MathPSfrag: LaTeX labels in MATHEMATICA plots

Johannes Große
Institute of Physics, Jagiellonian University, Reymonta 4, 30-059 Kraków, Poland
`jgrosse (at) mppmu dot mpg dot de`
`http://wwwth.mppmu.mpg.de/members/jgrosse`

Abstract

A MATHEMATICA® package that allows inclusion of LaTeX labels in EPS graphics using PSfrag will be presented. The clue is that positioning information and TeX code is automatically generated by the package. It also contains a preview capability that imports a bitmap of the final image including the rendered LaTeX labels back into MATHEMATICA.

1 Introduction

MATHEMATICA (Wolfram, 1999; Wolfram Research, Inc., 2005) is one of the major commercial computer algebra systems and as such used in many fields of scientific research.

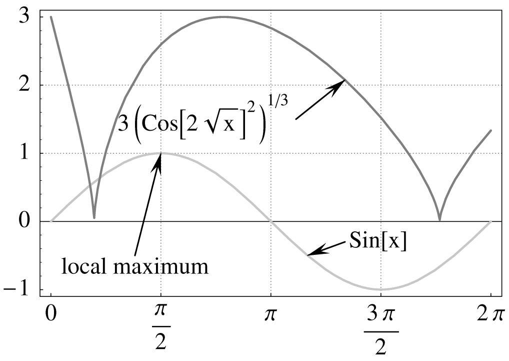
Unfortunately, labels in graphics produced by MATHEMATICA — like those of most other graphics programs — are not visually compatible with TeX's standard fonts. Even though MATHEMATICA provides advanced typesetting capabilities for complex mathematical expressions that are close to a faithful representation of standard mathematical notation, it cannot compete with TeX in this regard.

MathPSfrag (Große, 2005) is intended to fill this gap, but it is also meant to address another problem: Many authors consider the layout of the manuscript as something to be safely left to the computer. While TeX does a remarkable job in providing excellent typesetting with little user intervention, the same cannot be said about image preparation.

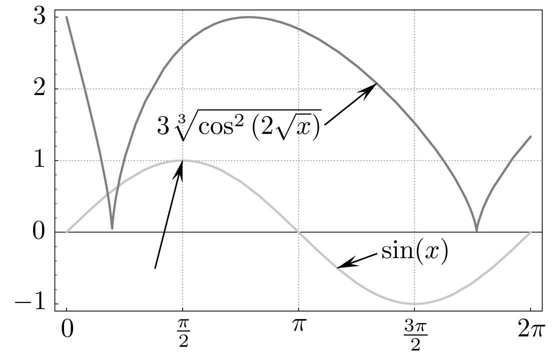
Solutions to the common task of attaching labels to plots range from sophisticated (McKay and Moore, 1999) to crude: conversion of the exported EPS file to JPEG, editing in a graphics program, back conversion to EPS for inclusion in LaTeX.

From the latter example it is clear that any solution addressing this problem needs to work not only from a typesetting point of view but also from a daily user's perspective. MathPSfrag is an attempt to combine the existing technique of PSfrag (Grant and Carlisle, 1998) with a transparent, easy to use convenience layer.

The PSfrag package provides TeX macros that allow replacement of text strings ("tags") in EPS graphics. For PSfrag to work these tags have to be output unaltered using a single PostScript `show` directive. Since MATHEMATICA splits complicated expressions into several `show` commands, simple alphanumeric sequences have to be used as tags, which makes the resulting raw EPS file rather illegible. Bookkeeping of automatically generated tags was the only feature provided by the very first version of MathPSfrag, although several more sophisticated features have been added since.



(a) Conventional MATHEMATICA without MathPSfrag



(b) The same plot *after* automatically substituting all `Text` primitives (including tick mark labels) by LaTeX output.

Figure 1: Old vs. new graphics export

Ideally MathPSfrag does not require any user

```
f1[x_] := Sin[x]; f2[x_] := 3*((Cos[2 Sqrt[x]])^2)^(1/3);

rawplot = Plot[{f1[x], f2[x]}, {x, 0, 2 Pi}, PlotStyle→{Hue[1.0], Hue[0.6]}, Frame→True,
    FrameTicks→{Pi/2*{0, 1, 2, 3, 4}, Automatic, None, None}];

Needs["Graphics`Arrow`"];
SimpleLabel[tip : {_, _}, txt_, txtpos : {_, _}, align : {_, _}] :=  Sequence[
    Arrow[txtpos, tip, HeadScaling→Absolute, HeadLength→8, HeadCenter→0.6],
    Text[txt, txtpos, align]];
textlabels = Graphics[{
    SimpleLabel[{Pi/2, f1[Pi/2]}, "local maximum", {1, -0.5}, {0, 1}],
    SimpleLabel[{7/6Pi, f1[7/6Pi]}, f1[x], {4.2, -0.3}, {-1, 0}],
    SimpleLabel[{4.2, f2[4.2]}, f2[x], {3.5, 1.5}, {1, 0}]
    }];
mygrid = Map[{#, {AbsoluteDashing[{0.1, 1}], GrayLevel[0.5]}} &, {Pi*{1/2, 1, 3/2}, {1, 2}}, {2}];
exampleplot = Show[rawplot, textlabels, GridLines→mygrid];
```

Figure 2: Full Mathematica code for the plot in Figure 1(a).

intervention except for calling a different graphics export command from within Mathematica.

MathPSfrag will take over the task of inserting tags into the EPS in place of the original labels and will also use Mathematica's `TeXForm` command to determine the LaTeX macros reproducing a pretty-printed version of the original Mathematica expression. These macros are written to a separate TeX file. In a second step, a LaTeX, `dvips`, Ghostscript sequence is carried out to merge the two files to a single EPS that is independent of PSfrag and will be called "unpsfraged" in the following. As such it can be (and by default is) converted into a PDF image suitable for inclusion in pdfLaTeX documents. Moreover a bitmap image is rendered and imported back into Mathematica as a preview.

In reality there are a number of problems that can arise — the simplest would be Mathematica producing undesired or flawed TeX code, such that the above rendering sequence would fail. Since the process of image creation described in this article involves many programs and production steps, there is actually quite a lot of potential for malfunction. In the first section of this tutorial we will nevertheless assume that this does not happen and that MathPSfrag has already been set up correctly to find LaTeX, `dvips` and Ghostscript. In subsequent sections, we will discuss these points in more detail. For a full presentation of all options and extended examples, the reader is referred to the manual accompanying MathPSfrag.

We would like to point out that we denote three different objects *psfrag*: the LaTeX package PSfrag, which provides the LaTeX macro `\psfrag`, and its Mathematica counterpart `PSfrag`.

2 A first example

For concreteness we will start by defining a conventional Mathematica plot without any reference to MathPSfrag. We will try to make it as beautiful as possible for a fair comparison with MathPSfrag. The code given in Figure 2 performs the following actions to draw Figure 1(a).

The first line defines the functions to be plotted, which happens in the second line. This already gives a decent plot, but to show off MathPSfrag's capabilities a few more elements are inserted into the plot. The third block of commands loads a standard Mathematica package and defines the function `SimpleLabel`, which draws an arrow and attaches a textual expression to the arrow. It is then used to define the three text labels seen in the plot. (By "textual expression" we denote all expressions that at some point end up as the argument of a `Text` primitive, in other words the expressions we want to replace by LaTeX commands eventually.) As a finishing touch, a grid of light gray lines is created. The last line merges all those elements into the final plots in Figure 1.

An EPS image can be produced by a simple `Export[exampleplot,"exampleplot.eps"]`. However, by default Mathematica unfortunately uses Courier as a font for the labels, and does not allow inclusion of fonts into the EPS image (for Mathematica versions before 4.2.1); as a result, any symbols, such as large brackets, that require Mathematica's special fonts can only be processed when the TeX distribution has been set up to find these fonts (WRI Support, 2007). For later Mathematica versions we should rather export the plot by:

```
Export[Show[exampleplot,
          TextStyle→{FontFamily→"Times"}],
       "pure-mma.eps", ConversionOptions→
                 {"IncludeSpecialFonts"→True}]
```

which sets the default font to Times Roman.

The corresponding export commands provided by MathPSfrag read

```
Needs["MathPSfrag‘"];
PSfragExport[exampleplot, "filename"]//UnPSfrag;
```

which loads the MathPSfrag package and creates files `filename-psfrag.tex` and `filename-psfrag.eps` containing the PSfrag versions of the plot.

The `UnPSfrag` command takes these files and creates an unpsfraged PDF and EPS version, which in turn is rendered into a bitmap, imported into MATHEMATICA and displayed as a preview. When the user wants to use only the PSfrag versions of the images, the `UnPSfrag` command can be omitted. The package is accompanied by a shell script for merging the two files into an unpsfraged EPS file. This may be useful when there is no LaTeX distribution available on the machine where MATHEMATICA resides.

In general, it is recommended *not to rescale* unpsfraged images within the LaTeX source of the final manuscript, because such a rescaling would also change the size of the rendered LaTeX expressions. While on modern TeX installations chances are good to end up with the scalable outline versions of Computer Modern in the EPS files because MathPSfrag by default invokes `dvips` with the `-Ppdf` switch, the overall visual consistency of the final manuscript will suffer from labels of different sizes. Hence it is recommended to set the size of the image at rendering time by providing suitable options to the `\includegraphics` command that is internally invoked by `UnPSfrag`. This can be achieved by

```
UnPSfrag[PSfragExport[exampleplot, "filename"],
  IncludeGraphicsOptions→"width=75mm"];
```

which will preserve the labels, while scaling the image *approximately* to the given size. The reason for the mismatch of size is that the bounding box of the final image correctly fits its *contents* while the size provided by the user refers to the bounding box of the original image, which changes during the PSfrag process.

When special symbols or different fonts are required for the graphics, `UnPSfrag` can be instructed to include a user-defined preamble by means of the `TeXPreamble` option.

3 Providing custom LaTeX commands

MathPSfrag generates LaTeX commands by employing MATHEMATICA's `TeXForm`, whose output may not always be what the author expected. In particular, the output of `TeXForm` depends very much on the version of MATHEMATICA— versions 5.1 or later are most suitable, though a compatibility TeX package has been implemented; see 'Known limitations'.

For overriding MathPSfrag's default treatment of single textual elements of the plot, the `PSfrag` command is provided. It can be simply wrapped around the argument of a `Text` primitive or a plot option that eventually produces a `Text` primitive. Most frequently used examples for the latter case are the `PlotLabel` and `AxesLabel` options. So instead of `PlotLabel→"chi-square test"`, the following could be used:

```
PlotLabel→PSfrag["chi-square-test",
                  TeXCommand→"$\\chi^2$-test"]
```

This would still display as "`chi-square test`" in MATHEMATICA, but appear as "χ^2-test" in the final manuscript. Note that a doubling of any backslash in the argument of the `TeXCommand` options is required because MATHEMATICA considers the backslash character to be an escape symbol.

Since MATHEMATICA provides the means for entering formatted expressions as part of ordinary text strings, the above example is somewhat artificial. The same effect could have been achieved by simply using `PlotLabel→"`χ^2`-test"` and relying on MathPSfrag (or to be more precise `TeXForm`) to produce the corresponding TeX representation.

A more realistic example would be changing one of the labels in Figure 1(b) by replacing

$$3\sqrt[3]{\cos^2(2\sqrt{x})} \qquad \text{by} \qquad 3\left|\cos\sqrt{4x}\right|^{\frac{2}{3}}.$$

This can be achieved by substituting the argument of the corresponding `SimpleLabel` line in Figure 2.

```
tex="$3\\left|\\cos\\sqrt{4x}\\right|^
...                             \\frac{2}{3}$";
SimpleLabel[{4.2, f2[4.2]},
  PSfrag[f2[x],TeXCommand→tex],
  {3.5, 1.5}, {1, 0}]
```

The additional command has been written in italics; the resulting plot is shown in Figure 3.

PSfrag can also be used to pass alignment information, (angular) orientation or a scale factor to MathPSfrag. The respective options (`TeXPosition`, `PSPosition`, `PSRotation`, `PSScaling`) are in a one-to-one correspondence with the options of the command `\psfrag` provided by the LaTeX package. In

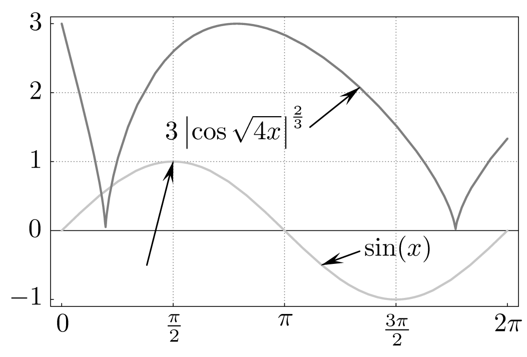


Figure 3: Manual replacement of the "cos ..." label.

particular, the first two options accept strings built from two characters (**t**op, **c**enter, **B**aseline, **b**ottom, plus **l**eft, **r**ight, **c**enter) describing the vertical and horizontal position of an anchor point in the tag and LaTeX box. When replacing the tag by the LaTeX box, the new anchor point is glued to the position of the old one. Rotation is in degrees; the use of the scale factor is discouraged and provided only for completeness' sake.

Unless three-dimensional plots are used none of the above should be necessary. As a last resort, when there is a positioning bug, one may use the `TeXShift→{"x","y"}` option. MathPSfrag shifts the content of the corresponding expression by the amount specified in the two strings, which should contain valid TeX dimensions.

4 Setup

As mentioned in the introduction, we left out some crucial points. The most important of these is that MathPSfrag needs information about the actual location of the LaTeX, `dvips` and Ghostscript executables unless they can be found in the system's execution path. (Specifically, these binaries are needed by `UnPSfrag` whereas the other parts of MathPSfrag will continue to work without them. This is the main reason for having a separate `UnPSfrag` command at all.) While this is usually the case for Unix-like operating systems, it is rather the exception for Windows-based systems. Moreover, MathPSfrag by default uses the typical system-specific names of the executables, which might differ on some installations.

The user can either fix the system settings or provide the absolute path to the executables by setting the appropriate `UnPSfrag` options as outlined in Figure 4(a). The configuration may be checked by executing `MathPSfragConfigurationTest`, which will output diagnostic information. Step-by-step instructions guiding through the configuration are provided in a separate MATHEMATICA notebook ("`MathPSfrag-Test.nb`"), which also generates a number of examples.

In order to avoid the necessity of setting the correct paths each time MathPSfrag is loaded, the configuration can also be stored in an `init.m` file, which is automatically loaded by MATHEMATICA during start-up. Valid locations of the `init.m` file depend both on the operating system and on the MATHEMATICA version, but are documented in MATHEMATICA's Help Browser. A typical example for such a file is given in Figure 4(b).

```
SetOptions[UnPSfrag,
  LaTeXExecutable→"C:\\path-to\\latex.exe",
  DvipsExecutable→"C:\\path-to\\dvips.exe",
  GhostscriptExecutable→
              "C:\\path-to\\gswin32c.exe"];
```

(a) Setting the locations of external programs

```
AppendTo[$Path, "C:\\path-to\\MathPSfrag"];
$PostMathPSfrag := SetOptions[UnPSfrag,...];
```

(b) Minimal `init.m` file

Figure 4: Configuration of MathPSfrag

5 In the manuscript

Unpsfraged graphics can be treated in the usual manner and included by the `\includegraphics` command, where it is good practice to leave out the file's suffix to allow LaTeX or pdfLaTeX to load the appropriate format. For best quality it is recommended to avoid usage of the `width` or `height` options, but instead to set the size of the plot during creation from within MATHEMATICA as described in 'A first example'.

PSfrag-based graphics are generically included as follows:

```
\usepackage{psfrag,graphicx}
...
\begin{psfrags}
\input{exampleplot-psfrag}
\includegraphics[width=75mm,
          trim=-10 -20 -30 -40]{exampleplot-psfrag}
\end{psfrags}
```

where any numbers of course have to be adapted. The `trim` option enlarges the bounding box when negative arguments are used, which is sometimes required to avoid clipping problems. Hence one should always check the bounding box by enclosing each `\includegraphics` macro by an `\fbox`.

6 Known limitations

As mentioned in the introduction for the sake of presentation we have ignored all potential problems so far. It is however vital to point out that image production with MathPSfrag employs several programs which come in different versions and installations on different computer systems with all the associated compatibility problems. The work flow also involves a number of user decisions, which have a strong impact on the final image. For the discussion of these choices it is convenient to think of the process of manuscript creation in terms of the following stages.

1. Plot preparation with MATHEMATICA
2. Export with MathPSfrag
3. Manuscript preparation (and inclusion of images)
4. Output format generation (PS or PDF)

We will discuss these stages in reverse order. Let us assume that the final output format should be PDF. There are currently two possibilities to achieve this. Either following the traditional path of translating the manuscript with LaTeX, `dvips` and a distillation to PDF, e.g., by `ps2pdf`; or using pdfLaTeX with its enhanced typesetting capabilities. Since this choice is not necessarily under the author's control, it may be wise to keep both paths open.

When using unpsfraged images, this amounts to simply invoking `\includegraphics` without providing the filename's suffix and placing both the EPS and PDF version of the image where TeX can find them. When using PSfrag-based images, due to their PostScript-centric nature, additional effort is necessary to make these work with pdfLaTeX. Fortunately, because of the popularity of the pstricks package there are several packages that provide methods for incorporating PostScript into pdfLaTeX documents, namely pdftricks, pst-pdf and ps4pdf. While they differ considerably regarding ease-of-use and limitations of the respective implementation, all of them generate PDF versions of PostScript related images essentially by extracting them from a conventional LaTeX run. An example file for each of those packages accompanies MathPSfrag.

Before deciding whether to employ PSfrag-based or unpsfraged images, one should keep in mind that unpsfraged images are hard to edit: They neither allow rescaling without changing the size of the labels nor is there an easy method of changing the contents of the labels. It is therefore advisable to design the MATHEMATICA notebook generating the plots in such a way that replotting can be achieved without recalculating. In other words the result of a time-consuming calculation should be stored separately before plotting. Moreover the author should know in advance which fonts will be used for the final manuscript. These can be loaded by setting the `TeXPreamble` option of `UnPSfrag` to a suitable `\usepackage` command.

For PSfrag-based images changing a label only requires editing the corresponding `\psfrag` macro in the TeX file associated to the image, whereas rescaling of the image will preserve the size of the labels. However, PSfrag-based images always have a wrong bounding box, which can potentially lead to clipping errors and should therefore be manually corrected by use of the `trim` option for `\includegraphics`. Wrapping an `\fbox` around the image while doing so considerably facilitates this task.

The limitations of MathPSfrag we now discuss are mainly due to its dependence on three MATHEMATICA functions: `TeXForm`, `FullGraphics` and `AbsoluteOptions`.

The output of `TeXForm` consists of a MATHEMATICA-specific set of LaTeX commands for versions earlier than 5.1, whereas starting from 5.1 `amsmath` macros are used. While still having deficiencies regarding symbols that do not have a direct LaTeX counterpart, the latter is most suitable for use with MathPSfrag. The output of earlier MATHEMATICA versions, on the other hand, except for very basic expressions, will require a compatibility layer, which is part of the MathPSfrag distribution. It does however need to be installed where LaTeX can find it when called by `UnPSfrag` and should also accompany the manuscript when PSfrag-based images are used. Alternatively it is of course possible to manually provide LaTeX macros with the `PSfrag` command as described in 'Providing custom LaTeX commands'.

Both `FullGraphics` and `AbsoluteOptions` convert (certain aspects of) MATHEMATICA graphics from a logical to a physical representation in terms of so-called primitives. MathPSfrag needs this physical representation for the extraction of alignment information of all text elements of a plot as well as for content generation in the case of tick marks. Unfortunately, neither command is faithful; i.e., they do not preserve the visual appearance of the plot. MathPSfrag has been carefully implemented to work around these shortcomings, but fails at one minor point: Floating point numbers very close to integers (difference $< 10^{-10}$) will be rounded.

While MathPSfrag should be able to correctly position and align any text elements, the respective options of `PSfrag` can also be used as a quick and dirty solution to any misplacements. In particular the `TeXShift` option is provided solely for this purpose since it is not used during default processing

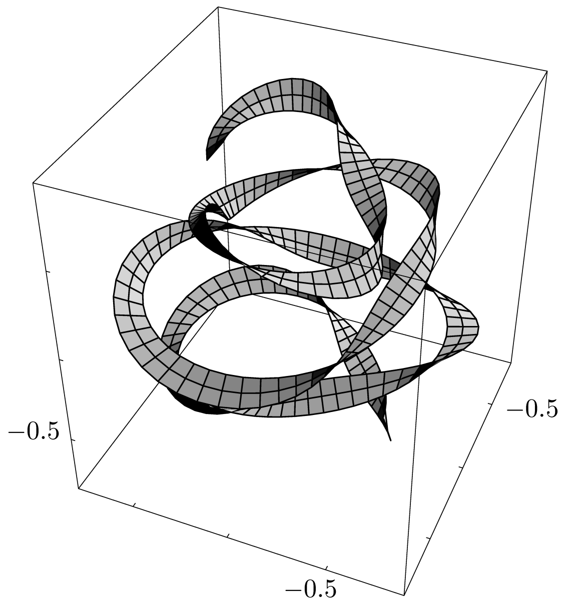


Figure 5: Three dimensional example: As there exists no `FullGraphics3D` command, manual labeling was required.

at all. In the case of any such misplacements, the author would appreciate a bug report.

Moreover there is no `FullGraphics` command for three-dimensional plots. As a consequence, the required alignment information has to be provided by hand for every text element of a plot. This sounds more tedious than it actually is in most cases — example code is provided as part of MathPSfrag. Here it shall suffice to show the result; cf. Figure 5.

Finally, MathPSfrag does not provide methods to construct correct tick mark *content* as it is strictly focused on shape. It does however integrate well with the CustomTicks package (Caprio, 2005), which provides that functionality.

7 Conclusion

MathPSfrag provides a convenient interface to PSfrag permitting the generation of high-quality labels in Mathematica graphics. While it automates all tedious aspects of PSfrag, it still allows the user to seamlessly override all of its internal assumptions. The possibility to create images that do not depend on PSfrag anymore provides a simple method for achieving pdfLaTeX compatibility.

MathPSfrag has been tested with Mathematica versions 4.1, 5.0 and 5.2 under Linux and Windows XP. A previous version has also been tested under Mac OS X.

For the future it would be interesting to incorporate the PSfragx extension that allows including the `\psfrag` commands into the comment lines of the EPS file. It would also be interesting to provide means for the generation of `\overpic` commands, thus bypassing many of the shortcomings of the PSfrag approach. For Mathematica, because of its closed source nature, this is not a simple task because the position information of graphics primitives in terms of absolute coordinates for the final image is not easily accessible.

8 Acknowledgments

The author acknowledges support by ENRAGE (European Network on Random Geometry), a Marie Curie Research Training Network in the European Community's Sixth Framework Programme, network contract MRTN-CT-2004-005616. Part of the work on MathPSfrag were supported by the DFG Graduiertenkolleg "The Standard Model of Particle Physics — structure, precision tests and extensions" at Humboldt-Universität zu Berlin and the Max-Planck-Institut für Physik, München.

References

Caprio, Mark. "Custom tick marks for linear, logarithmic, and general nonlinear axes". `http://library.wolfram.com/infocenter/MathSource/5599/`, 2005.

Grant, Micheal C., and D. Carlisle. "The PSfrag system, version 3". Available from CTAN, `macros/latex/contrib/psfrag`, 1998.

Große, Johannes. "MathPSfrag". `http://wwwth.mppmu.mpg.de/members/jgrosse/mathpsfrag`, 2005.

McKay, Wendy, and R. Moore. "Convenient Labelling of Graphics, the WARMreader Way". *TUGboat* **20**(3), 262–271, 1999. `http://www.tug.org/TUGboat/Articles/tb20-3/tb64ross.pdf`.

Wolfram, Stephen. *The Mathematica book*. Cambridge University Press, New York, NY, USA, 4th edition, 1999.

Wolfram Research, Inc. "Mathematica 5.2". 2005. `http://www.wolfram.com/`.

WRI Support. "Mathematica Fonts in EPS files and Ghostscript". Available from `http://support.wolfram.com/mathematica/graphics/export/`; `ghostscript.html`, `includefonts.html`, 2007.