`matlabfrag` user guide

Zebb Prime

May 4, 2010

Contents

1 Introduction

`matlabfrag` is a function which exports a MATLAB figure to .EPS and .TEX files for use in LaTeX with `psfrag`. It is inspired by LaPrint, but is intended to be more WYSIWYG, by respecting figure handles better. The main reasons to use `matlabfrag` are the same as those for using `psfrag`: figure font matching that of the document, and the ability to have complex mathematical expressions typeset by LaTeX.

2 Background

I wrote `matlabfrag` after becoming frustrated with the default LaPrint behaviour, including it putting the line to insert the graphic in the output .TEX file: `\includegraphics{FileName.eps}`

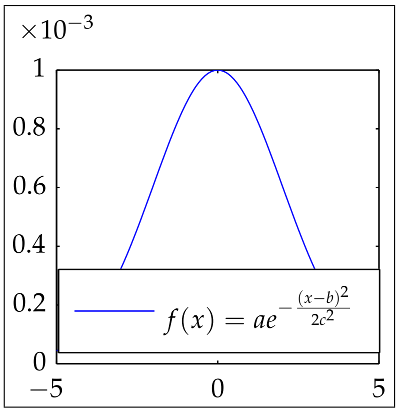
Whilst these problems could be addressed using one of the many options in LaPrint, I decided to take a stand against the many MATLAB scripts and functions available which try to impose their own sense of style on my figures, and instead write a function which does everything possible to respect the figure handles. Everyone has their own sense of style, for better or worse, and I'm sure most users have their own little scripts set up to format their figures in their own way.

The problem I have with the `\includegraphics{FileName.eps}` line in the .TEX file is that I keep my figures in a `graphics` subdirectory of my main document. This meant I had to manually edit the .TEX file every time I exported the figure from MATLAB.

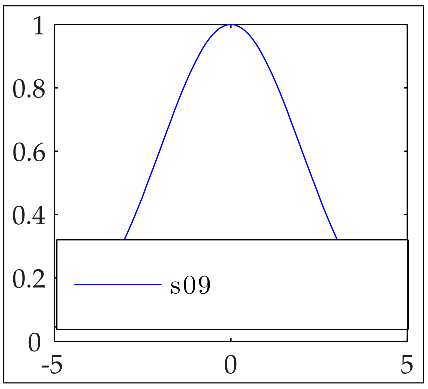
3 Compared to LaPrint

In this section I compare the output from `LaPrint` and `matlabfrag` for an identical figure. I have chosen some examples that shows some of the weaknesses of `LaPrint`, so be sure to take these comparisons with a grain of salt.

The code for the first comparison is given in
`examples/comparison01.m`
and the results are shown in Figure 1. Notice that the legend box for the LaPrint output has been shrunk, and is too small for the equation, and the y axis scale has disappeared. I am unwilling to blame `LaPrint` for the equation not rendering as it may by a bug in any stage of the fairly complicated conversion process (which has nothing to do with `matlabfrag`). The output .TEX and .EPS files appear to have the text conversion set up properly.



(a) graphics/comparison01-matlabfrag

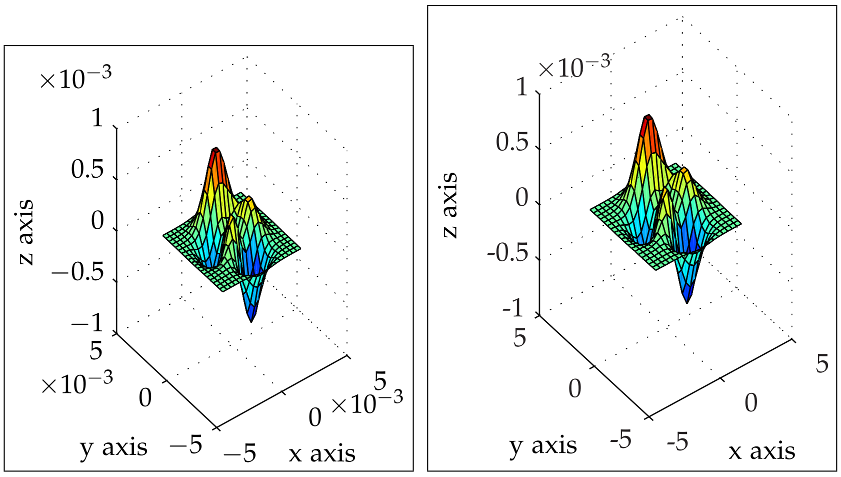


(b) graphics/comparison01-laprint

Figure 1: `matlabfrag` output versus `LaPrint` output for a simple graph. Notice the legend box for the `LaPrint` is incorrectly sized, and the y axis scale has disappeared. I am unsure why the equation is not rendering in the `LaPrint` version, and it is unlikely to be the fault of `LaPrint`.

In this second comparison a scaled version of the MATLAB `peaks` function is presented. The MATLAB code for this example is given in:
`examples/comparison02.m`
and the results are shown in Figure 2. The axis labels are handled better by `matlabfrag` and it also manages to reproduce the x and y axis scales, but I stress that they only work well for the default orientation of a `surf` plot. If you start rotating the image they are unlikely to come out correctly. This is

discussed further in Section 9.



(a) graphics/comparison02-matlabfrag (b) graphics/comparison02-laprint

Figure 2: `matlabfrag` output versus `LaPrint`.

4 Usage

4.1 Within Matlab

Using `matlabfrag` within Matlab is easy. Simply format the figure how you like it, then run `matlabfrag`. The format for the `matlabfrag` command is:

```
matlabfrag(FileName,OPTIONS)
```

where `FileName` (required) is the file name for the output .tex and .eps files, and `OPTIONS` are key-value pairs for the optional options:

`'handle'` The handle for the figure to export. The default is `gcf` ('get current figure').

`'epspad'` The number of points to pad the output .eps file by. The default is `[0,0,0,0]`.

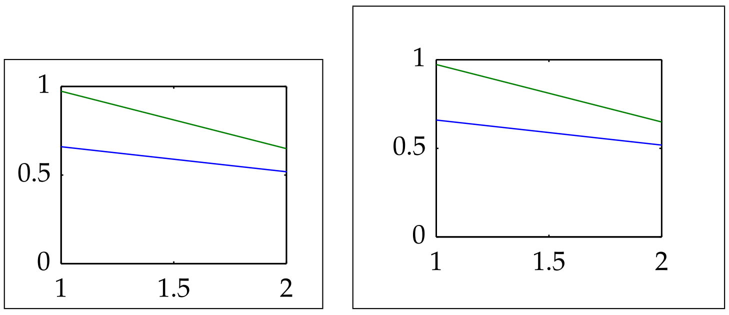
`'renderer'` The output renderer. The default is `painters`. The renderer is discussed in more detail in Section 8.

'dpi' The output resolution (DPI or more appropriately, PPI) of the figure. For the painters renderer this defaults to 3200, and when using the opengl or zbuffer renderers the default is 300. A discussion of renderers is given in Section 8.

Some examples are given below.

```
hfig = figure(gcf);
plot(rand(2));
axis([1 2 0 1]);
matlabfrag('graphics/ex01-1');
matlabfrag('graphics/ex01-2','epspad',[10,10,10,10],...
  'handle',hfig);
```

See Figure 3 for the output.



(a) graphics/ex01-1 (b) graphics/ex01-2

Figure 3: `matlabfrag` options example showing the `epspad` option at work.

If you wish to show something different in MATLAB and in the .TEX document, then you can add it into the `UserData` property of the text, with a `matlabfrag:` prefix. This is especially useful if you have macros in LaTeX which you want to use, or if you want to use some commands not included in the MATLAB LaTeX installation. In this example, the `\LaTeX` macro typesets LaTeX as LaTeX.

```
plot(rand(2));
axis([1 2 0 1]);
xlabel('should not see this text','UserData',...
  'matlabfrag:Plays nice with \LaTeX');
matlabfrag('graphics/ex02');
```

See Figure 4 for the output.

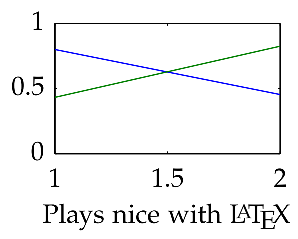


Figure 4: UserData example; X-label should say 'Plays nice with LaTeX'.

4.2 Within LaTeX

The .TEX and .EPS files can be included in a LaTeX document (pdfLaTeX is treated seperately below) by either loading the psfrag package and including the .TEX file before the .EPS file:

```
\documentclass{article}
\usepackage{graphicx,psfrag}
\begin{document}
  \input{FileName.tex}
  \includegraphics{FileName.eps}
\end{document}
```

or my preferred method; loading the pstool package (v1.2 onwards), and using the \psfragfig macro:

```
\documentclass{article}
\usepackage{pstool}
\begin{document}
  \psfragfig{FileName}
\end{document}
```

Notice the lack of a file extension in the \psfragfig macro. This is a requirement of pstool. pstool also loads the graphicx, psfrag and color packages (as well as a few others), so it is not necessary to explicitly load these packages.

For more information about pstool or psfrag please see their respective manuals.

The text replacement performed by psfrag occurs somewhere between the conversion from DVI to PS in some seemingly magical process that I don't understand. This means that if you look at the DVI output directly you will see a `PSFrag replacements` table next to the figure. This is normal, so do not panic.

4.3 Within pdfLaTeX

There are several ways to include files with PostScript commands (such as `psfrag`) in pdfLaTeX such as `pst-pdf`, `auto-pst-pdf`, `ps4pdf` and `pstool`, by far the most useful of which is `pstool`. The document is set up the same way as the LaTeX example above:

```
\documentclass{article}
\usepackage{pstool}
\begin{document}
  \psfragfig{FileName}
\end{document}
```

The only difference being that `pstool` runs some external processes to run the postscript commands, and as such it needs the `-shell-escape` command when being called:

```
   pdflatex -shell-escape example.tex
```

where `example.tex` is the code above.

4.4 Within LyX

Using `matlabfrag` withing LyX is relatively simple. The steps to setting LyX up with `pstool` are:

1. Ensure `pstool` is installed in your LaTeX distribution. If you wish to use `pstool` with non-PDF output, please ensure `pstool` is v1.2 or greater.
2. Load `pstool` in the document preamble by going to: `Document - Settings`, then choose `LaTeX Preamble` in the navigation panel in the left. Insert the line `\usepackage{pstool}` and apply the settings.
3. Enable `-shell-escape` in pdfLaTeX by going to: `Tools - Preferences`, then choose `Converters` in the navigation panel in the left. Under `Converter Definitions` find the `LaTeX (pdflatex) -> PDF (pdflatex)` entry. Below this change the line

   ```
   pdflatex $$i
   ```

   to

```
pdflatex -shell-escape $$i
```

then click `Modify` to the upper right, then `Apply` and `Save` down the bottom.

Please note that these steps were written for LyX version 1.5.6, but they should be similar for other versions.

Once `pstool` is set up for use in LyX, `matlabfrag` images can be inserted by clicking on the TEX button (Insert Tex Code), and placing the `psfragfig` command in:

```
\psfragfig{FileName}
```

The `ERT` box created with the `psfragfig` command inside it should be placed as you would place a normal figure, for example inside a `float:Figure`.

Please also read the discussion at the end of Section 4.2 regarding the DVI output. Naturally the same applies for the DVI output from LyX as well.

5 Creating figures in MATLAB

I am now going to suggest a few ways to manipulate figures in MATLAB before exporting them using `matlabfrag`.

5.1 Sizing

I suggest that the first thing you do when creating a figure is setting it to the size that you want in the final document. This way the figure does not need to be resized at any stage, which prevents line sizes changing, text over-running borders, etc. This can be done within MATLAB using:

```
hfig = figure;
plot([-1 1],rand(2));
axis([-1 1 0 1]);
set(hfig,'units','centimeters',...
  'NumberTitle','off','Name','ex03');
pos = get(hfig,'position');
set(hfig,'position',[pos(1:2),8,3]);
```

See Figure 5 for the output from this example, `examples/ex03.m`.

5.2 `Color`

The `color` property is a three element vector representing the RGB colour of the text. `matlabfrag` will respect the colour set in MATLAB, but may

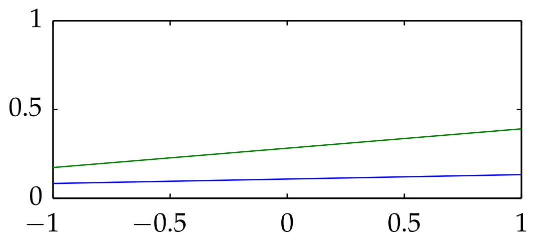


Figure 5: Resizing example; the figure is sized to 8 cm by 3 cm.

require that you load the `color` package in LaTeX (`pstool` implicitly loads the `color` package).

```
plot(rand(2));
axis([1 2 0 1]);
xlabel('Red text!','color',[1 0 0]);
ht = text(1,0.5,'Green text!');
set(ht,'color',[0 1 0]);
```

See Figure 6 for the output from this example, `examples/ex04.m`.

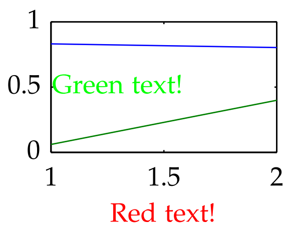


Figure 6: `Color` example; the text should be coloured.

5.3 `FontSize`

All text in MATLAB figures has a `FontSize` property which `matlabfrag` respects; the size specified in the figure is the size it will be in the LaTeX output document.

```
plot(rand(2));
axis([1 2 0 1]);
hx = xlabel('12pt font');
set(hx,'FontSize',12);
text(1,0.5,'8pt font','FontSize',8);
```

See Figure 7 for the output from this example, `examples/ex05.m`.

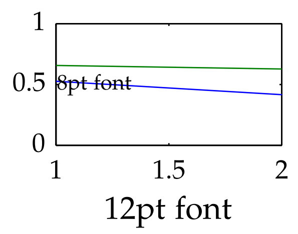


Figure 7: `FontSize` example; the text should be different sizes.

5.4 `FontAngle`

`FontAngle` is a property shared by all text in a MATLAB figure. `matlabfrag` respects `FontAngle` in that `italic` and `oblique` will be italic in the LaTeX output document.

```
plot(rand(2));
axis([1 2 0 1]);
xlabel('Italic font','FontAngle','italic');
ht = text(1,0.5,'Oblique font');
set(ht,'FontAngle','oblique');
```

See Figure 8 for the output from this example, `examples/ex06.m`.

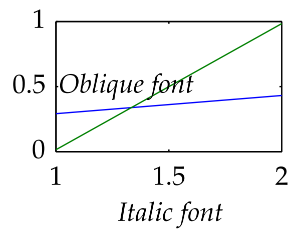


Figure 8: `FontAngle` example; Oblique and Italic font should both be italic here.

5.5 `FontWeight`

Another text property from MATLAB figures that `matlabfrag` respects is `FontWeight`, such that any text set to `bold` will be made bold in the output LaTeX document. `light`, `demi` and `normal` have no effect on the font in the LaTeX output.

```
plot(rand(2));
axis([1 2 0 1]);
hx=xlabel('Bold font');
```

```
set(hx,'FontWeight','bold');
text(1,0.5,'Demi font','FontWeight','demi');
title('Light font','FontWeight','light');
```

See Figure 9 for the output from this example, `examples/ex07.m`.

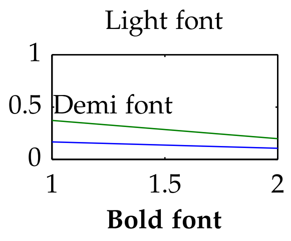


Figure 9: `FontWeight` example; bold should be bold, demi and light do not really translate to LaTeX, so they should default to normal.

5.6 `FontName`

`FontName` is a property that `matlabfrag` *does not* respect, with one exception. One of the basic ideas behind using `matlabfrag` and `psfrag` is to match the font in figures to that of the text, so it does not make sense to use this property, except when it has been set to `FixedWidth`, in which case the font in the output LaTeX document will be fixed-width (i.e., `\ttfamily`).

```
plot(rand(2));
axis([1 2 0 1]);
xlabel('Comic sans?','FontName','Comic Sans MS');
ht = text(1,0.5,'Fixed-width');
set(ht,'FontName','FixedWidth');
```

See Figure 10 for the output from this example, `examples/ex08.m`.

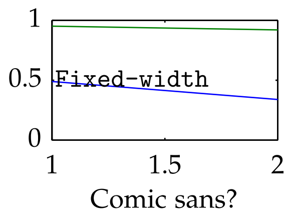


Figure 10: `FontName` example; fixed-width should be in a typewriter font, while Comic Sans should be changed to the font of the document.

5.7 `Interpreter`

MATLAB has three (well, two really) text interpreters that is uses to render the text. These are `tex`, `latex` and `none`. I generally don't recommend the use of the default `tex` interpreter if rendering anything mathematical and using `matlabfrag`, as while the `tex` interpreter may render a mathematical expression fine, it may not work in LaTeX.

The `latex` interpreter is especially useful for rendering mathematical formula.

```
plot(rand(2));
axis([1 2 0 1]);
xlabel('$x=\frac{\alpha}{\beta}$','interpreter','none');
text(1.5,0.5,'$x=\frac{\alpha}{\beta}$',...
  'interpreter','latex');
```

See Figure 11 for the output from this example, `examples/ex09.m`.

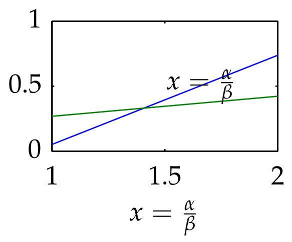


Figure 11: `Interpreter` example; both equations should be the same, despite using a different interpreter in MATLAB.

5.8 Multi-line text

Multi-line text can be entered into MATLAB using either a cell or a two-dimensional character matrix. If using a character matrix, `matlabfrag` preserves all of the white spaces, so if you do not wish for this to occur, use cells instead (or check out the `cellstr` function).

`matlabfrag` recreates multi-line text using a `tabular` environment. If you also use the `UserData` property, you will need to manually put the `tabular` environment in, as `matlabfrag` will ignore any `UserData` that is not a string.

```
plot(rand(2));
axis([1 2 0 1]);
title(['2d matrix           ';'     weird alignment']);
```

```
xlabel({'cells';'are';'better!'});
text(2,0.5,'multiline in LaTeX','HorizontalAlignment',...
  'right','UserData',['matlabfrag:\begin{tabular}',...
  '{@{}r@{}}multiline\\in\\\LaTeX\end{tabular}']);
```

See Figure 12 for the output from this example, `examples/ex10.m`.

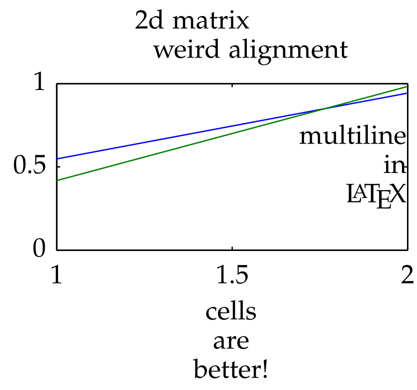


Figure 12: Multi-line example; three different ways to make multi-line text. 2D arrays, cells and using `tabular`.

5.9 Custom tick labels

Custom tick labels can be added directly as strings for LaTeX to interpret. In this example, the `\ytick` macro has been defined in the preamble of this document as:

```
\newcommand\ytick[1]{\ensuremath{\mathcal R(e^{#1 i})}}
```

The MATLAB example code is:

```
plot([1 exp(1)],rand(2));
hax = gca;
set(hax,'xlim',[1 exp(1)],'xtick',[1 exp(0.5) exp(1)],...
  'xticklabel',{'$e^0$','$e^{0.5}$','$e^1$'});
set(hax,'ylim',[0 1],'ytick',[0 real(exp(-i*7*pi/4)) 1],...
  'yticklabel',{'\ytick{3\pi/2}','\ytick{7\pi/4}','\ytick{2\pi}'});
```

See Figure 13 for the output from this example, `examples/ex11.m`.

5.10 Legends

Legends can, with a bit of manipulation, be used in exactly the same way as other text objects.

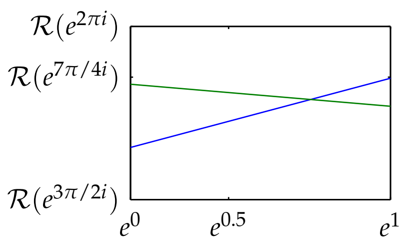


Figure 13: Custom tick label example.

If LaTeX strings are entered into the legend, the LaTeX interpreter can be turned on using the legend handle directly:

```
hleg = legend($\ddot\alpha$,$\ddot\beta$);
set(hleg,'interpreter','latex');
```

To use the `matlabfrag` `userdata` auto-substitution feature with legend texts, the text objects inside the legend need to be accessed directly. Each legend entry appears as three child objects of the legend (`line`, `marker` and `text` in that order), and the entries are in reverse order to that which they were entered into `legend` as. Knowing this order, the `userdata` property of the text objects can be set:

```
plot([-1 1],rand(3,2));
hleg = legend('L = 1 m','L = 2 m','L = 3 m');
hlegc = get(hleg,'children');
set(hlegc(9),'userdata','matlabfrag:$L=\SI{1}{m}$');
set(hlegc(6),'userdata','matlabfrag:$L=\SI{2}{m}$');
```

If each legend string is unique, then another method is find the text object is to use the `findobj` function in MATLAB.

```
set( findobj(hlegc,'string','L = 3 m'), 'userdata',...
  'matlabfrag:$L=\SI{3}{m}$');
```

The output from this example, `examples/ex12.m`, is given in Figure 14.

5.11 Tick scaling values

Tick values automatically generated by MATLAB may have a scaling value placed at the end of the axis. In two-dimensional plots these will be handled by `matlabfrag` fairly well, but in some circumstances, such as three-dimensional plots, they will work badly. An example of them being automatically placed in the figure is given in Figure 2 and the file

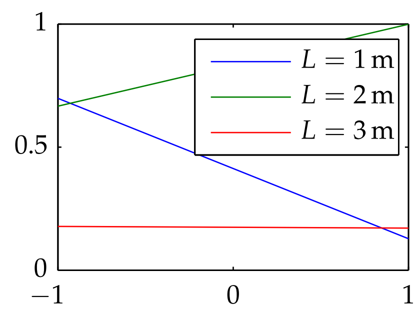


Figure 14: Using the `userdata` feature with legends.

`examples/comparison02.m`

When `matlabfrag` fails to insert the scaling properly, I suggest you manually modify the axis ticks, and place the scaling in either the axis label, or place it manually as a text object. Example code for doing this is:

```
[x,y,z] = peaks(20);
surf(x.*1e-3,y.*1e-3,z);
% placed inside the label, need to set xticklabelmode to manual
xlabel('x axis ($x10^{-3}$)','interpreter','latex',...
  'userdata','matlabfrag:x axis $\left(\times10^{-3}\right)$');
set(gca,'xticklabelmode','manual');
% manually placed as a text object:
ylabel('ylabel');
set(gca,'yticklabelmode','manual');
text(-10e-3,0,-10,'$x10^{-3}$','interpreter','latex',...
  'userdata','matlabfrag:$\times10^{-3}$');
```

See Figure 15 for the output from this example, `examples/ex13.m`. You can

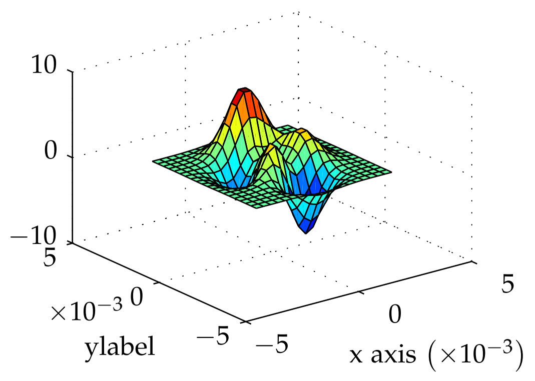


Figure 15: Manually inserted axis scalings.

read more about why this is necessary in Section 9.

6 Frequently (occasionally) asked questions

6.1 Why does the output have three digit numbers all through it?

Either you are looking at the .EPS output and that is how it is supposed to look, or there was a problem processing the image with `psfrag`. Check out the troubleshooting section below.

6.2 Why doesn't the .TEX file have the `includegraphics` statement in it like LaPrint does?

Firstly, read the Background section above. With all the image files stored in a `graphics` subdirectory of the main document, I would have to manually open the .TEX file and insert the path into the that line. This was one of the reasons I wrote `matlabfrag` in the first place.

I have considered including an option to produce this output for those who would rather use this behaviour than load `pstool` but decided against it due to the directory problems. From v1.2 onwards `pstool` works fine in LaTeX so there really is no need to manually put the `\includegraphics` line in.

7 Troubleshooting

7.1 I get an error in MATLAB!

The only errors that should be generated from `matlabfrag` are from using it with an old version of MATLAB or calling it with invalid parameters. Unfortunately the error messages generated from the `inputParser` class are a bit cryptic, but you should be able to work out if it is a parameter error. A simple way to check this is to use an example that is known to work.

If using a version of MATLAB older than v7, you will receive a parse error, rather than a sane error message telling you your version of matlab is too old. An example of this error is:

```
"identifier" expected, "(" found.
```

As this is a parse error, it occurs before any code in the script has run, hence I am unable to output a sane error message. The only way to output a sane error message in such an old version of Matlab would be to rewrite `matlabfrag` so that it doesn't use anonymous functions, which I don't want to do.

No other errors should be generated by `matlabfrag`. If you encounter one it is most likely a bug. Please email the smallest self-contained example that reproduces the bug to me at
`zebb.prime+matlabfrag@gmail.com`

7.2 I get a warning in Matlab!

Warnings created by `matlabfrag` are there for a reason. Read through them, as they explain what the problem is and what `matlabfrag` is doing about it.

7.3 I get the `Unable to interpret TeX string` warning in Matlab!

This warning occurs when the TeX or LaTeX interpreter (whichever is chosen) cannot interpret the string, and can be caused by calling macros which don't exist in the version of LaTeX that ships with Matlab or by trying to interpret a LaTeX string with the TeX interpreter (and vice versa).

To troubleshoot, ensure the string is correct, and the appropriate interpreter is used. An example known to work is:

```
title('Force versus $\sin(\alpha)$','interpreter','latex');
```

The `none` interpreter can also be used, however the box around the text may end up a very different size to the final text. The following example also works, but the bounding box in Matlab is much wider than in the final document:

```
title('Force versus $\sin(\alpha)$','interpreter','none');
```

Another alternative is to use the `userdata` feature of `matlabfrag` with a placeholder for the screen text. A working example is:

```
title('Force versus sin(a)','userdata',...
  'matlabfrag:Force versus $\sin(\alpha)$');
```

This is also a good option for using macros that aren't included with the version of LaTeX that ships with MATLAB. For example, in Section 5.10, the `\SI` macro is used to format the numbers and units. The package that provides this command, `sistyle`, is loaded in the preamble of this document.

As this warning has to do with the way MATLAB renders text on screen and not `matlabfrag`, please do not contact me about it.

7.4 My graphic exports from MATLAB fine, but does not compile in LaTeX/pdfLaTeX/LyX!

1. Check that all LaTeX statements in your figure are valid. If you are having problems, try a simple example that is known to work.
2. If you have manually loaded the `graphicx` package, make sure you don't have a driver (e.g. `pdftex`) specified.
3. If using pdfLaTeX and `pstool`, check:
   (a) `-shell-escape` option is set when calling pdfLaTeX.
4. If using LaTeX and `pstool`, check:
   (a) `pstool` is at least at version 1.2. An alternative location to finding `pstool` is `http://github.com/wspr/pstool/`.
5. If using LaTeX or pdfLaTeX with anything else:
   (a) Install `pstool`!
6. If using LyX:
   (a) Check to see if the `pstool` package is loading by going to `Document - LaTeX log`, and trying to find the `pstool` entry. If it is not found then try repeating the instructions in Section 4.4. If it is there, check to see if it is issuing a warning that `-shell-escape` is disabled. If so, follow the instructions in Section 4.4.

7.5 My graphic exports from MATLAB and compiles in LaTeX/pdfLaTeX/LyX but doesn't look right!

1. If you use a `includegraphics` option such as `rotate`, there is a bug in `pstool` prior to v1.2b that caused the option to be applied twice.

2. The text is running over lines or is cut out!
    (a) This most likely happens if you resize the image in LaTeX using a `[width=x]` argument. It is usually better to size the image in Matlab in the first place, and do no further resizing.
    (b) If you have mathematical equations, use the `latex` interpreter. This will make the text object be roughly the right size, which will let Matlab place it properly. This is especially useful for legends, as shown in Figure 1a.
    (c) If the text appears to be cropped from the outside of the graphic then try padding the .eps file using the `epspad` option, or try the `pdfcrop` option for `pstool`:
    `\usepackage[crop=pdfcrop]{pstool}`
3. A complicated graphic appears different on screen than it does in the document.
    (a) This is most likely an issue caused by a renderer. For more information, read the discussion on renderers in Section 8.

8 Advanced usage: a discussion on renderers

Matlab has three different renderers that it uses to render a graphic on screen; `Painters`, `OpenGL`, and `Z-Buffer`. Of these, only the `Painters` renderer can export an image in a native text format which is needed for the text substitution process. If one of the other renderers is used the text becomes rasterised, and hence normally no text replacement can take place.

As of `matlabfrag v0.6.0`[1], an `EXPORTFIG` and `epscombine` style hack has been included, which manually places text objects inside rasterised .eps images. The result of this is that text can now be substituted with LaTeX font and code, but also that all the text now appears in the foreground (i.e. all text appears in front of the image, regardless of its placement in the original image).

A downside to placing a rasterised image inside an .eps file is that it is uncompressed, which creates a very large .eps file. However, during the conversion to `pdf` the image is compressed to a reasonable file size. If you are using `pstool`, from `v1.3` it defaults to a lossless compression technique similar to png. This is the most suitable compression for technical graphics,

[1]There have been a series of bug fixes since `v0.6.0`, so I always recommend using the latest version.

but should you prefer it, lossy compression can be turned on. For more information, refer to the `pstool` documentation.

When printing a rasterised image, the resolution is set using the `dpi` option in `matlabfrag`. The default is 300 for `OpenGL` and `Z-Buffer` rendered images, which is usually considered to be print quality. The default for the `Painters` renderer is 3200. Using a large `dpi` value for `Painters` rendered images reduces quantisation error while only increasing the file size by a small amount.

By default `matlabfrag` will use the `Painters` renderer unless either the figure has had the renderer manually set (which causes the `renderermode` property to become `manual`), or the `renderer` option is set in `matlabfrag`.

An example of choosing the renderer in `matlabfrag`, and increasing the image DPI is given in
`examples/ex14.m`
an extract of which is:

```
peaks;
hs = get(gca,'children');
set(hs,'facealpha',0.4,'edgealpha',0.4);
hl=legend('legend');
set(hl,'location','northeast');
xlabel('X','userdata','matlabfrag:$\mathrm X$');
ylabel('Y','userdata','matlabfrag:$\mathbf Y$');
zlabel('Z','fontsize',12,'userdata','matlabfrag:$\mathcal Z$')
matlabfrag('graphics/ex14','renderer','opengl','dpi',720);
```

The output for this example is given in Figure 16.

An example showing `matlabfrag` detecting a manually set renderer is given in
`examples/ex15.m`
which is largely based off of the "Moving the Camera Through a Scene" example in the MATLAB documentation. An extract of this example is:

```
lighting phong;
set(gcf,'Renderer','zbuffer');
text(0,0,0,'smiley','userdata','matlabfrag:\blacksmiley');
drawnow;
matlabfrag('graphics/ex15');
```

The output from this example is given in Figure 17.

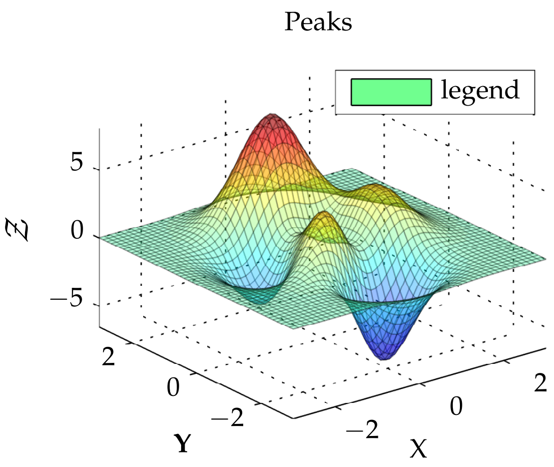


Figure 16: Example showing the use of the `OpenGL` renderer.

9 Advanced usage: a discussion on axis ticks

Easily the hardest part about writing `matlabfrag` has been dealing with axis ticks. Their positioning and circumstances when the labels are modified are completely hidden from the user. For example, if the axis is logarithmic, and the ticks occur at powers of 10, then the `xticklabel` properties are written as the power of ten (e.g. 2) and rendered as the 10 to the power of the number (e.g. 10^2). `matlabfrag` handles this case by detecting whether `xscale` is set to `log`, and `xticklabelmode` is set to `auto`. If so, it automatically inserts the appropriate LaTeX code. A simple example of this behaviour can be shown with the code:

```
s = tf('s');
bodemag(1/(s/100+1));
```

from `examples/ex16.m`, which produces Figure 18.

The above case is fairly simple in comparison to how MATLAB handles axes that have been scaled by a power of ten. For example, if $x \in [0 \quad 10^{-3}]$, then the x ticks would read, say $[0 \quad 0.5 \quad 1]$, and a $\times 10^{-3}$ would be placed to the lower right of the x axis. Unfortunately, this $\times 10^{-3}$ text object is completely hidden from the user, and as soon as the tick labels are modified by the user it disappears. Both `matlabfrag` and `LaPrint` have to modify the tick labels to make it possible for `psfrag` to do text substitution, thus we have to manually recreate this scaling. In `matlabfrag` this is performed by detecting when:

1. `xticklabelmode` is set to `auto`, and

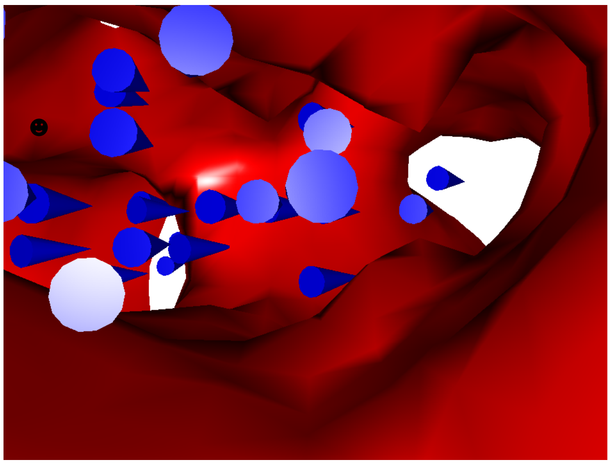


Figure 17: Example output when the renderer is set to `zbuffer` and shading set to `phong`.

2. there is a scaling value between the `xtick` and `xticklabel` that is equal to a power of ten.

In these circumstances, `matlabfrag` does its best to place the scaling value in the most appropriate location. For two-dimensional graphs this isn't too bad, but for three-dimensional plots it is very difficult.

The locations for all of the scaling values for all permutations of x and y axis locations in a two dimensional plot are shown in Figure 19. The code for this example is given in:

`examples/ex17.m`.

Lastly, support for tick scaling in three-dimensional plots has been added to `matlabfrag` for the default orientation of `surf` plots, such as that shown previously in Figure 2. If you start to rotate the orientation of the plot it will probably break horribly. In these circumstances I suggest you manually place the scaling in the axis label, or as a text object. Please read through Section 5.11 for some information and an example of how to do this.

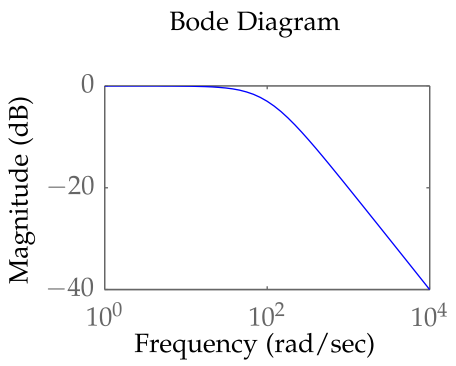


Figure 18: An example `bodemag` plot demonstrating ticks of the format 10^x. Note that the axis tick labels are grey. This is due to the `bodemag` function setting them grey using the `xcolor` and `ycolor` axis properties.

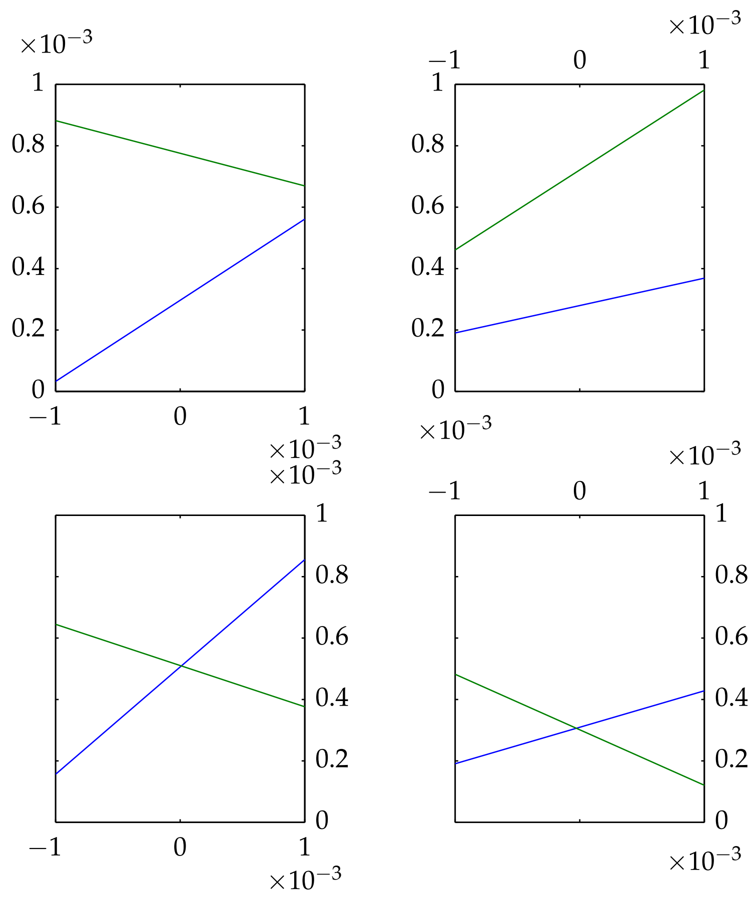


Figure 19: Axis scale positions for all permutations of the x and y axis locations.