Efficient use of `pgfSweave` on large data-sets

Hans Ekbrand

January 6, 2011

Contents

1 Summary

This document is rather long, so here is the summary if you are in a hurry:
Create a directory to hold the cache.

```
mkdir -p ~/sweave-cache/figs
mkdir -p ~/sweave-cache/values
```

Here is a template `Sweave` document.

```
────── my-title.Rnw ──────
\documentclass{article}
\usepackage{tikz}
\usepackage[nogin]{Sweave}
\ SweaveOpts{echo=F,eval=T,fig=F,results=hide,
             cache=T,prefix.string=sweave-cache/figs/fig}
\begin{document}
<<setup,cache=F>>=
setCacheDir("sweave-cache/values")
@
\end{document}
```

For an example that actually uses the tips on how to use the caching mechanism smart, see this complete example.

2 Intended usage, target audience

This article focus on real world applications of `pgfSweave`, involving writing articles based on really large datasets. Using `pgfSweave` for teaching purposes, typically do not involve large datasets so there these advice is likely not useful. On the contrary, not imposing the rules I propose here will do your creativity good when writing `Sweave` documents for teaching `R`, particularly since it is a good a teaching practice to dynamically make up the data to operate on.

3 How to get your `Sweave` documents to compile super fast

The trick is to encapsulate all calculations that are being done on **large objects** into functions. Since functions do not affect the top environment, there is no large objects to cache - or read from the cache, and the compilation will be super fast. For a chunk not to be repeated, it still has to have `cache=T`, even if it does not actually save anything to the cache. In `pgfSweave`, `rm()` does not work as it normally does, which is why you really should wrap the code in a function instead of merely `rm()` the large object(s) before closing the chunk.

Trying to minimise the stuff that will be cached will **enforce good coding routines**, since you will have to write functions that return what you want and it will

be clear what objects/variables in your code are only temporarily used as means to summarize data. The temporary stuff will never turn up in the global environment nor in the cache.

4 All your code-chunks should be one of these five types

1. a function to **shape-up the raw data** the way you want it (e.g. import spss file(s), convert some factors to numeric, convert value labels to UTF-8, remove uninteresting cases/variables, merge data-sets, . . .), save it in a RData-file and not returning anything at all, **cached**.

2. a function **summarize data**, returning an object or a list of objects that should be plotted or tabulated by the other two types of chunks; **cached**.

3. a **figure generator**, **cached**

4. a **table or value output generator**, **not cached**.

5. a **configurator/sourcing chunk**, **not cached**.

4.1 Data shape-up chunk

This type of chunk hold the code used to create the R object(s) that further analysis will use. Typically this involves:

- import data from external formats (`.csv`, `.sav`, etc.)
- merge data-sets
- combine data from various external sources to one R object (e.g. a dataframe).
- remove uninteresting cases or variables
- saving the R object(s)

By hiding away this work within a function, the cache is not cluttered with the raw data. The function does not return anything, so nothing is cached. But still `cache=T` is needed, otherwise the chunk will be repeated the next run. Here is a minimal example, which reads a `csv`-file, removes columns 14 to 21 and saves the object in a file `foo.RData`.

```
────── data shape-up chunk ──────
<<importing.files>>=
importing.files.function <- function () {
  foo <- read.csv("original-source.csv")
  foo <- foo[,-c(14:21)]
  save(foo, file="foo.RData")
}
importing.files.function()
@
```

4.2 Data summarize chunk

Chunks of this type load the object from a file, run some analysis on it, and return the interesting parts of the summary object of the analysis. When the analysis is `lm`, `summary(lm(formula = ...))` is not small enough, save only the `$coefficients`.

```
————————————————— data summarize chunk ————————————————
<<load.foo.and.run.regressions>>=
regressions.function <- function() {
  load("foo.RData")

  ## total crime activity
  my.fit.tca <- summary(lm(formula = total.crime.activity ~ sex +
  one.parent.immigrant + both.parents.immigrants +
  both.parents.and.child.imigrants + socio.economic.status +
  parents.do.not.live.together + good.contact.with.parents +
  parents.usually.monitors.child + likes.school + school.grades +
  can.talk.openheartedly.to.friends + friends.tolerate.crimes +
  friends.tolerate.alcohol, data = foo))$coefficients

  ## fire-setting
  my.fit.fs <- summary(lm(formula = fire.setting ~ sex +
  one.parent.immigrant + both.parents.immigrants +
  both.parents.and.child.imigrants + socio.economic.status +
  parents.do.not.live.together + good.contact.with.parents +
  parents.usually.monitors.child + likes.school + school.grades +
  can.talk.openheartedly.to.friends + friends.tolerate.crimes +
  friends.tolerate.alcohol, data = sd.1995))$coefficients

  list(my.fit.tca=my.fit.tca, my.fit.fs=my.fit.fs)
}
regressions.results <- regressions.function()
@
```

`regressions.results` is returned and cached and can later be used by ploting or table-generating chunks.

Admittedly, the line

```
list(my.fit.tca=my.fit.tca, my.fit.fs=my.fit.fs)
```

sure is ugly. The function `llist` in Frank E Harrell Jr's useful package Hmisc gives the same result using a much nicer syntax:

```
llist(my.fit.tca, my.fit.fs)
```

4.3 Data plot chunk (a figure generator)

The data plot chunk type should use a summarized object created by a data summarize chunk. Since the data plot chunks is cached, you won't bloat the cache if you have that code in the same chunk as the data summarizing code, but there are other good reasons for keeping ploting and summarizing in different chunks:

- you can change parameters in the plot without having to redo the calculations

- you can generate more than one plot

Data plot chunks must have `fig=T` in the definition of the chunk. In the example below, a previous chunk has summarized data into two objects with long and descriptive names, which are now being plotted.

```
────── a figure generating chunk ──────
<<age,fig=T>>=
plot(15:20, misstankta.summary.results$prop.fire.setters.by.age,
     type = "l", ylab = "Proportion of suspects", xlab = "Age",
     col = "blue", ylim = c(0,31))
lines(15:20, misstankta.summary.results$prop.non.fire.setters.by.age,
     col = "red")
@
```

4.4 The table or value output generator

Unlike data plot chunks, **tables cannot be cached**. Tables are printed output, a sort of anonymous objects. Therefore a table generator chunk will have to be evaluated everytime `pgfSweave` processes the document. This is not a problem, since generating a LATEX-table is done in a fraction of a second even on a slow computer, if the figures that should go in it are already calculated. In the example below, the figures are cached in the variable `regressions.results$my.fit.tca` which was the result of the `regressions.function()` defined and evaluated in the `load.foo.and.run.regressions` chunk above.

```
────── a table or value output generator ──────
<<regression.fit.total.table,results=tex,cache=F>>=
my.table.lmfit(regressions.results$my.fit.tca)
@
```

4.5 Configuration/sourcing chunk

This chunk type configures R at runtime or sources general - stable - R functions. In the first case, it does not return anything, and in the latter, you want it to source the current version of the functions, so **caching is not meaningful here**. Typical examples include

1. sourcing general R functions from external files

4.5.1 Sourcing general R functions from external files

While sourcing functions from external files in a way counters the aim of a single file as a the source for the analysis, you may still want to source some general utility functions, e.g. for laying out tables and graphs. Another way is to create proper R packages (and publish them), but for most users, I think that involves an unecessary extra task. As an example I have a few wrapper functions for xtable, which I prefer not to include in every `.Rnw`-file.

```
────── sourcing general R functions ──────
<<load-my-functions,cache=F>>=
## load my.table, my.table.lmfit and my.table.fact.anal
source("text/src/r/my.xtable.r")
@
```

4.5.2 Configuring `pgfSweave` to use a particular directory as cache:

Example:

```
────── configuring - pgfSweave ──────
<<setup,cache=F>>=
setCacheDir("sweave-cache/values")
@
```

4.5.3 Setting width on auto-generated output:

Example:

```
────── configuring - R ──────
<<set.width,cache=F>>=
options(width=45)
@
```

5 Understanding caching in `pgfSweave`

5.1 Never, ever, close a code-chunk which has variables/objects containing raw data

It will hurt your harddisk, and waste RAM every run.

I once made four simple regressions on a large data-set, and printed a table with statistics for each regression. The wrong way of doing it is to run the first regression and print a table and close the chunk, then start a new chunk, run the second regression, print the table with statistics from this run, and so on. The problem with that method is that the large data-set itself will be cached. The right way of doing it is to load

the data-set within a function, run the four regressions in that function and save (return) the coefficients part of the lm.fit object for each regression, **caching only the coefficients of the lm.fit object not the data-set itself**. The actual printing of the statistics - in nice LaTeX-tables - is done in four later chunks, which can have normal text between them.

Admittedly, if I change a parameter in one of these regressions, the whole chunk will have to be run again. But that is - in my usage of sweave a rather uncommon event. First you develop the code that does what you want, then you include in your article. If you have relatively small data-sets and more complicated computations you might off course think differently, or load the data-set separately for each chunk.

5.2 Strategies for weaving

Weaving is nice, it makes a `.Rnw`-file easy to follow. If you are bothered that the rules I suggest in this document will tend to decrease weaving and make you write very large data summarizing chunks, then note that as long as you wrap loading of the large data-set into a function, you can still weave code-chunks with text in the normal way **as long as you load the data-set from disk in each chunk**. For chunks with `cache=T`, the performance hit of `save()` and `load()` is small since such chunks are only evaluated once. Thus, you can keep the data summarizing chunks as small as you want them.

5.3 Anonymous objects are not cached!

Do not try to cache the return value of a function, it will not work unless a variable is used to "catch" the value.

- Variables (named objects, that exists in the global environment) is automatically cached by `pgfSweave`.

- Plots are cached by `pgfSweave`.

A typical example of an anonymous object that **will not be cached** is the return value of a function that creates LaTeX-code, for example `xtable` (or more correctly the "`print`" method of the "`xtable`" class).

To maximise the caching effect, save as much as absolutely possible in variables in a separate code chunck that is cached, before a non-cached code chunk that merely prints the values in the saved variables. This strategy is mentioned in the documentation of `cacheSweave`.

6 Some other things to note

6.1 If you set a cachedir in sweaveopts, you have to create the dir manually.

```
\ SweaveOpts{echo=FALSE,keep.source=TRUE,prefix.string=sweave-cache/figs/fig}
```

will fail unless `sweave-cache/figs` exists.

6.2 No dots in the name of chunks with fig=T

When `pdfSweave` is called with `pdf=T`, there must be no dots in the name of the chunk, if the cunk has `fig=T`.

```
<<prevalence.means.plot,fig=T,width=4,height=3.5>>=
```

will fail, try "-" instead of ".":

```
<<prevalence-means-plot,fig=T,width=4,height=3.5>>=
```

For chunks with fig=F (or when target output is a `dvi`-file) dots are OK in the name:

```
<<regression.fit.stealing.from.car.table,fig=F>>=
```

6.3 Consider runing `pgfSweave` from within an active `R` session rather than from a shell-script

You will gain the time it takes to start up R and load pgfSweave and the library it depends on. Each and every time you rebuild your document. As a bonus, it will save RAM too.

```
pgfSweave(file="Young-arsonists.Rnw", compile.tex=F, pdf=T)
```

The `pdf=T` is passed on to `tex2dvi`, It is there for `texi2dvi` to create `pdf`-files rather then `dvi`-files.

I use `compile.tex=F` because I use a custom make command to run `pdflatex` (This way I can use my favourite LaTeX-packages without having to specify them in the every `.tex`-document).

7 Complete example

```
────────────────── my-filename.Rnw ──────────────────

\documentclass{article}
\usepackage{tikz}
\usepackage[nogin]{Sweave}
\ SweaveOpts{echo=F,eval=T,fig=F,results=hide,cache=T,tikz=T,
            external=T,prefix.string=sweave-cache/figs/fig}
\begin{document}
<<setup,cache=F>>=
setCacheDir("sweave-cache/values")
```

```
@

% First a chunk that only does some data washing and merges two
% dataset. The function returns nothing, so nothing is cached,
% but since cache=T, it will only be evaluated once

<<shape-up-data>>=
shape.up.data.function <- function() {
  foo <- read.spss("~/datasets/foo.sav")
  ## remove three last columns 937:939
  foo <- foo[,1:936]
  my.variable.indices <- function(x, name, data.frame=foo) {
    base <- which(names(data.frame) == name)
    seq(from = base, to = (base+28)*30, by = 28)[x] }
  ## wash the values of the date variables, by removing the separator "-"
  ## "2008-08-17" -> "20080817"
  for(i in 1:max(foo$AntBrott)){
    foo[,my.variable.indices(i, "B1_from")] <-
        gsub("-", "", foo[,my.variable.indices(i, "B1_from")]) }
  ## add a column indicating that this is the
  ## control/reference-group, control = T
  foo[,ncol(foo)+1] <- T
  colnames(foo) <- c(colnames(foo)[-ncol(foo)], "control")
  ## Load dataset bar, which is already in RData format.
  load("~/datasets/bar.RData")
  ## was list, want data.frame
  bar <- as.data.frame(bar)
  ## remove columns 937:939
  bar <- bar[,1:936]
  ## add a column indicating that this is the fire-setters, control = F
  bar[,ncol(bar)+1] <- F
  colnames(bar) <- c(colnames(bar)[-ncol(bar)], "control")
  ## merge data-sets, but first modify LOPNR in bar, so that LOPNR is
  ## unique (length(foo[,1] + LOPNR makes LOPNR unique in bar.)
  bar$LOPNR <- bar$LOPNR + length(foo[,1])
  final <- rbind(bar, foo)
  save(final, file="~/datasets/final.RData")
  return()
}
shape.up.data.function()
@
% then a summarizing function, which only caches a summary of each regression
<<summarize-data-one>>=
summarize.data.function <- function() {
  load("~/datasets/final.RData")

  my.fit.a <- summary(lm(formula = abuse ~ sex + one.parent.immigrant
  + both.parents.immigrants + both.parents.and.child.imigrants +
  socio.economic.status + parents.do.not.live.together +
```

9

```
    good.contact.with.parents + parents.usually.monitors.child +
    likes.school + school.grades + can.talk.openheartedly.to.friends +
    friends.tolerate.crimes + friends.tolerate.alcohol, data =
    sd.1995))$coefficients

    my.fit.sfc <- summary(lm(formula = stealing.from.car ~ sex +
    one.parent.immigrant + both.parents.immigrants +
    both.parents.and.child.imigrants + socio.economic.status +
    parents.do.not.live.together + good.contact.with.parents +
    parents.usually.monitors.child + likes.school + school.grades +
    can.talk.openheartedly.to.friends + friends.tolerate.crimes +
    friends.tolerate.alcohol, data = sd.1995))$coefficients

    ## return a list with a summary of each regression
    list(my.fit.a=my.fit.a, my.fit.sfc=my.fit.sfc)
}
regressions.results <- regressions.function()
@

% Now, we want to print a table in a non-caching code-chunk. The
% result of this code chunk is that a table is printed, and since
% printing cannot be cached, there is no point in caching this
% code-chunk.

\section{Factors explaining abuse}
In the table below summaries a linear regression with abuse as the
dependent variable.
<<table-one,cache=F,results=tex>>=
xtable(regressions.results$my.fit.a)
@
% Finally, a caching code chunk that outputs a picture, type-set with tikz,
% thus with the same font as the main document.
<<picture-one,fig=T,tikz=T,external=T>>=
plot(regressions.results$my.fit.a)
@
\end{document}
```