The **pgfSweave** Package

Cameron Bracken and Charlie Sharpsteen

April 3, 2011

http://cran.r-project.org/web/packages/pgfSweave/index.html

Contents

1 Introduction

The **pgfSweave** package is about speed and style. For speed, the package provides capabilities for "caching" graphics generated with **Sweave** on top of the caching funcitonality of **cacheSweave**[1]. For style the **pgfSweave** package facilitates the integration of R graphics with LaTeX reports through the **tikzDevice**[2] package. With these tools, figure labels are converted to LaTeX strings so they match the style of the document and the full range of LaTeX math symbols/equations are available. In addition **pgfSweave** can produce syntax highlighted and/or cleaned up source code.

The backbone of **pgfSweave** is a a new driver for **Sweave** (`pgfSweaveDriver`). The driver provides new chunk options `tikz`, `pgf` and `external`, `sanitize`, `highlight` and `tidy` on top of the `cache` option provided by **cacheSweave**. This package started as a fork of **cacheSweave**. This document highlights the features and usage of **pgfSweave**. This document assumes familiarity with **Sweave**.

2 Motivation and Background

Sweave is a tool for generating "reproducible research" documents by embedding R or S "code chunks" directly into a LaTeX document. For small projects, this approach works well. For large papers or projects, heavy data analysis or computation can cause document compilation times that are unacceptable. The problem of performing lengthy computations in Sweave documents is not a new one. Previous attempts to tackle this problem include the **cacheSweave** and **weaver**[3] packages. These packages address the problem that code chunks with lengthy computations are executed every time a document is compiled. Both packages provide a `cache` option which saves R objects for quick access during successive compilations. The **cacheSweave** package stores results in a **filehash**[4] databases while the **weaver** package stores RData files. The benefit of the **cacheSweave** method is lazy loading of objects. Both methods provide significant speedup for most **Sweave** documents, namely those which create objects in the global environment.

The existing methods have some drawbacks:

1. Plots are not cached (since plots do not generally create objects in the global environment). If a plot takes a long time to generate, the same problem exists as when lengthy computations are present. Ideally we would like to reuse a plot if the code that generated it has not changed.
2. Consistency in style (font, point size) in automatically generated graphics is difficult to achieve. The default font and point size in R does not match LaTeX very well and getting this to match precisely is tricky business. The previously mentioned tools, **tikzDevice** and **eps2pgf**, counter this but using them with **Sweave** manually can be cumbersome.

The **pgfSweave** package addresses these drawbacks. The so called "caching" of plots is achieved with the help of two tools: the TeX package **PGF**[5] and the R package **tikzDevice**. When we refer to the "caching" of a graphic we mean that if the code chunk which generated the graphic is unchanged, an image included from a file rather than regenerated from the code. The TeX package **pgf** provides the ability to "externalize graphics." The effect of externalization is that graphics get extracted and compiled separately, saving time on subsequent compilations. The externalization chapter in the **PGF/Ti*k*Z** manual is extremely well written, and we refer the interested user there for more information. Externalization plus some clever checking on the part of **pgfSweave** makes up the caching mechanism.

[1] http://cran.r-project.org/web/packages/cacheSweave/index.html
[2] http://cran.r-project.org/web/packages/tikzDevice/index.html
[3] http://www.bioconductor.org/packages/2.3/bioc/html/weaver.html
[4] http://cran.r-project.org/package=filehash
[5] http://sourceforge.net/projects/pgf/

The plot style consistency drawback is addressed by the handy options `tikz` and `pgf` which allow for graphics to be output in these formats. Again, it is possible to do this manually but the chunk options make things easier.

3 System Requirements

In general **pgfSweave** depends on:

1. A working TEX distribution (such as TeXLive for linux and mac and MiKTex for Windows)
2. At least version 2.10 of the **PGF/Ti*k*Z** package for LATEX.
3. The GNU make utility for externalizing graphics.

That should be it for any *nix or Mac OS X system.

3.1 Windows specific requirements

The **pgfSweave** package can work on Windows with some special care. First of all it is strongly recommended that R be installed in a location that does not have spaces in its path name such as `C:\R`. This will save much grief when using **Sweave**. In addition, do the following in the order listed.

1. Install MiKTEX.
2. Upgrade to or install PGF 2.10 if not already done (the command mpm).
3. Install Rtools[6]. Make sure to allow the Rtools installer to modify your PATH.

If everything is set up correctly, the commands `make` and `pdflatex` or `latex` should be available at the command prompt.

4 Usage

We assume a familiarity with the usage of **Sweave**, for more information see the **Sweave** manual.[7] This section will explain the usage of the `tikz`, `pgf` and `external` options and then provide a complete example.

4.1 The `tikz` option

The first new code chunk option, `tikz`, acts the same as the `pdf` or `eps` options but instead of resulting in an `\includegraphics{}` statement the result is an `\input{}` statement. Consider the following code:

[6] http://www.murdoch-sutherland.com/Rtools/
[7] http://www.stat.uni-muenchen.de/~leisch/Sweave/Sweave-manual.pdf

Input:

```
\begin{figure}[ht]
<<tikz-option,fig=T,tikz=T,echo=F>>=
    x <- rnorm(100)
    plot(x)
@
\caption{caption}
\label{fig:tikz-option}
\end{figure}
```

Output:

```
\begin{figure}[ht]
\input{tikz-option.tikz}
\caption{caption}
\label{fig:tikz-option}
\end{figure}
```

The `.tikz` file is generated with the **tikzDevice** package. This is the default graphics output for **pgfSweave**, the `tikz` option is set to `TRUE` by default.

4.2 The `pgf` option

This option is depreciated, please use the `tikz` option instead.

The second new code chunk option `pgf`, acts the same as the tikz option in that the result is an `\input{}` statement. Consider the following code:

Input:

```
\begin{figure}[ht]
<<pgf-option,fig=T,pgf=T,tikz=F,echo=F>>=
    x <- rnorm(100)
    plot(x)
@
\caption{caption}
\label{fig:pgf-option}
\end{figure}
```

Output:

```
\begin{figure}[ht]
\input{pgf-option.pgf}
\caption{caption}
\label{fig:pgf-option}
\end{figure}
```

The `.pgf` file is generated with the **eps2pgf** utility. The `postscript` graphics device is used first to generate a `.eps` file. Then the command

```
$ java -jar /path/to/eps2pgf.jar -m directcopy graphic.eps
```

is run on every code chunk that has `fig=TRUE` and `pgf=TRUE`. We do not recommend using this option in favor of the `tikz` option. Using the `pgf` option involves two creation steps instead of one and it strips the R text styles (such as boldface).

4.3 The `sanitize` option

This option is a boolean value that accesses the `tikz()` option of the same name. Please see the `tikz()` documentation for more details.

4.4 The `external` option

Externalization is a feature of the TikZ package that is the main component of graphics caching in **pgfSweave**. Graphics that do ont change between successive runs will not be re-compiled, but simply included as pdf files. For very complex graphs, this may not actually save much time as the inclusion of

the large pdf will still take significant time. In these situations, you may want to consider using raster graphics (such as png).

Before version 1.2, **pgfSweave** used the 'old' method of externalizing graphics (`\beginpgfgraphicnamed`...). This method works fine but is fairly limited in terms of cool features. As of version 1.2, **pgfSweave** uses the 'new' externalization via the TikZ externalization library. This library has a lot of cool bells and whistles and can even make some speed improvements over the old method. Please refer to the pgf manual for a comprehensive overview of the TikZ externalization library.

The new method uses the externalization library feature `list and make`. To quote from the TikZ manual:

> The mode `list and make` is similar to `list only`: it generates the same file <main file>.figlist, but any images which exist already are included as graphics instead of ignoring them. Furthermore, this mode generates an additional file: <main file>.makefile. This allows to use a work flow like
>
> ```
> % step 1: generate main.makefile:
> pdflatex main
> % step 2: generate ALL graphics on 2 processors:
> make -j 2 main.makefile
> % step 3: include the graphics:
> pdflatex main
> ```
>
> This last make method is, however unnecessary: `list and make` just assumes that images are generated somehow (not necessarily with the generated makefile). The generated makefile allows parallel externalization of graphics on multi-core systems and it supports any file dependencies configured with `\tikzpicturedependsonfile`.

Input:

```
\begin{figure}[ht]
<<external,fig=T,tikz=T,external=T,echo=F>>=
    x <- rnorm(100)
    plot(x)
@
\caption{caption}
\label{fig:external-option}
\end{figure}
```

Output:

```
\begin{figure}[ht]
\tikzsetnextfilename{external}
\tikzexternalfiledependsonfile{external}{external.tikz}
\input{external}
\caption{caption}
\label{fig:external}
\end{figure}
```

This both registers the graphic to externalize, generating an entry in the makefile for it, and registers a dependency on the tikz file itself so that if it is changed, the graphic is re-externalized. After the **Sweave** process is done, it is necessary to run `pdflatex` and make as in the quote from the `pgf` manual. This

4.5 The \tikzexternalize commmand

The **pgfSweave** will automatically add the lines

```
\usetikzlibrary{external}
\tikzexternalize[mode=list and make]
```

below the `\usepackage{tikz}` command. The use of TikZ externalization library with **pgfSweave** requires these lines. You may want to customize the options to the externalization library, for example to generate eps graphics, to do this you should use `\tikzset`. See the externalization library documentation in the pgf manual to see how. An example is shown in the next section.

4.6 Generating other graphics formats (was 'The tex.driver option (Externalization Driver)'

As of version 1.2, the `text.driver` option is defunct. This was never really a well developed feature and the TikZ externalization driver provides a much better mechanism for generating different graphics formats (eps for example). It is also possible to generate multiple formats at once. For example, to generate a eps and pdf version of your graphic, use the following line below `\usepackage{tikz}`

```
\tikzset{external/system call={latex \tikzexternalcheckshellescape -halt-on-error
  -interaction=batchmode -jobname "\image" "\texsource";
  dvips -o "\image".eps "\image".dvi;
  pdflatex \tikzexternalcheckshellescape -halt-on-error
  -interaction=batchmode -jobname "\image" "\texsource";}}
```

Similarly you can use this method to generate Bitmap graphics, see the example in section 32.7 "Bitmap Graphics Export" of the pgf 2.10 manual.

This method can also be used to work with plain latex. The process is a bit kludgy, but it works. Use a similar line as above the generate an eps graphic then around the graphic:

```
\begin{figure}[!ht]
\centering
{
\pgfkeys{/pgf/images/include external/.code={\includegraphics[width=3in]{#1}}}
 <<plainlatex,echo=F,fig=T,width=3,height=3,tikz=T,external=T>>=
  x <- rnorm(100)
   plot(x)
 @
}% this brace ends the effect of include external
\caption{plain latex}
\end{figure}
```

Then the compilation process would be

```
  % generate tex file
R CMD pgfSweave --pgfsweave-only main.Rnw
  % generate makefile
latex main.tex
  % externalize graphics
make -f main.makefile
  % include the graphics
latex main.tex
```

```
  % finally create a pdf file
dvipdf main.dvi
```

4.7 The highlight option

Version 1.1.0 of **pgfSweave** introduced the `highlight` option. The default is `TRUE` by default so code will be syntax-highlighted with the **highlight** package. To disable highlighting add \SweaveOpts{highlight=F} to disable it. This option can be combined with the `keep.source` option or the `tidy` option. For example:

The original code in the chunk looks like:

```
    # here is a comment
x <- numeric(100)
for( i in 1:100){
    x[i] <- rnorm(1)
}
    # OR
y <- rnorm(100)  # and another comment

plot(x,y)
```

Normal:

```
> x <- numeric(100)
> for (i in 1:100) {
+     x[i] <- rnorm(1)
+ }
> y <- rnorm(100)
> plot(x, y)
```

With Highlighting:

```
> x <- numeric(100)
> for (i in 1:100) {
+     x[i] <- rnorm(1)
+ }
> y <- rnorm(100)
> plot(x, y)
```

4.8 The tidy option

Version 1.1.0 of **pgfSweave** introduced the `tidy` option. This functionality was built into earlier releases, but is now available as an option for greater control. Setting this chunk option to `TRUE` will clean up your code with an internal version of the `tidy.source` function from the `formatR` package. Using this option will not only preserve comments in the code but will reformat it's appearance. This option takes precedent over the `keep.source` function but can be combined with the `highlight` option. Using the previous example:

Default (highlighting only, cleaned up with parse):

```
> x <- numeric(100)
> for (i in 1:100) {
+     x[i] <- rnorm(1)
```

```
+ }
> z <- rnorm(100)
> plot(x, z)
```

Normal with `keep.source=T` (highlighting only, not cleaned up):

```
>         # here is a comment
>     x <- numeric(100)  # here is a comment
>     for( i in 1:100){ x[i] <- rnorm(1)}
>     z <- rnorm(100)  # and another comment
>     plot(x,z)
```

Only Tidying:

```
> # here is a comment
> x <- numeric(100)  # here is a comment
> for (i in 1:100) {
    x[i] <- rnorm(1)
}
> y <- rnorm(100)  # and another comment
> plot(x, y)
```

With Tidying and highlighting:

```
> # here is a comment
> x <- numeric(100)  # here is a comment
> for (i in 1:100) {
+     x[i] <- rnorm(1)
+ }
> y <- rnorm(100)  # and another comment
> plot(x, y)
```

NOTE: Inline comments are defined by the `tidy.source()` function to be "two or more spaces plus the hash symbol # in your source code." Two hash symbols will also not work.

4.9 Compilation Time

The combination of **cacheSweave** code caching and **pgfSweave** figure caching can provide drastic decrease in compilation time. The time speedup is highly dependednt on what code you are executing but using **pgfSweave** effectivly reduces the compilation time of **Sweave** to the time it takes to compile the LaTeX document.

4.10 A Complete Example

At this point we will provide a complete example. The example from the **Sweave** manual is used to highlight the differences. The two frame below show the input Sweave file `pgfSweave-example-Rnw.in` and the resulting tex file `pgfSweave-example-tex.in`.

pgfSweave-example.Rnw

```
\documentclass{article}

\usepackage{tikz}
\usepackage[margin=1in]{geometry}
```

```
\title{Minimal pgfSweave Example}
\author{Cameron Bracken}

\begin{document}

<<setup,echo=F>>=
setCacheDir("cache")
options(keep.space=TRUE)
@

\maketitle
This example is identical to that in the Sweave manual and is intended to
introduce pgfSweave and highlight the basic differences.  Please refer to
the pgfSweave vignette for more usage instructions.

We embed parts of the examples from the \texttt{kruskal.test} help page
into a \LaTeX{} document:

<<data,cache=T,tidy=T>>=
        #     hey, a comment
    data(airquality)
    print(kruskal.test( Ozone ~ Month, data = airquality ))  # and another
@

\noindent which shows that the location parameter of the Ozone distribution varies
significantly from month to month. Finally we include a boxplot of the data:

\begin{figure}[!ht]
\centering
%notice the new options
{\pgfkeys{/pgf/images/include external/.code={\includegraphics[width=3in]{#1}}}
<<boxplot,echo=T,fig=T,width=3,height=3,tikz=T,external=T,highlight=T>>=
  boxplot(Ozone ~ Month, data = airquality,
    main='Ozone distribution',xlab='Month',ylab='Concentration')
@
}% this brace ends the effect of include external
\caption{This is from pgfSweave. Text is typset by \LaTeX\ and so matches the
 font of the document.}
\end{figure}


\end{document}
```

pgfSweave-example.Rnw

On the input file run:

```
R> library(pgfSweave)
R> pgfSweave('example.Rnw',pdf=T)
```

or

```
$ R CMD pgfsweave example.Rnw
```

And we get (from compiling `pgfSweave-example-tex.in`):

Minimal pgfSweave Example

Cameron Bracken

April 3, 2011

This example is identical to that in the Sweave manual and is intended to introduce pgfSweave and highlight the basic differences. Please refer to the pgfSweave vignette for more usage instructions.

We embed parts of the examples from the `kruskal.test` help page into a LaTeX document:

```
>         #     hey, a comment
> data(airquality)
> print(kruskal.test(Ozone ~ Month, data = airquality))  # and another


        Kruskal-Wallis rank sum test

data:  Ozone by Month
Kruskal-Wallis chi-squared = 29.2666, df = 4, p-value = 6.901e-06
```

which shows that the location parameter of the Ozone distribution varies significantly from month to month. Finally we include a boxplot of the data:

```
> boxplot(Ozone ~ Month, data = airquality, main = "Ozone distribution",
+     xlab = "Month", ylab = "Concentration")
```

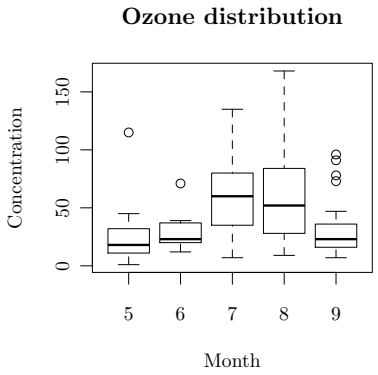

Figure 1: This is from pgfSweave. Text is typset by LaTeX and so matches the font of the document.

5 The Process

The process that **pgfSweave** uses when caching and externalization are turned on is outlined in the flow chart below:

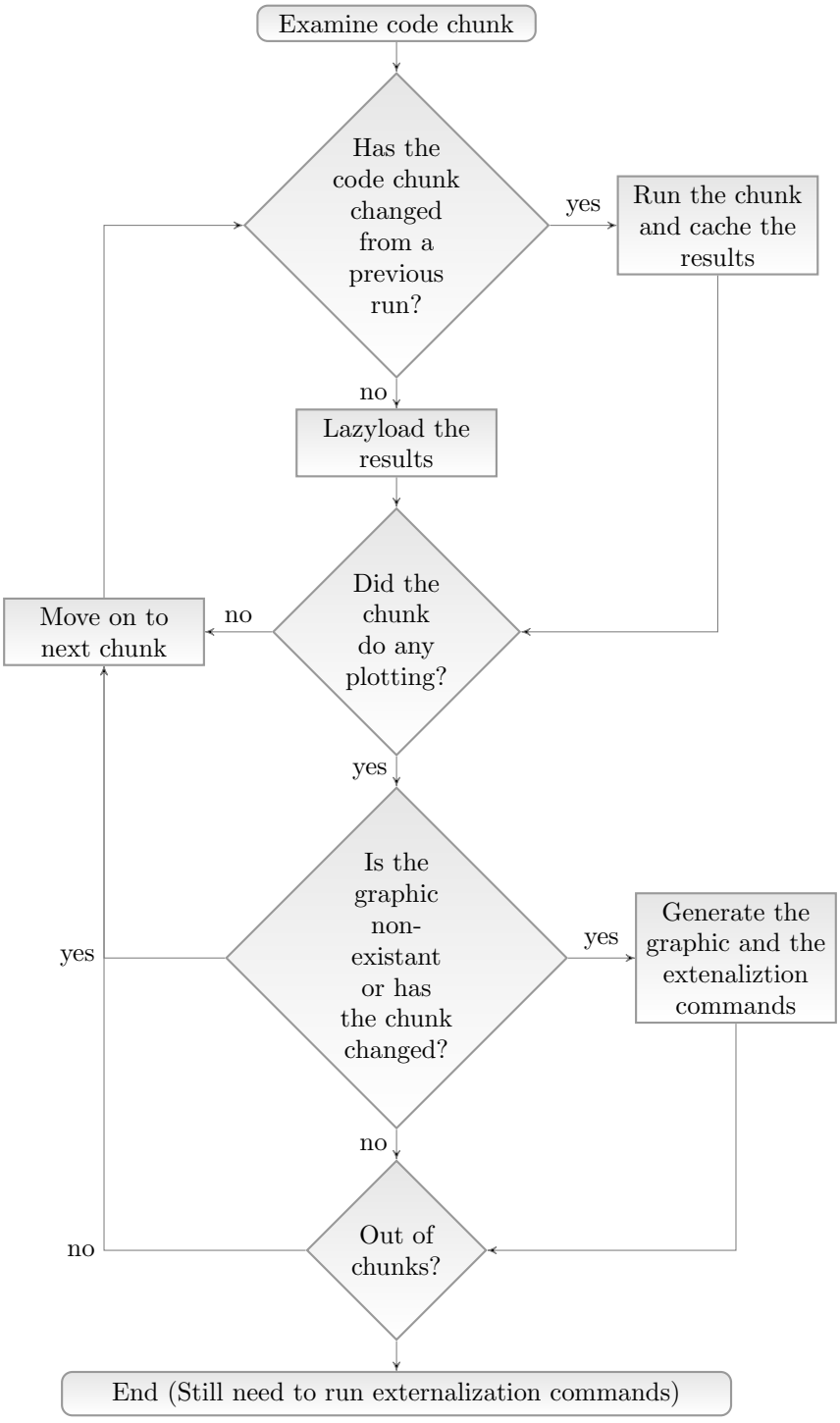

Figure 1: Flow chart of **pgfSweave** procedure.

6 Consistency in style between graphics and text

In Figure 2, notice the inconsistency in font and size between the default R output and the default LaTeX output. Fonts and font sizes can be changed from R but it is hard to be precise. What if you decide to change the font and and point size of your entire document? In Figure 3 and 4 the text is consistent with the rest of the document.

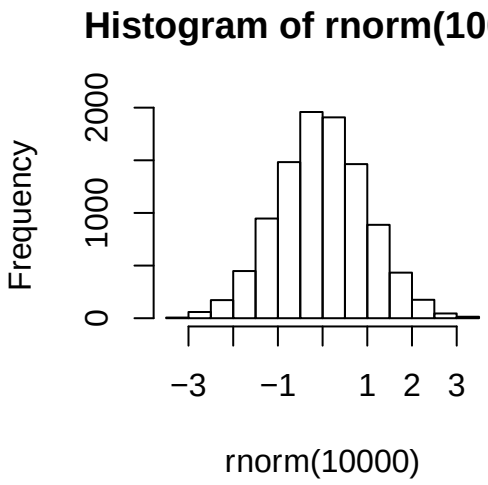

Figure 2: This is normal **Sweave**.

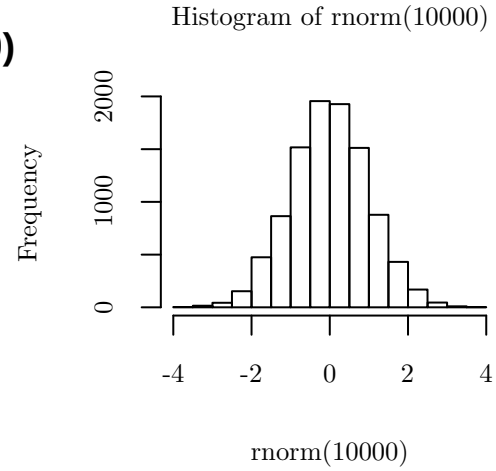

Figure 3: This is from **pgfSweave** with the `pgf` option.

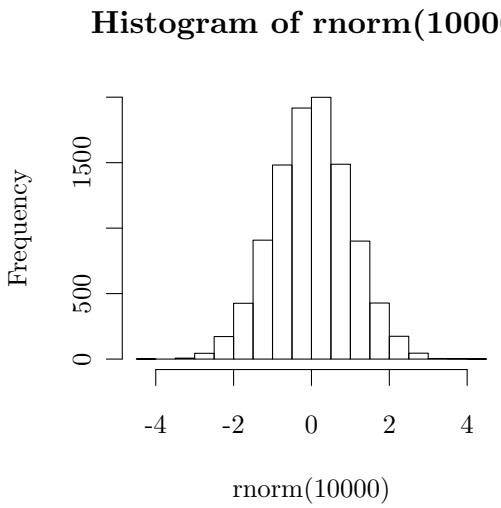

Figure 4: This is from **pgfSweave** with the `tikz` option.

7 Sweave graphic width defaults

The default in `Sweave.sty` is to fix the width of every image to 80% of the text width by using `\setkeys{Gin}{width=.8\textwidth}`. Say you have a 7 in text width and code chunk where you set `width=4`. The original 4 inch wide graphic will have text size matching your document but when it is included in your document it will be scaled up to 7 inched wide and the text will get bigger! This default is quite contrary to the philosophy of **pgfSweave**. There are two ways around this before each code chunk you can set `\setkeys{Gin}{width=<graphic width>}`. Alternatively (and the recommended way) you can turn off this feature globally by using `\usepackage[nogin]{Sweave}`, that way the width and height of the figure are controlled by the arguments to the code chunk.

8 Command line interface

In versions 0.7.0, **pgfSweave** got an `R CMD` command line interface. On Unix alikes (including Mac OS X) a symbolic link `$R_HOME/bin/pgfsweave` to `$R_HOME/library/pgfSweave/exec/pgfsweave-script.R`. On Windows a copy of the script is made instead. This script is only installed if pgfSweave is installed from source.

Here is a listing from `R CMD pgfsweave --help`:

```
        usage:
            R CMD pgfsweave [options] file

            A simple front-end for pgfSweave()

            The options below reference the following steps
              (1) Run Sweave using pgfSweaveDriver
              (2) Run the pgf externalization commands
              (3) Compile the resulting tex file using texi2dvi()

            Default behavior (no options) is to do (1), (2) then (3) in that order.

            Package repositories:
            http://github.com/cameronbracken/pgfSweave (cutting edge development)
            http://r-forge.r-project.org/projects/pgfsweave/ (precompiled dev versions)
            http://cran.r-project.org/web/packages/pgfSweave/index.html (stable release)

        options:
                -v, --version
                        Print version info and exit

                -d, --dvi
                        dont use texi2dvi() option pdf=T i.e. call plain latex to produce a dvi
                        file instead of a pdf

                -p NUMBER, --processors=NUMBER
                        Number of processors to use for graphics externalization [default 2]

                -n, --graphics-only
                        dont do (3), do (1) then (2); ignored if --pgfsweave-only is used

                -s, --pgfsweave-only
```

```
                dont do (2) or (3), only do (1)

        -h, --help
                Show this help message and exit
```

9 Frequently Asked Questions

Can pgfSweave be run from the command line?

Yes! See section 8.

```
$ R CMD pgfsweave <yourfile>.Rnw
```

Can pgfSweave be used to generate eps

Yes! See section 8.

```
$ R CMD pgfsweave <yourfile>.Rnw
```

The changes to my code chunk are not being recognized.

Occasionally **pgfSweave** suffers from overzealous caching. In these cases it may be necessary to manually delete the cache or the figure files. This is something we need to improve but this is better than compiling too often which is what used to happen.

How do I set subdirectories for figures and caches?

This is straight out of the **Sweave** and **cacheSweave** manuals (nothing new here). For a figures subdirectory [8] use the `prefix.string` option:

```
\SweaveOpts{prefix.string=figs/fig}
```

For a caching subdirectory use a code chunk at the beginning or your document like:

```
<<setup,echo=F>>=
setCacheDir("cache")
@
```

Why are the width and height options being ignored?

This is another one from **Sweave**. You must use the `nogin` option in `Sweave.sty` for the width and height parameters to actually affect the size of the image in the document:

```
\usepackage[nogin]{Sweave}
```

LaTeX/PDFLaTeX is not found in R.app (Mac OS X) and [Possibly] R.exe (Windows)

Your latex program is not in the default search path. Put a line such as:

```
Sys.setenv("PATH" = paste(Sys.getenv("PATH"),"/usr/texbin",sep=":"))
```

in your `.Rprofile` file.

[8] make sure to create the directory first!

I get a bunch of "Incompatible list can't be unboxed" errors when compiling.

This is a problem with the CVS version of PGF. The workaround is to load the **atbegshi** package before PGF or TikZ:

```
\usepackage{atbegshi}
\usepackage{pgf}
```

or

```
\usepackage{atbegshi}
\usepackage{tikz}
```

The vignette in `/inst/doc/` does not contain any code chunks!

That is because the vignette in `/inst/doc/` is a "fake" vignette generated from the "real" vignette in `/inst/misc/vignette-src/`. The reason for this extra step is that package vignettes must be able to be compiled with `R CMD Sweave`, which is precisely what we don't want to use!

To compile the vignette yourself, download the package source, unpack it and then do the following:

```
git clone git://github.com/cameronbracken/pgfSweave.git
R CMD INSTALL pgfSweave
cd pgfSweave/inst/misc/vignette-src/
make
```

Which will create `pgfSweave-vignette-source.pdf`

10 Additional Resources

This is an on going list of third-party resources for **pgfSweave**. Suggestions for this list are welcome.

- http://code.cjb.net/pgfSweave.html
- http://yihui.name/en/tag/pgfsweave/