

# How to Write a Shiny Paper<sup>☆</sup>

Thomas de Graaff

*Department of Spatial Economics, Vrije Universiteit*

---

## Abstract

This paper explains what tools you need to write a paper that (in this case) looks like an Elsevier article. The underlying theory, however, of why certain tools are better than others is applicable to almost all types of output (e.g., theses, papers, presentations, posters, and articles in any other format). With this paper I hope that (advanced) students are willing to give the tools discussed a try—even if the learning curve in the beginning might be quite steep.

*Keywords:* Automation, workflow, text files, L<sup>A</sup>T<sub>E</sub>X, citation manager

---

## 1. Why bother?

Well, first of all, you obviously want to write a shiny paper; a paper that looks really nice, is easy to read, and where each of the elements—including references, mathematics, tables and figures—have a consistent look. And the point is: this is far more difficult (though possible) with conventional—read Word—tools. Secondly, and more persuasive perhaps, conventional proprietary tools do not lend themselves easily to customisation and automatisation. For instance, it is rather difficult to directly import Stata output in tables in Word; it always has to go indirectly, or using Excel or using Rich Text Format (rtf).

Although this tutorial advocates the use of some particular tools, it actually does not matter that much as long as they are able to use plain text files. Plain text files namely have a number of advantages over other type of files; they are transparent (you can always read and edit them), portable (they are the same on each system) and they are not likely to become obsolete in the coming decades. And a lot of files or even tools just consists of plain text files, such as HTML code, most open source tools and Stata ‘do’ files. Most proprietary tools have formats that are not made of plain text, such as Word, Excel, or powerpoint (as well as the Stata data format by the way). This makes them not only less flexible but the risk of getting stuck in

an obsolete data format<sup>1</sup> is nontrivial, while output is not similar on each operation system or even across versions.

## 2. Managing workflow

It is difficult to define what ‘good workflow’ is, but it certainly has to satisfy each of the following elements to some degree:<sup>2</sup>

1. Transparency: A good workflow organizes the elements of the project logically and clearly, to make it easy for an observer (including yourself) to understand how the pieces come together.
2. Maintainability: A good workflow makes it easy to modify and adapt the project. Standardized script names and good commenting practices are key here.
3. Modularity: A good workflow encapsulates discrete tasks into separate components (e.g. scripts), so that it’s always clear where modifications need to be made (and only made in one place), and components are re-usable for other projects.
4. Portability: A good workflow makes it easy to move the project to another system, or hand it over to another person to work on, in such a way that it can still easily be run elsewhere. (By using relative (not absolute) pathnames is one example.)
5. Reproducibility: A good workflow makes it easy for you, or others, to reproduce your results.
6. Efficiency: A good workflow saves you time, by making it easier to work on the project, and by automating as much of the process as possible.

---

<sup>☆</sup>This paper is written for those STREEM or TSCM master-students who want to write a master thesis in the form of an article and who like to use more sophisticated tools to do so. Note that I do not aim to discuss writing skills, but instead aim to discuss tools for better presentation of research results. Why by the way an Elsevier type of article? Well, it doesn’t matter that much, but I for myself appreciate the format. And it is created by people who actually understand typesetting. So, for the moment we do not want to reinvent the wheel and knick Elsevier’s style file. Welcome to the realm of open source!

*Email address:* [t.de.graaff@vu.nl](mailto:t.de.graaff@vu.nl) (Thomas de Graaff)

*URL:* [www.thomasdegraaff.net](http://www.thomasdegraaff.net) (Thomas de Graaff)

---

<sup>1</sup>The case of the late but much beloved wordprocessor ‘Word Perfect’ comes to mind.

<sup>2</sup>Almost directly taken from this [blog](#); another wonderful source is [Healy \(2013\)](#).

### 3. L<sup>A</sup>T<sub>E</sub>X instead of ‘What You See Is What You Get’ tools

Word and Powerpoint are usually described as ‘What You See Is What You Get’ (WYSIWYG) tools. But what you get is sometimes below par. That’s why some advocate the use of ‘What You Mean Is What You Get’ (WYMIWYG) tools. And these tools usually apply to markup languages. The most famous of these languages is HTML, which is just a bunch of code describing how text *should* look like on screen.

L<sup>A</sup>T<sub>E</sub>X is a markup language as well, that just consists of code describing how a text should look like (on paper, screen or beamer). Instead of a word processing program, L<sup>A</sup>T<sub>E</sub>X is a typesetting program that determines for itself how words, sentences, paragraphs, etcetera, are placed on paper, slide or pdf file. Actually, L<sup>A</sup>T<sub>E</sub>X is a program (or compiler) that is fed by an input text file. Novice users find this most difficult; everything should be more or less coded. However, this allows for great flexibility, transparency, portability, and even efficiency. The same code can namely be used for papers, presentations and posters alike.

Where L<sup>A</sup>T<sub>E</sub>X truly shines is in the scientific domain.<sup>3</sup> If one wants to consistently typeset text, figures, tables, references and especially equations, nothing outperforms L<sup>A</sup>T<sub>E</sub>X.<sup>4</sup> And in the end, that is what you want to do: write a nice shiny paper.

#### 3.1. Where to get it?

One of the nice things about L<sup>A</sup>T<sub>E</sub>X is that it works on (almost) all operating systems. To get it to work you need two things: the program itself and an editor. For windows this usually means downloading [MikTeX](#) and [TeXNicCenter](#) or [TeXworks](#). On a mac, both program and editor (and lots more) can be downloaded from the [MacTeX](#) site with its celebrated TexShop editor.

#### 3.2. I got it, and now what?

There are numerous tutorials on the web, but I recommend to first take a look at the [latex project](#) website, which comes with a whole bunch of tutorials (and yes, they are all free).

Without really going into the nitty gritty of this language, let’s write our quintessential ‘hello world’ program.

```
% hello.tex - A very simple LaTeX example!
\documentclass{article}
\begin{document}
Hello World!
\end{document}
```

<sup>3</sup>But not only given the nice examples on this [website](#).

<sup>4</sup>L<sup>A</sup>T<sub>E</sub>X itself is just a bunch of macros around T<sub>E</sub>X, a typesetting program devised by the notorious computer scientist Donald Knuth. L<sup>A</sup>T<sub>E</sub>X was created by Leslie Lamport in 1984 and T<sub>E</sub>X was created in 1978. Yes, it is that old.

with the following output (but then as a separate document):

Hello World!

Okay, this perhaps doesn’t look like much, but it actually contains quite some interesting elements. First, lines starting with a % sign are used as comments. Second, you first have to specify what kind of text you are typesetting (in this case an article). And finally, every element needs a ‘begin’ and an ‘end’ (just as we all do). In this case, the whole document needs to be specified. Things start to heat up a bit, when you add formulas such as when we add the following lines to our simple example

```
You know, I just discovered that:
\begin{equation}
E = mc^2
\end{equation}
```

with the following output:

You know, I just discovered that:

$$E = mc^2 \quad (1)$$

Where again, we need to put something between a ‘begin’ and an ‘end’ statement, in this case the equation. The code for equations (or math mode) is remarkably similar to that of Excel and does not take long to get used to. If one wants to have mathematics displayed inline it usually is best to use dollar signs such as:<sup>5</sup>

```
 $\frac{\partial u}{\partial x} = -\frac{\partial u}{\partial y}$
\frac{\partial u(x,y)}{\partial x} = -\frac{\partial u(x,y)}{\partial y}
```

which displays as:  $\frac{dx}{dy} = -\frac{\partial u(x,y)/\partial y}{\partial u(x,y)/\partial x}$ . Before we now set off, there is one thing you need to know, and that is the use of packages and classes. Because L<sup>A</sup>T<sub>E</sub>X consists of macros it is completely adjustable and that is exactly what people have done and where the strength of the application can be found; in user-written packages and classes. For example, when you want your paper to look like an Elsevier article (and in this case you do), then use the *elsarticle* class instead of the *article* class.<sup>6</sup>

Usually, the best way of learning this is by looking at examples of others. Take a look, e.g., at the source code for this pdf file. At the top it says use the *elsarticle* with some

<sup>5</sup>It takes some time to get used to all the mathematics commands. Luckily, the internet is full of cheatsheets, wikis and manuals about this, such as [this](#).

<sup>6</sup>This class usually is already installed or can be found on Elsevier’s website or on the repository of L<sup>A</sup>T<sub>E</sub>X packages—[www.ctan.org](#). On Elsevier’s website you can download as well a template, which comes in handy when starting a paper.

options (e.g., the *5p* option states that the text should be typesetted in two-columns—*3p* would indicate one column). Below that I load a bunch of packages which usually have something to do with how to display tables, mathematics, links and references. Then there is some front matter stuff, such as authors, title, addresses and an abstract, after which the main text follows, nicely divided in sections and subsections. The only further thing handy to know in this stage is that an open line indicates a new paragraph.

### 3.3. *Ok, but how about Tables?*

One of the more painful things to do in L<sup>A</sup>T<sub>E</sub>X is composing tables. That’s why you should never do so. Instead, you should let other applications do the trick for you. For instance, R and Stata can easily transform regression output to a L<sup>A</sup>T<sub>E</sub>X format which can be directly inserted in your L<sup>A</sup>T<sub>E</sub>X file—remember, every element here is a text file.<sup>7</sup> The following table gives an example and displays the output of a regression model of car characteristics on car mileage with two specifications.

Table 1: Dep = Miles per Gallon		
	(1) Mileage (mpg)	(2) Mileage (mpg)
Car type	−1.6500 (1.0760)	−2.2035* (1.0592)
Weight (lbs.)	−0.0066 * * (0.0006)	−0.0166 * * (0.0040)
weight_sqr		0.0000* (0.0000)
Constant	41.6797 * * (2.1655)	56.5388 * * (6.1974)
Observations	74	74
R <sup>2</sup>	0.663	0.691
F	69.7485	52.2515

Standard errors in parentheses  
+  $p < 0.1$ , \*  $p < 0.05$ , \*\*  $p < 0.01$

This table is created with only four commands, two to do the regressions in Stata, one to write the output in two columns to a file in L<sup>A</sup>T<sub>E</sub>X format, and the last to import the table in this text file. This approach has numerous advantages, namely: it is very efficient, it is not error prone<sup>8</sup> and you can very easily change the results for different samples, specifications or functional forms.

### 3.4. *And what about figures, diagrams and graphs?*

Fortunately, figures, diagrams and graphs are much easier to handle in L<sup>A</sup>T<sub>E</sub>X than tables. And, as long as

<sup>7</sup>Typical packages for Stata are `outtex` and `outreg2` and for R `xtable` and `stargazer`.

<sup>8</sup>Most mistakes in scientific research are made in pre- or post analysis operations; a.k.a. fumbling a bit with Excel and typing over your results.

you want to draw something in a schematic way, you can even make your own diagrams and graphs using a couple of additional packages!

#### 3.4.1. *Figures*

Because the output of L<sup>A</sup>T<sub>E</sub>X is already in pdf format it can easily incorporate figures from many format, such as the printer languages .pdf and .ps, but as well .jpeg and others. For example, this command:

```
\begin{figure}[h!]
\center
\includegraphics[width=0.3\textwidth]
{ligatures_latex} \caption{Correct
use of ligatures in \LaTeX (Source:
\href{http://nitens.org/ taraborelli/latex}
{Taborelli})}
\end{figure}
```

produces Figure (1):

fire flower fjörd

Figure 1: Correct use of ligatures in L<sup>A</sup>T<sub>E</sub>X (Source: [Taborelli](#))

The command `h!` states that the figure should very explicitly (the `!` command) be placed about here (the `h` command).

When captions are included they are correctly numbered and size and position of the figure (even across columns) can easily be adjusted.

#### 3.4.2. *Graphs & plots*

Creating structured graphs and plots based on data is rather straightforward with the use of appropriate packages. In this case the packages *tikz* and *pgfplots* come in very handy (they also come with very extended manuals full of examples). Suppose that we are interested in depicting the outcomes of a meta-analysis concerning the relation between the elasticity of between migration and imports on the one hand and the elasticity between the migration and exports on the other hand. Moreover, the original results have been written to a simple text file.<sup>9</sup> Then with a couple of commands (I refer to the source of this file to avoid too much clutter) one gets Figure (2).

One of the main advantages in doing it as such, is that typesetting remains consistent. This also applies to symbols and equations in the title(s), legends, tick-marks or even as labels within the graph. Another large advantage is that the slope estimate is automatically generated. Thus, if additional observations (research outcomes) are added to the meta-analysis, then Figure (2) will be automatically updated.

<sup>9</sup>The text file is named ‘migrationFDI.txt’ and is generously provided by Masood Gheasi. The original research is documented in [Ghen et al. \(2012\)](#).

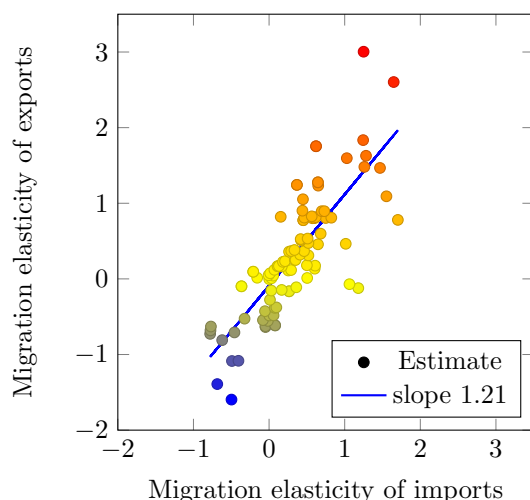


Figure 2: A meta-analysis on the elasticity between migration and trade

### 3.4.3. Diagrams

The *tikz* package can as well be used for drawing diagrams of your own.<sup>10</sup> This requires a bit more code and thought but is able to deliver very sharp structured diagrams with consistent typesetting. And, if you have done it once, you can always reuse the code (e.g., for presentation, handouts or class-slides). Very good sources on the internet are [Tikz examples](#) and the blog of Kevin Goulding [Tikz for economist](#). Once more, the [manual](#) of Tikz is as well quite good and rather elaborate.

One of my own examples is a diagram of an Edgeworth-Box, which is—as you might know—rather painful to do in Excel. Figure 3 has been coded to be used in the second year's course Regional Economics. And the particular advantage of doing it like such is that the parameters (in this case, the endowments of both capital and labour and the parameters of the production function of both countries  $A$  and  $B$ ) are fully adjustable, which might be particular useful for variations in examinations and assignments.

## 4. Always use a citation manager

The first thing I do recommend to all students (and researchers) is to use a citation manager. Even when using applications such as Microsoft Word (the horror...). This is one tool that immediately delivers in terms of efficiency. Even if initially setting up a database with references might be time-consuming it still pays off in later stages. The point is that you only once have to type in a reference within a database. Later on you can always use this reference within the text and the reference or bibliography list should be automatically generated. And there are even reference

<sup>10</sup>There are other ways of creating structured diagrams, both within the and outside the  $\text{\LaTeX}$  environment. An example of the latter is the metapost language.

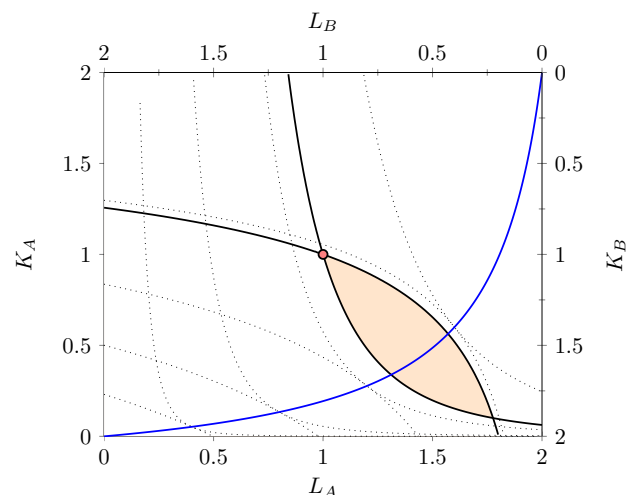


Figure 3: Tikh Diagram of an Edgeworth Box (Source: own code)

managers that can import folders with articles in pdf format and automatically generate a reference database.

Within the  $\text{\LaTeX}$  environment, the application BibTeX is usually used. There are separate editors for this system, but the easiest way is using a more generic reference manager (the free Mendeley application comes to mind) that can write files to .bib files (the BibTeX format) or incorporate references directly within Microsoft Word. In this way, you remain as flexible as possible.

## 5. From here ... into the abyss

Right. so we know now how to write a shiny paper with tables, references and perhaps even customised diagrams. Would that be it? No, it is just the start of a completely automated, customised, transparent and (hopefully) more efficient workflow. For that you need a set of tools that are complementary to each other. And fortunately, most of them are. However, choose your tools carefully. All of them are costly in terms of learning curves and there is only so much time available to you. Having said that, there are a couple of choices you might want to make eventually, of which the following three are probably most important.

First of all, the choice of editor. Of course, TexShop or TeXNicCenter are nice editors, but are mostly suited for  $\text{\LaTeX}$ . In the end, you might want to use a more versatile editor. One editor capable of editing R scripts, stata do files, html code, etcetera. All with proper highlighting and a consistent set of shortcuts. I am not saying which editor you should use. Too many have found their demise at the [editor war](#). There are many good (and free) ones out there. And even the ones you have to pay for are well worth the money. Pick up and stick to it for some time. For OSX there is now a free version of Textmate, which is quite good. And cross platforms, sublime is another editor well worth looking into. If you really want to go nerdy, opt for Emacs of Vi(m). Longstanding, very capable (and fast) editors. However, be aware of a steep learning [curve](#).

Second, as already hinted upon, the combination of L<sup>A</sup>T<sub>E</sub>X with statistical programs such as Stata and R is a killer app. If one has to choose, I would advise you to invest in R. Stata is more an econometric package, while R is much more statistical. And the userbase of the latter is much larger than the former, ensuring that with R most problems dealing with statistics (or even GIS) have already been dealt with. Moreover, there is a very nice editor, Rstudio, able to integrate them both. Heck, you can even weave L<sup>A</sup>T<sub>E</sub>X and R code together (called Sweave or Knitr) with mindblowing results. This comes rather close to what Donald Knuth coined literate programming: the simultaneous generation of the output of both text as analysis.

Third, and finally, think very carefully about backup programs. You know, you never need it until you really need it. The first advise is to always use an external disk with a backup application (such as Time Machine on OSX). Secondly, having some of your applications (especially your code, and thus text, files) somewhere in the cloud has additional benefits. Apart from a backup advantage, it also facilitates working together on a project. The best known of these applications is Dropbox, although cooperation using this is still a bit cumbersome. Another, more nerdy, example is using Git, which is a full blown distributed revision control and source code management system, created by Linus Torwards (that guy from Linux). It takes some time to get use to, but in the end it grows on you and becomes not only easier to use, but makes you in the end even more efficient.

As a last example about complementarity, the before mentioned Rstudio application does not only edit L<sup>A</sup>T<sub>E</sub>X and R, but is able as well to set up a Git repository system. In the end, it comes all together.

## 6. Some final thoughts

This might all seem a bit daunting. The best advise for novices would be: start one step at a time (“A journey of a thousand miles begins with a single step”, Lao-tzu). In this case this simply means that you start with a template file (such as that from the Elsevier article class), incorporate or copy in some text, divide it by section and subsection headers and see whether it works.

Granted, the learning curves of most of these tools are rather steep (except for that of using citation managers—you really should use them). However, it does pay-off in the end. And most of these tools have the fortunate feature to be very *complementary* to each other. A good example is making slides. There is a package out there called beamer, which allows you to create slides from your text. This basically means, that all your codes used for your paper can be used again for your slides (within a L<sup>A</sup>T<sub>E</sub>X context, so equations won’t fly all over the slide as sometimes happens with more conventional presentation programs). If you know one application really well, it becomes easier to understand others or even use the same commands. And there are quite a few applications or languages out there that work

with text files and are very handy for a researcher—HTML, Markdown, Latex, scripting languages from Stata and R and even languages as Ruby and Python come to mind. The latter, for example, is the native language of ArcGis scripts.

Obviously, this paper is way too short to do justice to them. However, starting with L<sup>A</sup>T<sub>E</sub>X would be a good introduction to a more customated and automated workflow.

## References

- Ghen, M., Gheasi, M., Nijkamp, P., Poot, J., 2012. The impact of immigration on international trade: A meta-analysis. In: Nijkamp, P., Poot, J., Sahin, M. (Eds.), *Migration Impact Assessment: New Horizons*. Edward Elgar, Cheltenham, Ch. 9, pp. 301–317.
- Healy, K., 2013. *Choosing Your Work Flow Applications*. Tech. rep., Duke University.