Making Publication Quality Figures

Sigvald Marholm

02.03.20

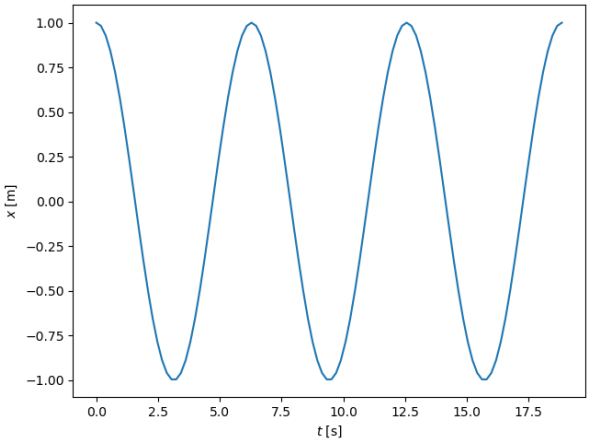
1 Introduction

It is not uncommon to see technical reports and scientific articles with figures of different style than the rest of the document. The text in the figures may be too small or too large, the wrong font may be used, the lines in plots so thin they are barely visible, or figures may appear pixelated when printed due to insufficient resolution. Figure 1a is an example of such a figure. Although it is quite understandable that such figures appear, there is a very simple recipe for avoiding them[1]:

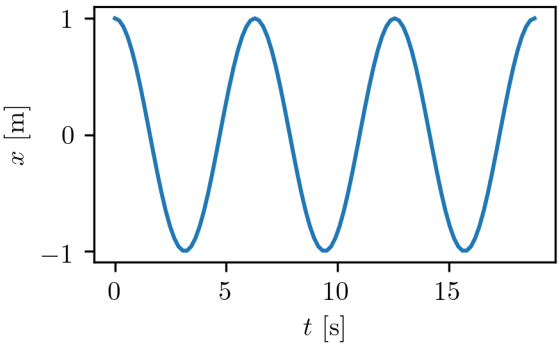
1. Obtain the correct figure size and font from the document where the figure is to be inserted.
2. Create the figure using the right sizes, font and resolution from the start. For consistency between figures you may use the same line width, color palettes, etc. in all your figures.
3. Import the image into the document *without* rescaling it.

The only difference between fig. 1a and fig. 1b is that the latter was made according to the above recipe.

In this document I shall explain how to carry out the above steps in practice, as well as some tricks for an efficient workflow. This being about publication quality, I assume LaTeX is used for typesetting, but demonstrate how to best incorporate figures from a selection of high quality graphics tools such as Matplotlib and Inkscape. That said, this is not an exhaustive tutorial on those tools, or on the artistic aspects of figure making.



(a) Poor figure



(b) Good figure

Figure 1: Example figures

[1] Admittedly, there are other ways too, for instance making figures directly in LaTeX using PGF or Tikz, but this recipe has become my preferred way. Note that vector graphics is not a replacement for the recipe. It gives you an easy way out when it comes to resolution, but text sizes and line widths gets scaled unless care is taken.

2 The Recipe

2.1 Step 1: Extract font and sizes

2.1.1 Figure Size

We shall start by determining the right size of the figure. For two-column articles one may for instance let the figure fill the entire column width, whereas for single column articles perhaps a lower fraction of the width is more suitable. In any case, it is a good start to know what the column width *is*.

The column width is not the same on all documents, but fortunately, it is rather easy to extract by inserting the following LaTeX command somewhere in your document:

```
\the\columnwidth
```

When the document is compiled, the width of the column will be written in place of this command, for instance "221.0pt" for a typical two-column document. In case you have a very wide figure and would like it to span both columns (using the `figure*` environment), you should instead insert

```
\the\textwidth
```

The text width accounts for both columns, as well as the spacing in between. For single column documents the text width and the column width are usually the same.

Although many programs allow you to specify the size of a figure directly in typographic points (pt), one should be aware that slightly different definitions of the pt exists. This usually isn't noticable when specifying the text size, which is usually around 10pt. However, if a figure ends up slightly wider than a column, you will get an "overfull hbox" warning by LaTeX. You may therefore wish to convert the column width to for instance millimeters or inches before creating the figure. In LaTeX, the conversion is given by

$$1\,\mathrm{pt} = 0.351\,\mathrm{mm} = 0.0138\,\mathrm{in}. \qquad (1)$$

A column width of for instance 221 pt thus converts to 77.6 mm. If you want a full width figure, it pays off to round downwards, for instance to 77 mm, to make sure you don't get an "overfull hbox". The height of the figure is simply chosen to get an aspect ratio deemed appropriate for the figure. I sometimes take the width multiplied or divided by 1.618 to get the golden ratio.

2.1.2 Font and Text Size

The next thing we need is to determine the font family and text size. This can be found by inserting the following into your document[2]:

```
\makeatletter
\f@family~\f@size pt
\makeatother
```

To make it a bit more convenient, the following code can be put in the preamble of your document:

```
\makeatletter
\newcommand{\getfont}{\f@family~\f@size pt}
\makeatother
```

Now you can just insert `\getfont` where you want to determine the font. Note that the text size often differs in foonotes, figure captions and in the body of the document. A good result can be obtained by using a text size in the figures that is identical to that in the figure caption. You may of course choose differently, as long as you do it consistently.

The output from the above command may be something like "cmr 10pt". cmr is short for Computer Modern Roman, which is the standard font in LaTeX. Unfortunately the font format used in LaTeX is a different kind than the TrueType or OpenType formats understood by most programs, and it may therefore be difficult to find the exact same font for use in your graphics software. Computer Modern, however, often have good replica available for download on the internet. If you are unable to find the exact same font, you may instead use something similar. Computer Modern or variants of Times or Times New Roman are often used fonts in scientific journals.

One thing in particular to pay attention to is whether or not the fonts have *serifs*, i.e., tiny feet

[2] The macro `\f@family` contains the font name and the macro `\f@size` the text size. The `@` symbol is frequently used in macro names internal to LaTeX packages to prevent them from being overwritten by the end user. The reason this works is because `@` behaves as a normal letter inside packages and that it can therefore be used in macro names. During normal usage, however, `@` is a special symbol. Nevertheless, we can temporarily turn `@` into a normal letter with `\makeatletter` to get access to these macros, and then turn it back to a symbol again using `\makeatother`. This is what we have done in the above code.

on letters such as “i” and “f”:

Fonts may be classified into serif fonts or sans (without) serif fonts according to this feature. It is interesting that the serifs almost touches one another at the base of each line, thereby creating the illusion of a continuous line on which the letters sit. There is also the illusion of a line on top of the “low” letters. These lines are believed to help guide the eye along the text, preventing you from losing track of which line you’re currently reading. Serif fonts are therefore the norm when it comes to printed books and articles, perhaps except for headings, or where there are only small fragments of text. Sans serif fonts, on the other hand, can be used with lower resolution, and are predominant on web pages, and other digital resources. Plotting tools such as Matplotlib often uses a sans serif font by default. Choosing *any* serif font in your plots goes a long way for a more consistent look. Compare for instance fig. 1a to fig. 1b. The former uses the default sans serif font in Matplotlib, whereas the latter uses Computer Modern, which is also used in the rest of this document.

2.2 Step 2: Create the Figure

Regardless of what software you use for creating your figures, you ought to use a suitable image format, and choose an appropriate resolution. If you fail at this, the picture may be grainy or show compression artefacts on print, or the file size of the document may become very large. Many journals have rather strict size limits on the submitted files. In this section I elaborate on which file formats to use and resolution requirements while postponing details about specific software for making images until later.

2.2.1 Image Formats

BMP This is one of the simplest formats, where an image is represented as a grid of pixels (a *raster*) with each pixel stored as a fixed number of bytes. This leads to comparatively large files. Use PNG or JPG instead.

PNG Similarly to BMP, this is also a raster format, but contrary to BMP, it uses a *lossless* compression scheme, meaning that it takes less space with no loss of quality. Save the files with compression level 9 to get the smallest files. The compression used in PNG is efficient for images where large areas have the same color, typically graphical illustrations, schematics and plots. The compression is not as efficient for photographs, although it does lead to significantly smaller files than BMP.

JPG Yet another raster format, but this one with a *lossy* compression, leading to a quality reduction. For JPG you can tune a quality factor between 0 and 100, where a higher number means better quality, but larger files. JPG files are very well suited for photographs, and even a quality factor of 90 or 95 will produce files much smaller than PNG files at still an excellent quality. The JPG compression do not handle sharp edges very well, where “foggy” artifacts may become visible, especially for lower quality factors. This makes it in particular unsuitable for graphical illustrations, schematics and plots, where there are large single-color regions with sharp edges between.

SVG Contrary to the previous formats, SVG is a *vector* format, meaning that it consits of geometric entities such as lines and curves being described by their end points, curvature, thickness, etc. Whereas the pixels would become visible if you scale up a raster format, the vector formats can be scaled up indefinitely without getting pixelated. For this reason, vector graphics is excellent for graphical illustrations and schematics, but the figures must be created using a dedicated program, such as InkScape, from the very start. You cannot convert a raster image into a vector format. Although it is sometimes possible to store plots in vector formats, this means storing every single line segment in the plot, which can potentially be quite many, and this causes needlessly large files. For that reason I have come to prefer PNG files for plots of real data. Unfortunately, LaTeX cannot import SVG files, so they must be converted to PDF files prior to being used in LaTeX. However, SVG files is the native format for many programs, and keeping

a copy of the figure also in SVG enables you to more simply make adjustments later on.

PDF In addition to full fledged documents, PDF files can also be used to store vector graphics, as described above.

EPS If you compile your documents using the command `latex`, none of the above formats can be used, and you must instead resort to EPS. EPS can also store raster and vector graphics, but I recommend rather compiling your documents with the more modern `pdflatex` if possible.

Although vector formats can be scaled without loss of quality, it is still desirable to create figures with the right dimensions to begin with, as determined in step 1. This means that if you insert text with a size of, say, 10pt in your graphics program, it will appear as 10pt in LaTeX because you do not have to scale the figure. You can also standardize dimensions such as line thicknesses and arrow sizes to your liking to get a consistent look throughout your entire document. When using raster formats, it is in addition necessary to have a sufficiently good resolution.

2.2.2 Resolution

Whereas a resolution of 100 PPI (pixels per inch) looks good on a computer screen, it is necessary with 300 PPI on printed paper. On the other hand, you may want to avoid much better resolution than 300 PPI to avoid excessive file sizes.

2.3 Step 3: Inserting the Figure

Finally comes the point of inserting the figure into your LaTeX document, which is done in the simplest way possible:

```
\includegraphics{file.png}
```

(given that the file is named `file.png`). Note that we do not use any options for scaling the figure, like `[width=0.7\columnwidth]` or `[scale=0.23]`. The default is already a scaling factor of one, and the figure should already have the right size.

2.4 InkScape

2.5 Gimp

2.6 Matplotlib

2.7 ImageMagick

That's all, good luck!