A minimal Org setup to write scientifc notebooks

April 2022

As a matter of fact, I no longer use Emacs. I switched to (Neo)vim a while ago now, since I found myself more comfortable editing text in Vim than I ever was in Emacs. To be honest, I don't really miss any other fancy parts from Emacs operating system, except maybe the ability to run REPL for mutliple languages within a few clicks (actually, I used to use `C-c C-c` most of the times), and I really don't miss the package dependencies mess that occured from time to time chen upgrading everything. To tell the truth I don't have great requirements in terms of text editor, but I want a responsive editor, which facilitates text manipulation and fuzzy search within a few clicks.

However, Emacs is there on my machine, with the bare essentials in 30 LOC of `init.el`, and Org is readily available from any decent package manager on Linux distros. The following wa written on not so recent versions of Emacs (26.3) and Org mode (9.3.1). Also, I will focus on scientific programming languages, namely R, Stata, Python and Mathematica. In the past I used to use Org mode mostly for functional programming languages (Scheme, Common Lisp and Clojure). For an overview of what Org is good for, take a look at Emacs org-mode examples and cookbook.

I tend to view Org as a three-fold utility. First, it is a very good markup language, which also happens to be more clean and rich than Markdown. You don't need to worry about spaces for hard breaklines, there's a verse environment, as well as todo or progress state indicators and various other things that can be managed under the umbrella of the `#+PROPERTY` element. Second, Org mode in Emacs comes with handy exporting facilities (think of Pandoc, but built in Emacs directly). Third, Org introduced Babel a while ago, which allows to evaluate code directly into an Emacs buffer or when exporting.[1] As such, this provides a way to do literate programming right into your preferred text editor, even if it's (Neo)vim! Of course, if you do not work under Emacs, you lose the ability to evaluate chunks of code right into Emacs, much like an interactive playground. However, you can still evaluate the whole document and export it to HTML or PDF, much like if you were sourcing the whole buffer in Emacs.[2]

The rationale is as follows: We could use general purpose tool like dexy or noweb, or more specialized one (Sweave, knitr, pweave, staweave), use built-in exporters (e.g., from Mathematica markup language), or all-in-one solution in the browser as in Jupyter notebooks. I don't really like working in my web browser, and for what matters I don't need a digital playground, but rather a way to embed snippets of code and their outputs into my document.

[1] There are many other thing built in Org mode, especially for "getting things done", which motivated the original development of Org, but I am not so much interested in these aspects.

[2] Note that the sniprun Neovim plugin allows to run lines/blocs of code from different languages, mimicking the inline evaluation available in Emacs.

How to write Org documents

The Org website comes with nice tutorials. Read them, you will learn the basic syntax for highlighting and delineating your text. Next comes the Babel aspect of Org. Each chunk of code will read more or less like the following snippet:[3]

[3] Example taken from Nicolas Rougier's 100 numpy exercises.

```
#+BEGIN_SRC python
import numpy as np
Z = np.zeros((10,10))
print("%d bytes" % (Z.size * Z.itemsize))
#+END_SRC
```

Everything between the `#+BEGIN_SRC` and `#+END_SRC` statements is pure Python code, as indicated in the header, just after `#+BEGIN_SRC`. This is much like Markdown fenced code blocks. Normally, such a code chunk can be evaluated in Emacs by pressing `C-c C-c`, and a `#+RESULTS` block will be displayed right after the source code. The header arguments determine how code should be processed and displayed. It can be global (i.e., valid for all code chunks in the current buffer) or local (i.e., only for the current code chunk). In the latter case, it is specified right after the language (here, `python`). Otherwise, we can put a general statement at the beginning of the document, and update header options on the go. Here is header stuff that you probably want to put at the top of your Org document:

```
#+PROPERTY: header-args :cache no :exports both :results output :session
```

How to proceed your Org documents