Contents

Introduction

1.1 How it's different from Word

1.1.1 WYSIWYG - What You See Is What You Get

You may be familiar with the challenge that is using Word to write equations even as simple as those in Figure 1.1. The formatted text on the page is exactly everything you have, which often means spending time trying to fix how it looks and what goes where, a process we call typesetting. So **what is the alternative?**

The Laplace transform of a function $f(t)$, defined for all real numbers $t \geq 0$, is the function $F(s)$, which is a unilateral transform defined by:

$$F(s) = \int_0^\infty f(t)e^{-st}dt$$

Figure 1.1: Word, an example of What You See Is What You Get

1.1.2 Typesetting

Let me introduce you to LaTeX, pronounced either *lah-tec* or *lay-tec*.

– **stuff here**.

The main motivators for why you would want to use it are:

1. Beautifully written documents. No more trying to deal with the various fonts, margins, spacing, etc.;
2. Easy bibliography management, citations and cross-references;
3. Easy equations and graphs. Even extremely complicated mathematical concepts can be easily written.

Let's go back to our example, and see how it would be done with LaTeX:

Listing 1.1: Example document written with LaTeX

```
\documentclass[12pt,a4paper]{article}
\usepackage{amsmath}
\begin{document}
The Laplace transform of a function $f(t)$, defined for all real numbers $t \geq 0$ is the function $F(s)$, which is a unilateral transform defined by:
\begin{equation*}
    F(s) = \int_0^\infty f(t)e^{-st} dt
\end{equation*}
\end{document}
```

If you are familiar with programming, this may vaguely look like a *markup language*, and it is! If you are not, don't worry. We will introduce the most distinctive features, and go into more detail as necessary.

1.1.3 Markup language

There are generally three types of markups:

1. **Environments** introduce some kind of formatting, such as lists, math mode or more complicated things. A backslash, `\`, is used to indicate a markup, curly brackets `{}` indicate an *argument*, and square brackets `[]` (optionally) provide options.

   With very few exceptions, environments have the following format:

   ```
   \begin{environment-name}[options]
       ...
   \end{environment-name}
   ```

   One of the exceptions is in Listing 1.1. Can you spot it? It is `$...$`, the *in-line math mode* environment. We will be exploring it in more detail later.

2. **Instructions** define some feature of the document. This ranges from breaking a page to defining the headings you have available. In our example you can see:

   `\documentclass[...]{article}`

3. **Variables** can either be predefined or user defined, and these range from greek letters to the integral sign to a whole expression. Some examples seen are `\geq` (**g**reater or **eq**ual) and `\int` ($\int$). Intuitively, `\alpha` results in α, and similarly for other greek letters.

   Note: There are some characters with predefined meaning, such as `{`, `}` and `&`. To use the literal curly brackets, ampersand, etc we would need to *escape* them. Conveniently, this is done with a backslash (\), so feel free to think of it as a variable: `\{`, `\}` and `\&`.

1.1.4 Packages

You may have noticed that `\usepackage{amsmath}` was not mentioned as an instruction the previous section. That's because packages are worth mentioning on their own.

Packages add features to our document, similar to *import* in most programming languages. `amsmath` gives us a wide array of maths tools, but there are packages for drawing, graphing, colouring, better management of bibliography, easier organisation of your files... and the list goes on.

1.1.5 Compilation

We can use any text editing software to write and save our documents in `.tex` files. In order to produce a `.pdf`, it needs to be *compiled*. As a beginner you may spend a lot of time scratching your head, wondering why it's not compiling.

The good news is that recommended IDEs (**I**ntegrated **D**evelopment **E**nvironments, basically text editors filled with features) handle the compilation for you and have features to help you spot mistakes.

Installation

2.1 Installing LaTeX

2.2 Installing an IDE

Getting started

3.1 Importing graphs from Matlab

3.1.1 Importing bitmap graph

The simplest method to import get your Octave/Matlab graphs onto a document is to export them to a bitmap picture, and import like we did before.

Our code looks something like the following[1], and the full example is in `Examples/Importing-graphs`.

Listing 3.1: `Importing-graphs/bitmap/example.m`

```
x=linspace(0,2*pi);
y=sin(x);
plot(x,y);
xlabel('x'); ylabel('f(x)');
legend('f(x)=sin(x)', 'Location', 'best');
print('figures/example.eps','-depsc')
```

Then, to include the graph, we simply use `\includegraphics`.

```
\begin{figure}[h]
    \centering
    \includegraphics[width=\textwidth]{figures/example.eps}
\end{figure}
```

Below you can see the output compared to something natively drawn with pgfplots. It is a deliberately simple graph that can be easily replicated natively, but it highlights that font sizes and other small issues can become problematic for legibility.

So what can we do instead? We can generate the graphs in Octave/Matlab and export the data for `tikz` and `pgfplots` to natively render it. We have two options to do so: `matlab2tikz` and exporting to a `.svg` file.

[1] You may notice the figure is exported to `.eps`, not `.png`, `.tiff` or something otherwise more common. For that you may want to look into `export_fig`.

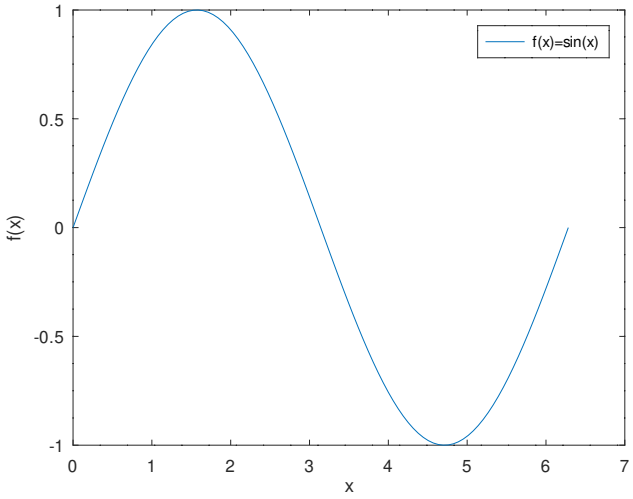

Figure 3.1: Matlab

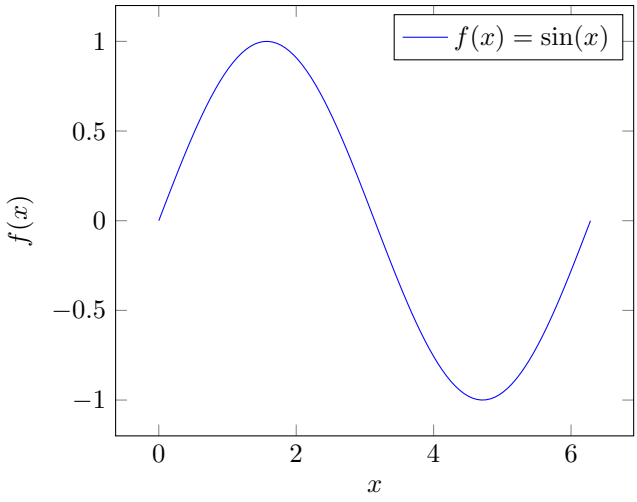

Figure 3.2: Natively rendered

Challenge: Replicate the sinusoidal graph with `pgfplots`, and you might see a straight line, indicating that it uses degrees, not radians. Try to search for the solution! Hopefully you will run into this link.

3.1.2 Matlab2tikz

This is an outstanding script written by Nico Schlömer which can be downloaded from Mathworks' file exchange, or directly from the github. Please refer to the installation on the github page, if you are unfamiliar with installing external matlab functions. In short, use Set Path to include the downloaded `src/` to your path.

Usage is very similar to exporting a bitmap image, the only difference is that we get a `.tex` to import, which can be further edited if you really want to:

Listing 3.2: `Importing-graphs/matlab2tikz/mat2tikz.m`

```
x=linspace(0,2*pi);
y=sin(x);
plot(x,y);
xlabel('x'); ylabel('f(x)');
legend('f(x)=sin(x)', 'Location', 'best');
matlab2tikz('figures/graph.tex', 'height', '\fheight', 'width', '\fwidth');
```

Importing requires a few packages and settings, as can be seen below. The information is extracted from `matlab2tikz`'s github, and reading their FAQ may be very helpful

The code is not separated into preamble and content, but it should be clear what goes where. Importantly, the whole example can be found in `Examples/Importing-graphs`.

Listing 3.3: `Importing-graphs/matlab2tikz/totikz.tex`

```
\documentclass{article}
\usepackage{pgfplots}
\pgfplotsset{compat=newest}
\usetikzlibrary{plotmarks}
\usetikzlibrary{arrows.meta}
\usepgfplotslibrary{patchplots}
\usepackage{amsmath}
\pgfplotsset{plot coordinates/math parser=false}
\begin{document}
\begin{figure}[h] \centering
    \newlength\fheight{} \newlength\fwidth{}
    \setlength\fheight{8cm}
    \setlength\fwidth{12cm}
    \input{figures/graph.tex}
\end{figure}
\end{document}
```

Resulting in:

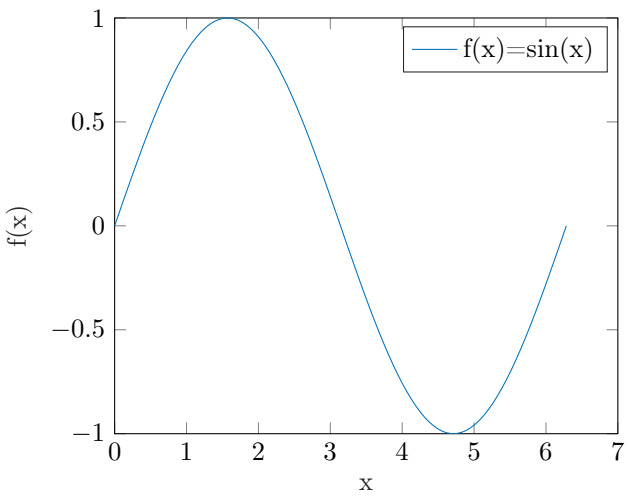

Figure 3.3: Matlab generated and rendered natively

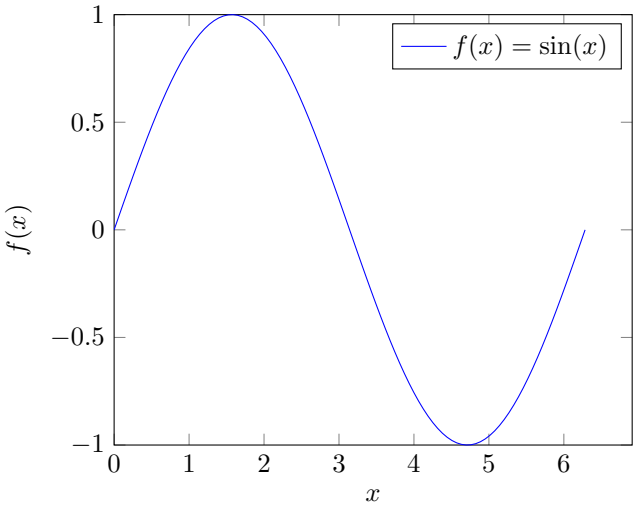

Figure 3.4: Generated with pgfplots

Do you know why the fonts look still different? In the graph generated by matlab, we did not tell it to render using math mode — f(x) vs $f(x)$.

3.1.3 Importing vector graph

How do I...?

4.1 Search engine

4.2 Stack Exchange

4.3 CTAN package information

Working faster

5.1 Becoming familiar with the IDE

5.2 Using snippets