rather than bridged the "great trench" between theory and data.

Part of the difficulty lay in the diverse backgrounds of the participants. Some were more comfortable with theory, some with empirical testing. The faculty therefore tried to present topics related to both individually as well as the link between the two. Still, students from both sides of the divide felt that presentations should have more consistently addressed the link, rather than either individually.

There also seemed to be a certain gravitational pull of the voting models literature. No other formal literature in political science can match it in terms of volume of articles. We were drawn in by their strong pull perhaps a little longer than we should have been, in light of the diverse interests of the group.

Just Do EITM

Those reservations notwithstanding, my experience was extremely valuable. I would strongly recommend participation in future workshops. The organizers also sought immense amounts of feedback from all of the participants, and I have no doubt that they will respond to our criticisms as best as is possible. As another participant said, creating the perfect workshop linking theory and data may be a challenge for our generation.

Articles

The LaTeX Corner: Wielding LaTeX To Greater Effect

Jeff Gill
University of Florida
jgill@polisci.ufl.edu

LaTeX is a wonderful and addicting tool. I have not used a word-processor in ten years and I even write letters to my mom in the TeX environment. Over time a number of tricks and shortcuts become second nature to users, and it's always fun to swap hints with people at meetings. Usually these conversations are quickly forgotten and we all return from Chicago (or elsewhere) to reinvent the wheel. I have *never* overheard, incidently, conversations about the latest MS-Word trick. Why is that? Despite the obvious selection bias of the conversations that I might be around, the real reason is that with proprietary software it really doesn't make sense to invest oneself in the nuances since it is the vendors' prerogative is to make substantial changes on each new version. Of course it could also be that using word processors is just unpleasant in general.

In the spirit of such conference exchanges this column presents a loosely organized potpourri of various helpful hints that I've accumulated that are not widely known or appreciated (i.e. not prominent in the standard references). None of these are going to change anyone's life, but I think that in sum, they might save some anguish and perhaps lead to nicer typesetting. In any case, I now have a document to assign to graduate students.

Clean Code

There are a few simple programming practices that will make your source-code more readable and in some cases improve the quality of the resulting document. LaTeX doesn't care about spacing in files with the exception of line-feeds, so if it is possible to write your source document to reveal structure then this often leads to better writing and easier debugging. For instance, hunting through long sections to find footnote structure can be annoying, so a good strategy is to indent footnote contents within paragraphs:

```
Perhaps the best known prison rodeo is in Angola,
Louisiana.\footnote{Angola runs
        every Sunday in October and
        one weekend in April each year.}
However, Texas has a long tradition of running prison
rodeos as well.
```

Also, anything after `\end{document}` is ignored, so you can use this area of the file for things like notes, extra tables, and unused references. Few people bother, but you can insert comments anywhere in the document to yourself or coauthors that will not be typeset if they follow the "`%`" character on a given line. For longer sections of text to be unprocessed it might be more convenient to use the `\begin{comment}` and `\end{comment}` statements, although these can sometimes hide amongst other statements in the file and cause one to lose track of what is "in" and what is "out."

Some small things... Inserting two spacings before a new sentence is not necessary since TEX controls intersentence spacings, but it makes the source file easier to read. Be sure to use the tilde character to keep figure and table reference numbers next to the word (rather than possibly being split over lines): `Figure~\ref{cowboy.figure}` and `Table~\ref{angola.history.table}`. Of course, it is well-known that such table and figure references shouldn't be hard-coded: `Table 1`, `Figure 3`. Computer code and URLs look much nicer when typeset with the `\texttt{}` command or the `verbatim` environment (which requires the `verbatim` package). Don't use the keyboard's double quote: `"`. In LATEX the way to get correct quotations is to use two left-facing single quotes at the beginning, ``, and two right-facing single quotes at the end, ''.

The standard way to input references is with `\bibitem{}` or `\item[]` for each entry and then a section at the end of the document delimited by `\begin{thebibliography}{99}` and `\end{thebibliography}` ({99} prepares LATEX for up to 99 citation reference numbers). This gives a nicely formatted reference section (including automatic sectioning with title), but unfortunately the default is more suited to referencing in a number of natural sciences rather than political science:

> [1] Bergner, Daniel. 1998. *God of the Rodeo: the Search for Hope, Faith, and a Six-second Ride in Louisiana's Angola Prison.* New York: Crown Publishers.

Years ago Jason Wittenberg gave me the following code which makes the reference section look appropriate for political science work: no citation numbers in the text or the references, and a standard indentation scheme. First, place for convenience the following in the preamble.

```
\renewcommand{\bibitem}{\vskip
    2pt\par\hangindent\parindent\hskip-\parindent}
```

Then begin the reference section with:

```
\section*{References}
\mbox{} \baselineskip=6pt \parskip=1.1\baselineskip
    plus 4pt minus 4pt \vspace{-\parskip}
```

and simply start each bibliography entry with `\bibitem`. There is no longer a need to put braces or brackets after `\bibitem` or to include an end statement to indicate the end of the reference section. The result looks like:

Bergner, Daniel. 1998. *God of the Rodeo: the Search for Hope, Faith, and a Six-second Ride in Louisiana's Angola Prison.* New York: Crown Publishers.

This code is easy is drop into a file and it avoids some of the extensive and confusing trickery that I have seen with other solutions. Eventually, though, one might want to migrate to the elaborate and flexible BIBTEX package which provides a personalized central inventory of references and can save a lot of work once setup.

Miscellaneous Math

It makes sense to always load the $A\mathcal{M}S$; packages (`\usepackage{amsfonts, amssymb, amsmath}`): better fonts, a lot more features, and you don't have to remember whether something is in the packages or not. The extra compile time is minimal anyway. The key to creating readable math when you look at the file two years later is hierarchical organization. Specifically, use spaces, tabs, and line feeds to show the structure of the formula so you can read and edit it much easier later. For example contrast the organized math (from actual work):

```
\begin{align}\label{QH.likelihood.separation}
\ell(\T|\X_{obs})
  &= \underbrace{ \int\ell(\T|\X_{obs},\X_{mis})
       f(\X_{mis}|\X_{obs},\T^{(0)}) d\X_{mis} }
           _{ Q(\T|\T^{(0)}) }
       \nonumber \\
  &- \underbrace{ \int\log f(\X_{mis}|\X_{obs},\T)
       f(\X_{mis}|\X_{obs},\T^{(0)}) d\X_{mis}. }
           _{ H(\T|\T^{(0)}) }
\end{align}
```

with an unorganized version of the same code, which provides the exact same typeset output:

```
\begin{align}\label{QH.likelihood.separation}
  \ell(\T|\X_{obs})&=\underbrace{\int\ell
  (\T|\X_{obs},\X_{mis}) f(\X_{mis}|\X_{obs},
  \T^{(0)})d\X_{mis}}_{Q(\T|\T^{(0)})}\nonumber\\
  &-\underbrace{\int\log f(\X_{mis}|\X_{obs}
  ,\T) f(\X_{mis}|\X_{obs},\T^{(0)})d\X_{mis}.}
  _{H(\T|\T^{(0)})}
\end{align}
```

This advice applies equally to formatting tables since it is very easy to leave out an `&` character when the columns are not lined up (even though they obviously do not have to be). Above, `\X` and `\T` are user-defined shortcuts for $\mathbf{X}$ and $\boldsymbol{\theta}$ that make the source file easier to write and read. These shortcuts are defined in the preamble with:

```
\newcommand{\T}{{\boldsymbol{\theta}}}
\newcommand{\X}{{\mathbf{X}}}
```

These are immensely time-saving and more complex forms can be created without a lot of energy:

```
\newcommand{\SIinv} {{\|\boldsymbol{\varSigma}\|^{-1}}}.
```
Note also that shortcuts are not confined to the math environment. The font shortcut for displaying computer code in this document was created with:

```
\newcommand{\code}[1]{\texttt{\small#1}}
```

where the different treatment of `\texttt` and `\small` reflects differences in how these font characteristics are programmed in TEX, and the `[1]` plus `#1` indicates that one argument only is processed in this new function. Sometimes the `newcommand` strategy is not flexible enough, particularly when there are optional arguments involved, and it is possible to define a new command from the original definition in the `latex.ltx` source file. Never modify this file directly, instead create a local style file such as `mycustom.sty` and modify the structure in there (you will have to add `\usepackage{mycustom}` to your preamble). For instance the same effect as above could be created in the style file by writing:

```
\DeclareTextFontCommand{\code}{\codefamily}
\DeclareRobustCommand\codefamily
        {\not@math@alphabet\ttfamily\mathtt\small
        \fontfamily\ttdefault\selectfont}
```

which is exactly the pertinent code in `latex.ltx` except `\texttt` based on the `\ttfamily` has been redefined to be `\code` based on the new `\codefamily`, which has `\small` added to the specification list.

Writing simply `exp` and `log` in math mode looks ugly since they will be automatically italicized. Most people therefore use `\text{exp}` and `\text{log}` to give a consistent font with the rest of the document. A minor trick, which is cleaner and gives the exact same result, is `\exp` and `\log`. This also works with limit functions and common trigonometric functions, `\lim`, `\sup`, `\inf`, `\sin`, `\tan`, etc. A related shortcut that has great flexibility is the modulo function. Typesetting `a = b \pmod{c}` in math mode gives $a = b \pmod{c}$, which handles the spacing perfectly (an annoying task with `\;` and other spacers). Also `a = b \mod{c}` provides $a = b \mod c$, `a = b \bmod{c}` provides $a = b \mod c$ (slightly less space), and `a = b \pod{c}` provides $a = b \pod{c}$, all with perfect math spacing.

A strangely frequent mistake is to use `\Pi` when it is appropriate to use `\prod`. Contrast the mistaken version: $L(\mathbf{x}, y, \theta) = \Pi_{i=1}^{n}\Pi_{j=1}^{k}[\Phi(\theta_j - \mathbf{x}'\boldsymbol{\gamma}) - \Phi(\theta_{j-1} - \mathbf{x}'\boldsymbol{\gamma}]^{z_{ij}}$, with the corrected version: $L(\mathbf{x}, y, \theta) = \prod_{i=1}^{n}\prod_{j=1}^{k}[\Phi(\theta_j - \mathbf{x}'\boldsymbol{\gamma}) - \Phi(\theta_{j-1} - \mathbf{x}'\boldsymbol{\gamma}]^{z_{ij}}$. Similarly, it seems surprisingly common to see x or $*$ instead of `\times` to get $\times$. A related type of error is incorrect parenthesis sizing; usually too small. To get parenthesis sizes that match the height of the math within, use `\left(` and `\right)` (this also works with other delimiters such as brackets, [, bars |, and double bars ‖. The only caveat is that LaTeX uses these *contextually* in formulas so you cannot break up a left-right pair across lines in the align environment. In this case you will have to manually size the parentheses with: `\bigl(`, `\Bigl(`, `\biggl(`, and `\Biggl(`.

LaTeX has some "nested" characters that improve the typesetting of certain repeated forms. For instance, $a << b$ looks dumb, but $a \ll b$ (created with `a \ll b`) looks perfect (there is also the $\lll$ symbol if one wants to be more emphatic). In the same vein, it does not look good to iterate integrals: $\int\int\int f(\zeta, \xi, \nu) d\zeta d\xi d\nu$. Instead use the provided construct, `\iiint`, to get $\iiint f(\zeta, \xi, \nu) d\zeta d\xi d\nu$. There are two to four iterations provided by the number of `i`'s in front of `nt`, and for more than four one can use: `\idotsint` to get $\int \cdots \int$. It is also much nicer to use `\ldots` rather than "..." since LaTeX will give distinct spacing. Note as well that there is `\cdots` for centered dots, `\vdots` for vertical dots, and `\ddots` for diagonal dots (useful in matrix expressions).

The align math environment is tremendously useful and flexible for multi-line math where you want to line up the equations vertically according to some character, usually an equal sign. If it is desired to put extended text somewhere in-between one can stop the align environment and then start a new one after the text. However, this may make it hard to continue the alignment process, so LaTeX has the `\intertext{}` command for inserting text without losing the alignment process. Simply insert the command with the desired text (no need to supply the `\text` command) and the right thing will happen (the new line command is not needed at the end of this line: `\\`).

Terrific Tables

Tables can consume an immense amount of time in order to look really clean and sharp. However, this time can be quite well-spent since casual readers often look only at the table to see model results. The general structure of tables is to embed `tabular` in the `table` environment, for instance:

```
\begin{table}[t] \begin{center}
\caption{\textsc{Angola Prison Rodeo History}}%
        \label{angola.history.table}
\vspace{0.07in}
\renewcommand{\arraystretch}{1.2}
\begin{tabular}{cr|l}
  \hline
  \multirow{4}{3mm}{
```

```
      \parbox[h]{3mm}{
         \begin{turn}{90}Milestones\end{turn} } }
  & 1965 & First rodeo (for prisoners \&
           staff only) \\
  & 1967 & Opened to the public \\
  & 1972 & Professional Rodeo Cowboys Association
           rules adopted \\
  & 1997 & Stadium expanded \\
  \hline
\end{tabular}\end{center}\end{table}
```

Here we get "Milestones" vertically in the first column using `multirow` from the package of the same name (see also the `rotate`, and `sideways` environments provided by the same `rotating` package). Note that this rotating effect will not show up with a `DVI` viewer because it is implemented at the `postscript` step. The use of `multirow` is slightly more involved than `multicolumn` because it is necessary to stipulate the spacing (`3mm` here).

The `tabular` command dictates the number of columns, the alignment in these columns, and whether or not there should be a vertical bar of separation (as done above). What can be frustrating is processing format exceptions within the table. That is, situations where one table cell deviates from the rest of the column's specification. TPM readers probably already know that the `multicolumn` statement allows text to cross numbers of columns, but it seems less-well known that the multicolumn statement with "1" as column width can be used to customize a single cell distinct from the rest of the column, for instance `\multicolumn{1}{|c}{1997}` would make the "1997" contents in the bottom left cell center-aligned instead of right-aligned, and move the wall to the left side from the right.

Some useful table options include: `\cline2-3` for underlining a subset of columns (columns 2 and 3 of the next line here, as opposed to `\hline` which underlines the entire table row), use of `p{10mm}` in the `tabular` line to give specific column spacing, the `dcolumn` package for more flexibility in controlling column formats, and using subenvironments like paragraph boxes (`parbox`) within cells.

Regretfully, it is often necessary to use the `\vspace` command to get nice looking separation between the caption and the table. Of course this is trial and error work. There are many ways to impose different vertical spacing in the table than in the text, but I rather like to use `\renewcommand` because it gives direct control.

A really nifty way to put confidence intervals (as opposed to moronic stars) into tables is to code the structure of the confidence interval into the `tabular` command. This makes the formatting of the table easier for you and usually produces very nicely lined-up columns. Unfortunately when the numeric values differ considerably in magnitude it is necessary to use spacing characters in order to line up the decimal points:

```
\begin{tabular}{lr@{:}l}
  \hline
  $\alpha_{\tau}$ & [~~~0.0026 & ~~~0.0511] \\
  $\beta_{\tau}$  & [109.3254  &  875.0422] \\
  $\tau_c$        & [~~~6.9151 & ~~~9.5301] \\
  \hline
\end{tabular}
```

Bad Behavior

The package `verbatim` sometimes does not "play well" with other packages and commands. It is very difficult to create a new environment (`\newenvironment` in the preamble) that includes the `verbatim` environment. Furthermore, commands like `\begin{small}` and `\end{small}` cannot be placed on the same line in the source file. However they can be on adjacent lines. Oddly enough, other formatting commands *can* be placed on the same line as the `verbatim` commands, like `\renewcommand{\baselinestretch}{1.00}`. It is also impossible to put the `verbatim` environment inside a `parbox`, but possible inside a `minipage`.

Managing "floats" can be a bear. Figures and tables that LaTeX moves around itself during typesetting are called floats. This arrangement is necessary since authors cannot see where the page breaks will be when editing the source file. The agony is that sometimes LaTeX will move these too far from the relevant discussion or clump several of them awkwardly on the same page. The primary weapon at one's disposal is the float placement specifier that is part of the `table` and `figure` statements: `\begin{table}[tbhH]`, where `t` means top of the page, `b` means bottom of the page (and sometimes unintentionally bottom of the document), `h` means try hard to put it *here* in the text, and `H` means put it *right here* even with ugly consequences. Some of these can be combined, and one can also add `!` in order to ask LaTeX to try really hard to comply with the specification. Sometimes it is helpful in this cause to resize figures (easy with `epsfig` and `includegraphics` since they have a sizing option), or tables (by changing font sizes or column widths). Other times this can be difficult such as with large tables like those often requiring the use of `sidewaystable`. Although it is rarely used, a nice effect that is easy to obtain with smaller tables and figures is to have the text wrap around the float using `floatingfigure` or `wrapfigure`, as done above. In reality there is often considerable trial and error so the standard, but often ignored, advice is to wait

Judge	Rider A	Rider B
1	3	0
2	3	0
3	3	2
4	0	5
5	0	5
Total	9	12

until all other editing is done before worrying about moving floats around. Chapter 6 of the indispensable LaTeX *Companion* (Goosens, Mittelbach, and Samarin [1994], Addison Wesley), has a wealth of useful advice on managing floats.

To quote one of my students, "LaTeX likes to complain." In particular it likes to complain during the typesetting process when it has a problem fitting the text into the paragraph width specified so that the right margin is justified. So it is extremely common to see "underfull" and "overfull' statements scroll by such as:

```
Overfull \hbox (4.4555pt too wide) in paragraph
    at lines 1020--1020
Underfull \hbox (badness 10000) in paragraph
    at lines 727--745
```

Underfull indicates that LaTeX could not nicely typeset the indicated lines(s) within the specifications and lets you know that it under-fit the line (too much end space) with warning levels given according to the parameter `\hbadness`. Here *badness* ranges from 0 badness for a perfect match, and 10,000 badness for something hopeless. The default badness is 1,000, and setting: `\hbadness=10000` will remove all such warnings from the screen. With regard to overfulls, Knuth (TeX's creator) decided that it was better to let text occasionally foray into the right margin (the overfull) rather than have really ugly spacings or weird hyphenations on that line. The amount of overfull complaining is determined by the `\hfuzz` (horizontal fuzz) parameter (defaulted to 0.1), and the algorithm is controlled (partially) by `\tolerance` (defaulted to 200). Setting something like `\hfuzz30pt` will generally prevent reporting of overfulls, and the tolerance can be made bigger if more space is to be allowed between words (exceeding 9999 apparently gives ugly results though). The algorithm here is reasonably sophisticated and it is not terribly common to have to intervene. Furthermore, since there are 72.27 points to an inch, many of the reported "problems" are not really worth fretting about.

LaTeX does not always know where to hyphenate words. This is especially true for highly technical or unusual words that are fairly long. So sometimes the word will be hyphenated in an inappropriate place. This can be solved by "telling" LaTeX where to hyphenate a specific word. Simply place the instructions in the preamble according to: `\hyphenation{non-con-verg-ence}`.

Hasty Conclusion

I hope that this little exercise is found to be helpful. Presumably we will see some really interesting additions to *The* LaTeX *Corner* on subjects like: advanced graphics, dissertation/book strategies, multi-line math tricks, METAFONTand METAPOST, as well as integration with `html` and `xml`.

Have You Mastered Tables and Graphics in /LaTeX? Share Your Secrets with TPM's Readers.

Contact Heather Ondercin at hlo114@psu.edu

Working on You Dissertation in LaTeX? Share Your Experience With Your Peers.

Contact Heather Ondercin at hlo114@psu.edu