document. This is especially true for graphics applications. Anything that can be displayed in an X-window can be saved and included in a LaTeX document by using `xv`. To do so, simply type `xv`. This will pull up the `xv` program. Clicking on the right mouse button will pull up the control panel for `xv`. If you click on the button labeled "grab," you can capture a any currently displayed window. The captured window can them be cropped and otherwise manipulated via `xv`. To include this window in a LaTeX document, follow the next three steps. First, save the picture in postscript format as `filename.ps`. Next, convert the file to encapsulated postscript format by issuing the command `ps2epsi filename.ps filename.eps`. Finally, include the encapsulated postscript document into LaTeX via the standard figure environment.

```
\begin{figure}
\begin{center}
\leavevmode
\epsfxsize 4.0 in
\epsffile{xv.eps}
\end{center}
\caption{A Figure created by XV} \label{xvfig}
\end{figure}
```

For example, the above steps could be used to create the screenshoot of the `xv` window found in Figure 3.

References

[1] C. T. Brusaw, G. J. Alred, and W. E. Oliu. *Handbook of Technical Writing*. St. Martin's Press, New York, third edition, 1987.

[2] L. A. Hemaspaandra and A. L. Selman. Writing and editing complexity theory: Tales and tools. *SIGACT News*, 29(4):20–27, 1998.

[3] N. J. Higham. *Handbook of Writing for the Mathematical Sciences*. SIAM, Philadelphia, second edition, 1998.

[4] W. Strunk Jr. and E. White. *The Elements of Style*. Macmillan Publishing Company, New York, third edition, 1979.

[5] Leslie Lamport and Duane Bibby. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, MA, 1994.