An Org-mode Demo

Eric Schulte

16 June 2010

Adapted from *An Sweave Demo* by Charles J. Geyer.

This is a demo for using Org-babel to produce LaTeX documents with embedded R code. To get started fire up Emacs and create a text file with the `.org` suffix. You should see Org-mode become your major mode – denoted by `Org` in your status bar.

Press `C-c C-e` while viewing this Org-mode buffer and you will see a menu appear with options for export to a variety target formats – herein we'll only consider export to LaTeX.

So now we have a more complicated file chain

$$\texttt{foo.org} \xrightarrow{\texttt{Sweave}} \texttt{foo.tex} \xrightarrow{\texttt{latex}} \texttt{foo.dvi} \xrightarrow{\texttt{xdvi}} \text{view of document}$$

and what have we accomplished other than making it twice as annoying as the WYSIWYG crows (having to use both `Org-mode` and `latex` to get anything that looks like the document)?

Well, we can now include `R` in our document. Here's a simple example

```
2 + 2
```

What I actually typed in `foo.org` was

```
#+begin_src R :exports both
  2 + 2
#+end_src
```

This is a "code block" to be processed by Org-babel. When Org-babel hits such a thing, it processes it, runs R to get the results, and stuffs the output in the LaTeX file it is creating. The LaTeX between code chunks is copied verbatim (except for in-line src code, about which see below). Hence to create a *active* document you just write plain old text interspersed with "code blocks" which are plain old R.

Plots get a little more complicated. First we make something to plot (simulated regression data).

```
n <- 50
x <- seq(1, n)
a.true <- 3
b.true <- 1.5
y.true <- a.true + b.true * x
s.true <- 17.3
y <- y.true + s.true * rnorm(n)
out1 <- lm(y ~ x)
summary(out1)
```

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-51.252  -9.922   1.613  12.315  32.141 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -3.2039     4.9776  -0.644    0.523    
x             1.5299     0.1699   9.005 6.96e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 17.34 on 48 degrees of freedom
Multiple R-squared: 0.6282,     Adjusted R-squared: 0.6204 
F-statistic:  81.1 on 1 and 48 DF,  p-value: 6.962e-12
```

(for once we won't show the code chunk itself, look at `foo.org` if you want to see what the actual code chunk was).

Figure 1 (p. 3) is produced by the following code

```
plot(x, y)
abline(out1)
```

Note that `x`, `y`, and `out1` are remembered from the preceding code chunk. We don't have to regenerate them. All code chunks are part of one R "session".

Now this was a little tricky. We did this with two code chunks, one visible and one invisible. First we did

```
 #+srcname: fig1plot
 #+begin_src R :exports code :file fig1plot.pdf
   plot(x, y)
   abline(out1)
 #+end_src
```

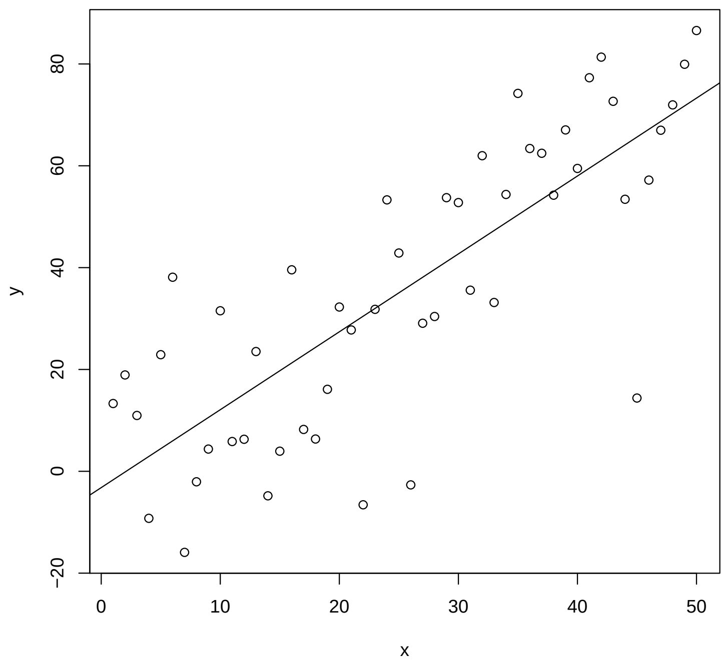

Figure 1: Scatter Plot with Regression Line

where the `:exports code` indicates that only the return value (not code) should be exported and the `#+srcname: fig1plot` gives the code block a name (to be used later). And "later" is almost immediate. Next we did

```
#+source: fig1
#+begin_src R :exports results :noweb yes :file fig1.pdf
  <<fig1plot>>
#+end_src
```

In this code block the `:file fig1.pdf` header argumentindicates that the block generates a figure. Org-babel automagically makes a PDF file for the figure, and Org-mode handles the export to LaTeX. The `«fig1plot»` is an example of "code block reuse". It means that we reuse the code of the code chunk named `fig1plot`. The `:exports results` in the code block means just what it says (we've already seen the code—it was produced by the preceding chunk—and we don't want to see it again, we only want to see the results). It is important that we observe the DRY/SPOT rule (*don't repeat yourself* or *single point of truth*) and only have one bit of code for generating the plot. What the reader sees is guaranteed to be the code that made the plot. If we had used cut-and-paste, just repeating the code, the duplicated code might get out of sync after edits. The rest of this should be recognizable to anyone who has ever done a LaTeX figure.

So making a figure is a bit more complicated in some ways, but much simpler than others. Note the following virtues

- The figure is guaranteed to be the one described by the text (at least by the R in the text).
- No messing around with sizing or rotations. It just works!

Note that if you don't care to show the R code to make the figure, it is simpler still. Figure 2 shows another plot. What I actually typed in `foo.org` was

```
#+srcname: fig2
#+begin_src R :exports results :file fig2.pdf
  out3 <- lm(y ~ x + I(x^2) + I(x^3))
  plot(x, y)
  curve(predict(out3, newdata=data.frame(x=x)), add = TRUE)
#+end_src
```

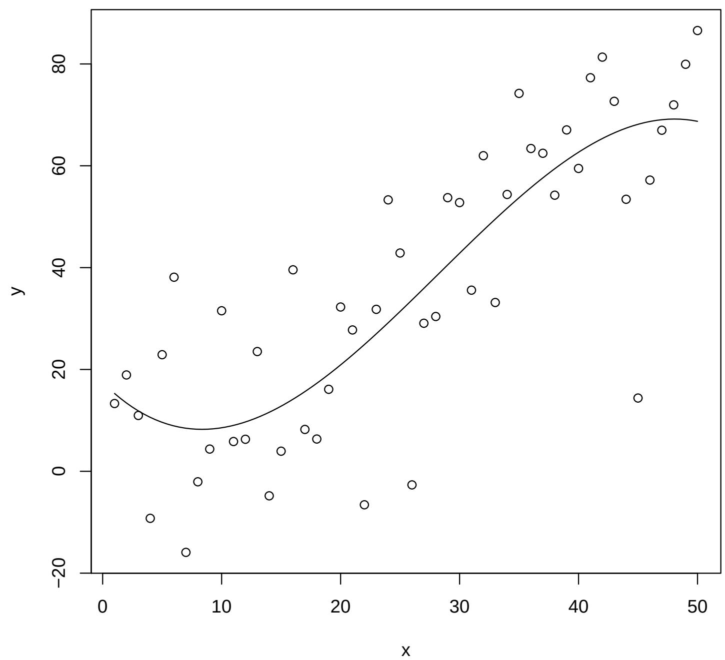

Figure 2: Scatter Plot with Cubic Regression Curve

Now we just excluded the code for the plot from the figure (with `:exports results` so it doesn't show).

Also note that every time we re-export Figures 1 and 2 change, the latter conspicuously (because the simulated data are random). Everything just works. This should tell you the main virtue of Org-babel. It's always correct. There is never a problem with stale cut-and-paste.

Simple numbers can be plugged into the text with the `src_R` command, for example, the quadratic and cubic regression coefficients in the preceding regression were β_2 = 0.1647 and β_3 = -0.0019. Just magic! What I actually typed in `foo.org` was

```
were \beta_2 =  src_R{round(out3$coef[3], 4)}
and \beta_3 =  src_R{round(out3$coef[4], 4)}
```

The `xtable` command is used to make tables. (The following is the Org-babel output of another code block that we don't explicitly show. Look at `foo.org` for details.)

```
out2 <- lm(y ~ x + I(x^2))
foo <- anova(out1, out2, out3)
foo
```

```
Analysis of Variance Table

Model 1: y ~ x
Model 2: y ~ x + I(x^2)
Model 3: y ~ x + I(x^2) + I(x^3)
  Res.Df   RSS Df Sum of Sq      F  Pr(>F)
1     48 14424
2     47 13991  1    433.37 1.5405 0.22083
3     46 12940  1   1050.85 3.7356 0.05944 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
class(foo)
```

```
[1] "anova"      "data.frame"
```

```
dim(foo)
```

```
[1] 3 6
```

```
foo <- as.matrix(foo)
foo
```

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	48	14424.45				
2	47	13991.08	1	433.37	1.541	0.221
3	46	12940.23	1	1050.85	3.736	0.059

Table 1: ANOVA Table

```
Res.Df      RSS Df Sum of Sq        F     Pr(>F)
1     48 14424.45 NA        NA       NA         NA
2     47 13991.08  1  433.3695 1.540545 0.22082917
3     46 12940.23  1 1050.8523 3.735576 0.05943547
```

So now we are ready to turn the matrix `foo` into Table 1 using the R chunk

```
#+begin_src R :results output latex :exports results
  library(xtable)
  xtable(foo, caption = "ANOVA Table", label = "tab:one",
      digits = c(0, 0, 2, 0, 2, 3, 3))
#+end_src
```

(note the difference between arguments to the `xtable` function and to the `xtable` method of the `print` function)

To summarize, Org-babel is terrific, so important that soon we'll not be able to get along without it. It's virtues are

- The numbers and graphics you report are actually what they are claimed to be.
- Your analysis is reproducible. Even years later, when you've completely forgotten what you did, the whole write-up, every single number or pixel in a plot is reproducible.
- Your analysis actually works—at least in this particular instance. The code you show actually executes without error.
- Toward the end of your work, with the write-up almost done you discover an error. Months of rework to do? No! Just fix the error and rerun `Sweave` and `latex`. One single problem like this and you will have all the time invested in `Sweave` repaid.
- This methodology provides dicipline. There's nothing that will make you clean up your code like the prospect of actually revealing it to the world.

Whether we're talking about homework, a consulting report, a textbook, or a research paper. If they involve computing and statistics, this is the way to do it.