ESS tutorial, UseR! 2011 meeting

Stephen Eglen

2011-08-15

Outline

Aims of the tutorial

- Introduce Emacs and ESS
- Reproducible research: Sweave and Org-babel
- Q & A / Discuss future development
- I'd like this to be very interactive. Feel free to ask questions/comment.

Course material

- All available via http://www.damtp.cam.ac.uk/user/sje30/ess11
- ess11.zip has the relevant material for today.

Disclaimers

- Learning Emacs takes a long time.
- I'm a core-developer on Emacs and ESS.
- I do not speak for emacs-devel or ess-core.
- ESS has several statistical interfaces; only R will be discussed today.
- Emacs (not XEmacs and SXEmacs) preferred.
- I have limited knowledge of Windows.

These slides

- Although there are pdf slides, probably most of the time I'll be showing you emacs.
- Of course, we could just view these slides in Emacs! (View `ess-slides.pdf` in Emacs ...)
- These slides are written in Org mode/Beamer. To compile them yourself, you will need:

```
(setq org-beamer-frame-level 2)
```

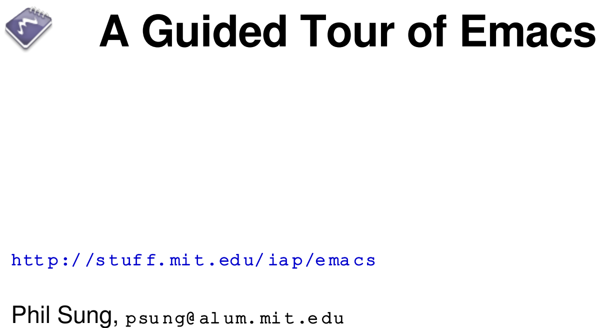
Acknowledgements

- ESS-core, in particular Tony Rossini, who gave an 2006 tutorial, from which these notes are based
- ESS-help contributors
- Org mode team, esp. Carsten Dominik, Eric Schulte and Dan Davison

Outline

Why Emacs?

- Being productive with Emacs (Phil Sung).
  http://web.psung.name/emacs/
  http://www.gnu.org/software/emacs/tour
- Explore the guided tour file:



A Guided Tour of Emacs

http://stuff.mit.edu/iap/emacs

Phil Sung, psung@alum.mit.edu

History of Emacs

- Key architect: Richard Stallman.
- Evolution from terminal only to rich graphical interface with remote editing support.
- Emacs release cycle is slow
- XEmacs: 1990s gained prominence as alternative, more graphical-friendly.
- Emacs 23 very stable.
- Emacs 24 due out next year?

Keystrokes, keybindings and functions

- Each keystroke is bound to a lisp function, even simple characters like “a”.
- `C-h k a`
- `C-h k C-a`

These create new *buffers* which are shown in new windows.

- `M-x` is bound to execute-extended-command, which allows you to run a command by name (there are many commands that are not bound to keys).
- Mapping between keybindings and commands is flexible; can change on fly e.g.

```
(global-set-key "\C-ca" 'org-agenda)
```

Understanding keystrokes

- `TAB` is the TAB (indent) key.
- `RET` is the Return (carriage return, enter) key.
- `C-h` means press control key AND "h" together
- `ESC-h` means press ESC key THEN "h"
- `M-h` means press ALT or Meta key AND "h" together.
- `M-C-h` means press Meta or Alt while pressing h and control key. OR (if no meta/alt): ESC THEN (control and h keys together).
- Older keyboards (and sometimes older Macs) without ALT or Meta lead to confusion between ESC and Meta, but ideally they should be different.

```
;; change CMD modifier to meta key (usually super).
(setq mac-command-modifier 'meta)
```

Example keybinding that makes a key 'smart'

```
  ;; From Peter.Weiss@Informatik.Uni-Oldenburg.DE (Peter Weiss)
  ;; Sun Nov 12 1995
  (defun match-paren (arg)
    "Go to the matching parenthesis if on parenthesis
otherwise insert %."
    (interactive "p")
    (cond ((looking-at "\\s\(")
           (forward-list 1) (backward-char 1))
          ((looking-at "\\s\)")
           (forward-char 1) (backward-list 1))
          (t
           (self-insert-command (or arg 1)))))

  (global-set-key "%"           'match-paren)
```

Moving Around

C-v	Move forward one screenful
M-v	Move backward one screenful
C-l	Clear and redraw screen
M- ->	Meta-<right> - moves forward a word
M- <-	Meta-<left> - moves back a word
M- |^	Meta-<up> - move up a paragraph
M- V	Meta-<down> - move down a paragraph
M- <	Meta-<less than> - move to file start
M- >	Meta-<greater than> - move to file end

Cut and Paste

```
C-d    _D_elete
C-k    _K_ill from the cursor position to
       end of line
C-y    Recover/Paste (_Y_ank) killed text
       (repeat to copy)
M-y    recover former killed text (after C-y).
       Repeat to go back through stack).
C-x u  _U_ndo  (multiple undo/redo)
```

Loading / Saving Files

```
C-x C-f  _F_ind a file
C-x C-s  _S_ave the file

If you find a second file with C-x C-f,
the first file remains inside Emacs.
You can switch back to it by finding it
again with C-x C-b.  This way you can get
quite a number of files inside Emacs.
```

Managing Buffers and Windows

C-x O	Move between windows (Oh, not Zero!)
C-x 1	One window (i.e., kill all other windows).
C-x b	Switch to new _b_uffer
C-x C-b	List _b_uffers

Search and Replace

```
M-x (then) replace-string
           Replace string
M-x (then) query-replace-string
           Will ask you, for each match,
           if you really want to replace
           the old string with the new one.
C-s        _S_earch forward (repeat to
           reuse past search strings)
C-r        Search _R_everse (repeat to
           reuse past search strings)
```

Misc Emacs Actions

```
C-h or C-h ?    _H_elp
C-h c (command) _H_elp on this _c_ommand

C-u 8 (character or command)
             Repeat character/command 8 times

C-g          Stop, unhang.
C-x C-c      Stop and exit (_c_lose) Emacs
```

Evaluating lisp

Sometimes you will see me evaluate lisp code within emacs, rather than typing a command. These are equivalent:

```
M-x next-line
(next-line)
above line eval’d using C-x C-e
```

Repeating yourself

Repeat C-n ten times:

C-u 10 C-n

C-u 70 #

Can anyone guess what the following will do?

C-u C-n

C-u C-u C-n

Keyboard macros allow for arbitrary key sequences to be repeated ...

Keyboard macros

`C-x (` lots of stuff `C-x )`
Can include counters. e.g.

```
 C-x ( TAB step C-x C-k C-i RET C-x )
```

will make:

```
   step 0
   step 1
   step 2
```

...
(info "(emacs)Keyboard Macros")

Point, mark and region

- *point* is current location of cursor
- *mark* `C-SPC` to define another point
- *region* is text between mark and point
- `C-x C-x` will swap point and mark.

Buffers

- Can contain:
  1. contents of files, e.g. ess-slides.org
  2. output from commands, e.g. `M-x man RET pwd RET`
  3. *inferior processes* e.g. `M-x shell` (these will be useful later for running R wihin Emacs.)
  4. Any text you care for ... `M-x animate-birthday-present`
- Switching between buffers from `Buffers` menu or `C-x b`
- Think of buffers as bits of text that can be edited/moved around.
- You can easily get 100+ buffers on a good day ... !

Windows and frames

- A *frame* can have one or more windows.
- An emacs process can create many frames (e.g. `M-x speedbar`, `C-x 5 2`)
- Each window has a *modeline* (describe).
- Each frame usually has an echo area, and a minibuffer. These normally overlap.

Major and minor modes

Modes contain specialisations (commands/variables) for different languages.

- Major modes are defined for different languages (C, latex, R, python, . . .).
- Consistency across modes where possible (e.g. commands for commenting, indentation). Keybindings consistent. *Font lock* also consistent, e.g. comments in red.
- Only one *major* mode active at a time (?Sweave?)
- Major mode decided typically on regexps on buffer name.

See `C-h v auto-mode-alist`

- Can have several *minor* modes, e.g. auto-fill-mode, flyspell-mode, font-lock-mode. Try `M-x tool-bar-mode`
- `C-h m` describes features available in current buffer.

Completion, defaults, history

- TAB completion where possible, e.g `M-x describe- TAB`
- sensible defaults, depending on text at point. e.g. `M-x man` with point on word *perl*
- context-dependent history, e.g. `M-x <up>` to see prev commands, or `M-x man RET <up>` to see previous man pages.

Getting help with Emacs

Keybinding	Function
C-h m	describe mode
C-h k	describe key
C-h i	info
C-h t	tutorial

Check out http://emacswiki.org

Some of my favourite Emacs things

- (show-paren-mode)
- find file at point: ffap
- iswitchb / ido
- VC: version control `C-x v l`
- ediff, e.g. `M-x ediff-revision`
- `M-x occur RET eval`
- `M-x grep`, `M-x compile` work similarly.
- AUCTeX + reftex
- org mode
- GNUS / MH-E / VM / RMAIL
- ESS !!!

Changing how emacs works

"Emacs is the extensible, customizable, self-documenting real-time display editor." (info "emacs")

- *Extensible* means that you can extend by writing elisp. Most of emacs is written in elisp, with C for core.
- *Customizable* means that you can change behaviour by changing values of variables.
- The .emacs file
- Reload by restarting emacs, `M-x eval-buffer` or `M-x eval-region`
- Or, after a closing bracket, just `C-x C-e`. Try twice e.g.:

```
(+ 5 (* 3 2) 4)
```

How does Emacs find its libraries?

- Emacs' `load-path` controls which directories are searched.

  ```
  (locate-library "ess")
  (add-to-list 'load-path "~/langs/emacs/org-mode/lisp")
  ```

Or to look up the definition of a function, try `M-x find-function ess-dirs`.

How do you find other packages?

- http://emacswiki.org mentions many, and has repository.
- Upcoming Emacs Lisp Package Archive (ELPA).
- http://www.damtp.cam.ac.uk/user/sje30/emacs/ell.html

Exercise: install Org mode 7.7

Although Emacs comes with org-mode, you probably should get a newer version.
Download, unpack, make and install.

```
wget http://orgmode.org/org-7.7.tar.gz
unpack,
cd org-7.7
make   ## not make install

Add to .emacs and eval.

(add-to-list 'load-path "~/org-7.7/lisp")
(add-to-list 'auto-mode-alist '("\\.org\\'" . org-mode))
(require 'org-install)

C-h v org-version
```

Exercise: make sure you have ESS 5.14 if you can.

Installing ESS is slightly simpler, especially if you want just the source. Download, unpack, and then:

```
    (load "~/langs/emacs/elisp-ds/ess/lisp/ess-site")
```

Aug 2011: Vincent Goulet's pre-made binaries for MAC and WIN are up to date.

Variables

Emacs variables influence the way functions behave.

```
(setq visibile-bell t)
```

- nil is FALSE, everything else, including t, is TRUE.
- `M-x customize`
- `M-x customize-group RET ess RET`
- `M-x customize-variable RET visible-bell RET`
- Finding new variables and functions: `M-x apropos RET occur RET`
- Hook variables . . .

Hook variables

- Hooks are run e.g. after major-mode has been created.

```
(add-hook 'emacs-lisp-mode-hook 'turn-on-eldoc-mode)
```

- Compare the following two versions:

```
## version 1
(defun my-ess-hook ()
  "Add my keybindings to ESS mode."
  (local-set-key (kbd "C-j") 'ess-eval-line-and-step))

(add-hook 'R-mode-hook 'my-ess-hook)

## version 2
(add-hook 'R-mode-hook
          '(lambda ()  (local-set-key (kbd "M-RET")
                                      'ess-R-use-this-dir)))
```

Read (and edit) the source

- `M-x find-function RET shell RET`

Emacs is extensible: write your own functions. e.g.

```
(defun date-stamp ()
  "Insert current date and time at point."
  (interactive)
  (insert (current-time-string)))
```

- Notes: 1. `interactive` makes it available to `M-x`. 2. Doc string accessible.

Byte compiler

- You will see .elc files; these are byte-compiled files.
- `M-x byte-compile-file`
- Should be faster than `.el` files, but can be confusing when developing code to have both .el .elc around.
- Note: `M-x find-function RET shell RET` will visit the *compressed* source file. Can edit .gz files as if they were uncompressed.

Philosophy

- Keeping files in plain text. Unicode: (view-hello-file)
- Keybindings should be consistent across modes.
- Keep one Emacs running (e.g. for many days).
- Where to draw the line about running everything in Emacs? mail? web browsing? `M-x artist-mode`

Outline

Why ESS?

Taken from the manual `(info "(ess)Features")`

- Command history, searchable + modifiable.
- Command-line completion of both object and file names.
- Hot-keys for quick entry of commonly-used commands in ‘S’.
- Transcript recording for a complete record of session.
- Interface to the help system.
- Object editing.
- Editing mode for code, providing syntactic indentation and highlighting.
- evaluating lines/regions/chunks of R code.
- loading and error-checking source files.

History of ESS (Tony)

- 3 branches: S-mode, SAS-mode, Stata-mode
- S-mode: 1989, 2 groups managing the project before (Bates/F Ritter/E Kademan, M Meyer, DM Smith). R added in 1995 (Rossini/Maechler)
- SAS: '94, Tom Cook ('92, John Sall, SAS). Integrated '95-6, Rossini
- Stata-mode: 1997, Thomas Lumley. Added 1997 (Rossini).
- 1997: last major restructuring ("grand unification")
- 2004: switch leaders: Maechler takes over
- We are still suffering from the 1997 grand unification.

ESS Installation

- http://ess.r-project.org/index.php?Section=download
- New users: grab pre-packaged bundle.
- For those who prefer a mac-like app: http://aquamacs.org/
- Debian package *ess*
- Available from Emacs Lisp Package Archive (ELPA) in near future
- Line in .emacs:

```
(load "~/langs/emacs/elisp-ds/ess/lisp/ess-site")
```

Editing and viewing R code

Indentation should be automatic; if in doubt `M-x indent-region`

- Imenu for browsing function definitons. (or use Speedbar)
- Demo: view example code in file:files/ms_funs.R
- The magic underscore. like it? If not, Try M-q _ (this works in general, e.g. M-q TAB). Or customize:

```
(ess-toggle-underscore nil)
```

- `C-c TAB` does object completion within .R buffers
- Moving around your functions:

Keybinding	Function
C-M-a	ess-beginning-of-function
C-M-e	ess-end-of-function
C-M-h	ess-mark-function

Making comments

- ESS follows the three-level comment model of elisp:
  1. One hash (#) for col 32 aligned end-of-line comments.
  2. Two hashes (##) to match current indentation.
  3. Three hashes (###) for column 0, major comments.

  See examples in `files/ms_funs.R`
- `comment-dwim` is bound to `M-;`.
- This 'do what I mean' should work well for cases 1 and 2.
- Or use `comment-region` (with or without prefix).

Indenting your code.

- By default, ESS uses a set of variables to set spacing.
- Following from `(info "(ess)Styles")`

  The combined value of nine variables that control indentation are collectively termed a style. ESS provides several styles covering the common styles of indentation: 'DEFAULT', 'OWN', 'GNU', 'BSD', 'K&R', 'C++', 'RRR', 'CLB'. The variable 'ess-style-alist' lists the value of each indentation variable per style.

- To reindent code: `M-x indent-region`

Viewing help files

e.g. `C-c C-v RET pbinom RET` (or ?pbinorm within `*R*`). Key features

- move easily between sections (n/p)
- browse help for new pages
- running examples (l)

Making help files (.Rd)

- Edit an exiting .Rd file: file:files/sjedrp.Rd
- Create a new file. file:files/empty.Rd then `C-c C-e`
- Preview your Rd files `C-c C-p`

Inferior ESS Processes (*R*)

- `M-x R` will start an R process within a new buffer.
- We can have multuple R processes, running either locally or remotely.
- Passing command-line switches to the R session.

```
  C-u M-x R RET --no-save RET
```

```
(setq inferior-R-args "--no-save ")
```

- These buffers are called *inferior* as they run a process under Emacs. Our ESS manual calls them *iESS* buffers.

Which version of R will it find?

- Should find all versions of R on your path, see the list found by

```
M-x R- TAB
```

- If you have a version of R on your path, try putting in a symlink somewhere on your path.

```
ln -s /my/version/of/R ~/bin/R-2.15.0-special
```

- Which versions of R are found?

```
ess-r-versions is a variable defined in ‘ess-r-d.el’.
Its value is ("R-1" "R-2" "R-devel" "R-patched")
```

- `M-x R-newest` will be bound to newest version of R on your system, by checking `R --version` for each R binary.
- (Windows might differ slightly)

What can I do in *R*?

- Run commands as normal.
- TAB Completion of functions, objects, filenames.
- At the prompt, recall history:

keybinding	function
M-p	comint-previous-input
M-n	comint-next-input
M-r	comint-previous-matching-input

- Can move through *R* and resissue commands from previous prompts

keybinding	function
RET	inferior-ess-send-input
C-c RET	comint-copy-old-input
M-RET	ess-transcript-send-command-and-move

What is emacsclient?

Why does `*R*` start with:

```
options(STERM='iESS', editor='emacsclient')
```

- emacsclient allows you to work with a running emacs, rather than start a new one.

```
M-x server-start
% emacsclient file.R
```

- Now in R, you can do things like

```
x <- rbinom(100, 10, 0.5)
edit(x)
fix(x)
page(x)
```

- `C-x #` should kill any client buffer.
- As an aside, `emacsclient -t` will bring up a new window.
- Warning: make sure you are using the right emacsclient if you have several versions on your system.

emacsclient / 2

- emacsclient allows you to run elisp

```
ess-slides.pdf: ess-slides.org
        emacsclient --eval "(org-to-pdf \"ess-slides.org\")"

(defun org-to-pdf (org)
  "Convert org buffer ORG to a pdf."
  (interactive)
  (save-current-buffer
    (set-buffer org)
    (org-export-as-pdf t)))
```

Transcripts

- If you save an `*R*` buffer, you create a transcript.
- You can keep using it.
- Clean up transcripts: `M-x ess-transcript-clean-buffer`
- Example `files/short_session.Rt`

How many versions of R can I run?

- `M-x R` multiple times will generate multiple processes, e.g. `*R*`, `*R:2*`, `...`
- You can tell your .R buffer to use a particular session by `C-c C-s` and be prompted (with completion!) for R process.

Remote R sessions

- login to remote computer via, e.g. by a ssh command in `M-x shell`
- Call `M-x ess-remote`, and select R.

Sending code from an R buffer to `*R*`

- An R source buffer must be associated with an iESS process. You can do this with, `C-c C-s` or accept default association.
- Loading source files with `C-c C-l`; errors can be parsed with `` C-c ` ``

Keybinding	Function
`C-c C-j`	`ess-eval-line`
`C-c M-j`	`ess-eval-line-and-go`
`C-c C-f`	`ess-eval-function`
`C-c M-f`	`ess-eval-function-and-go`
`C-c C-r`	`ess-eval-region`
`C-c M-r`	`ess-eval-region-and-go`
`C-c C-b`	`ess-eval-buffer`
`C-c M-b`	`ess-eval-buffer-and-go`
`C-c C-n`	`ess-eval-line-and-step`

- `C-c C-y` will switch to end of iESS buffer.

Sending code to *R* — is it slow?

If so, this is a long-standing bug. Add:

```
(setq ess-eval-visibly-p nil)
```

Narrowing

- You can temporarily narrow a buffer down to a particular region, and then `C-c C-b` works on that narrowed region:
- `M-x narrow-to-region C-x n n`
- `M-x narrow-to-defun C-x n d` works for R source buffers.
- `M-x widen C-x n w`

Customizing ESS

Recommended way is to do:

```
(customize-group 'ess)
```

or try editing your .emacs. e.g.

```
(setq ess-ask-for-ess-directory nil)
```

When things go wrong with ESS

- Read the info guides for sending bug reports

```
(info "(ess)Reporting Bugs")
```

- ess-help mailing list
- http://www.emacswiki.org/emacs/EmacsSpeaksStatistics

Outline

Literate Programming

From the web page describing his book *Literate Programming*, Donald E Knuth writes:
"Literate programming is a methodology that combines a programming language with a documentation language, thereby making programs more robust, more portable, more easily maintained, and arguably more fun to write than programs that are written only in a high-level language. The main idea is to treat a program as a piece of literature, addressed to human beings rather than to a computer. The program is also viewed as a hypertext document, rather like the World Wide Web. (Indeed, I used the word WEB for this purpose long before CERN grabbed it!) ... "
`http://www-cs-faculty.stanford.edu/~uno/lp.html`

Tangling and Weaving:

- CWEB: system for documenting C, C++, Java:

```
CTANGLE
    converts a source file foo.w to a compilable program file f
CWEAVE
    converts a source file foo.w to a prettily-printable and
    cross-indexed document file foo.tex.
```

`http://sunburn.stanford.edu/~knuth/cweb.html`

What is Reproducible Research (RR)?

- Gentleman et al (2004) advocate RR:

  Buckheit and Donoho (35) , referring to the work and philosophy of Claerbout, state the following principle: "An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and that complete set of instructions that generated the figures."

`http://genomebiology.com/2004/5/10/R80`

- Bioconductor packages are good examples of reproducible research.
- This article is also good background reader for open software development.

Sweave: literate programming for R

- Sweave is the system for mixing latex and R code in the same document.
- Used within R often to create "vignettes" which can be dynamically run.
- Allows you to write reports where results (tables,graphs) are automatically generated by your R code.

Sweave: including code chunks

- An example code chunk: by default we are in ‘LaTeX mode’.

```
We can then test the procedure a few
times, using the default number
of darts, 1000:

<<>>=
replicate(9, estimate.pi())
@
```

Sweave: including graphs

- Automatically creates filenames, e.g. `estimate-001.pdf`
- By default will generate .ps and .pdf; so change options:

```
\SweaveOpts{echo=T,pdf=T,eps=F,eval=T,keep.source=T}
```

```
\setkeys{Gin}{width=0.6\textwidth}
\begin{center}
<<fig=TRUE>>=
r <- 1; n <- 50; par(las=1)
plot(NA, xlim=c(-r,r), ylim=c(-r,r), asp=1, bty='n',
     xaxt='n', yaxt='n', xlab='', ylab='')
axis(1, at=c(-r,0,r)); axis(2, at=c(-r,0,r))
symbols(x=0, y=0, circles=r, inch=F, add=T)
...
rect(-r, -r, r, r, border='blue', lwd=2)
@
\end{center}
```

Sweave: including tables

- Use the *xtable* package from CRAN.
- Example from that package:

```
<<echo=FALSE>>=
library(xtable)
data(tli)
@

<<label=tab1,echo=FALSE,results=tex>>=
     ## Demonstrate data.frame
     tli.table <- xtable(tli[1:20,])
     digits(tli.table)[c(2,6)] <- 0
     print(tli.table)
@
```

Sweave: including inline computation

```
In this case the number of darts within
the circle is \Sexpr{d}, and so the estimated
value is $\pi \approx \Sexpr{4*d/n}$.
```

Sweave: a full example

- Example application: estimate the value of π using the

dartboard method.

- estimate.Rnw
- For nice ways to customize Sweave output (as in estimate.Rnw:

`http://www.stat.auckland.ac.nz/~stat782/downloads/Sweave-customisation.pdf`

- Compiling the document with make:

```
estimate.pdf: estimate.Rnw
        R CMD Sweave estimate.Rnw
        pdflatex estimate.tex
```

- Note: `R CMD Sweave --pdf` coming in 2.14.0

Emacs and Sweave: how to get n>1 major-mode?

- How can we switch between major modes within the same buffer?
- Approach: R-mode for code chunks, AUCTeX for latex sections.

Why AUCTeX?

- http://www.gnu.org/software/auctex/
- Excellent environment for (1) editing, (2) compiling and (3) viewing output.
- Shortcuts for inserting environments, blocks, and for completion.
- Interacts well with bibtex too.
- Reftex allows for easy editing/viewing of citations, table-of-contents. (Carsten Dominik).

Exercise: compile/edit a document in AUCTeX

- Warning: may need to install AUCTeX

See example file:coin/coin.tex and try the following

- edit it, compile it, fix errors.
- What do these commands do? `C-c [` `C-c &`
- Browse the TOC: `C-c =`
- I prefer to generate PDF (rather than postcript): `C-c C-t C-p`

Makefiles / compile modes.

- One approach to compiling your document is to use Makefiles.
- Emacs can run these with `M-x compile`
- Output shown in own buffer, which can be parsed.

Sweave editing help

- Typing < at start of line is *electric* and will expand to:

```
<<>>=
@
```

- Has many functions for navigating around and moving chunks. See the Noweb menu.

Sweave compilation

- 'M-n s' ('ess-swv-weave') Run Sweave on the current .Rnw file.
- 'M-n l' ('ess-swv-latex') Run LaTeX after Sweave'ing.
- 'M-n p' ('ess-swv-PS') Generate and display a postscript file after LaTeX'ing.
- 'M-n P' ('ess-swv-PDF') Generate and display a PDF file after LaTeX'ing. The command used to generate the PDF file is the first element of 'ess-swv-pdflatex-commands'.

Sweave + AUCTeX

This needs to be added in .emacs before place you load ESS, then restart Emacs. (**New in ESS 5.14.**)

```
(setq ess-swv-plug-into-AUCTeX-p t)
```

- Within an R buffer, `C-c C-c Sweave` runs Sweave in a separate process, not `*R*`.
- Compile latex file with `C-c C-c LaTeXSweave`.
- View resulting file with `C-c C-c View`

How it works in Emacs (advanced)

- multiple-major modes is quite fragile; more principled approaches welcome, e.g. based on Dave Love's work (indirect buffers).
- Current approach assumes that every time you switch in/out of a chunk, the major mode is reset; any local variable modifications must be set in hooks.

Outline

Prelim: make sure you have Org installed, see earlier section.

```
(require 'org-install)
```

My org mode setup

```
(add-to-list 'load-path "~/langs/emacs/elisp-ds/org-mode/lisp")
(add-to-list 'auto-mode-alist '("\\.org\\'" . org-mode))
(global-set-key "\C-cl" 'org-store-link)
(global-set-key "\C-ca" 'org-agenda)
(setq org-completion-use-iswitchb t)
(global-set-key "\C-cb" 'org-iswitchb)

;; stop C-c C-c within code blocks from querying
(setq org-confirm-babel-evaluate nil)

;; which languages do I want??
(org-babel-do-load-languages
 'org-babel-load-languages
 '((R . t)
   (sh . t)
   (emacs-lisp . t)
   ))
```

Live demo !

- Follow demo in file:org/intro.org
- simple outlining: folding/ moving/ promoting+demoting
- simple markup, e.g. **bold** *italics* underscore and `verbatim`
- editing lists
- look how math is formatted ... $\alpha + \beta$ without the need for math mode?
- tables; spreadsheets.
- export

Other key features of org mode (not covered here)

- agendas, dates.
- scheduler
- clocking tasks
- todo lists

Including code snippets

```
#+begin_src R :exports both :results graphics :file hist.png
  hist( rnorm(1e4))
#+end_src
```

These enviroments can be added using “Easy Templates”, e.g. `< s TAB` to get a source block.
These code snippets can be run using `C-c C-c` to generate results after the code snippet. see ob-R-examples.org

Exercise: Org examples to explore

1. file:org/nice/nice.org and export it to either latex –> pdf or html (`C-c C-e` for export menu)
   http://emacs-fu.blogspot.com/2011/04/nice-looking-pdfs-with-org-mode-and.html
2. file:org/drift/drift.org – Genetic drift. For browsing and exporting.
   http://orgmode.org/worg/org-contrib/babel/uses.html

1. ob-R-examples.org – Some simple examples in R, to run and export.
2. CISE – convert this to use R rather than gnuplot. Or try to get the baseball stats for year other than 2010.
   Eric Schulte, Dan Davison. Active Documents with Org-Mode Computing in Science & Engineering 2011.
   http://www.cs.unm.edu/ẽschulte/data/CISE-13-3-SciProg.pdf

Problems with org-mode for RR

1. Is it prime for RR yet?
2. Emacs 24 as stable base for documents . . .
3. Tom Dye has a nice reproducible solution for Org config. https://github.com/tsdye/hawaii-colonization
4. How do **you** think it compares with sweave?

Outline

Speedbar support for R code / imenu

- `M-x speedbar`
- Driven by the imenu support, which in turn depends on `ess-imenu-S-generic-expression` which you see has `<-` hard-coded.
- Show examples in `files/ms_funs.R`

Support for editing Roxygen documentation.

- See info node for full details: `(info "(ess)Roxygen")`

```
##’ Description of the function
##’
##’ Further details about this function
##’ @title A title
##’ @param me all parameters must be listed and documented
##’ @return Description of the return value
##’ @author The author
myfun <- function(me)
  cat("Hello", me, "\n")
@end verbatim
```

TAGS support

- TAGS files are supported for many different editing modes.
- They contain a list of where functions are defined in your source file.
- Normally found by regexps, but R now can generate tages directly:

```
R CMD rtags --help
```

- You can then ask Emacs to find definition of function name under point.
- `(info "(ess)TAGS")`

ESS-rdired

- Mode inspired by Dired for files.
- For viewing objects, deleting, plotting.
- `(info "(ess)Rdired")`

Rutils

- Recent addition to ESS, building upon ess-rdired. Adds many convenience functions.
- Browsing, adding, updating packages
- Editing objects
- Managing workspaces
- `(info "(ess)Rutils")`

Dynamically showing arguments of a function

Both of these methods work by calling the R function `args()` with function name under point. What differs is the feedback mechanism.

- ess-eldoc, works like other eldoc modes, e.g. elisp.

  Eldoc example

  ```
  (require 'ess-eldoc)
  (add-hook 'inferior-ess-mode-hook 'ess-use-eldoc)
  ```

- ess-r-args-show

  ```
  ;; bind ess-r-args-show to F2
  (define-key ess-mode-map [f2] 'ess-r-args-show)
  ```

r-autoyas

- http://www.svenhartenstein.de/Software/R-autoyas
- Dynamically creates *snippets* for editing R functions. Watch the movie online at:
  http://www.youtube.com/watch?v=jLJPorf2LBE

Debuggers

Two ESS-aware debuggers currently being developed:

1. *Ess-tracebug* "is a package for interactive visual debugging of R code from ESS. It provides visual debugging, browser, recover and conditional breakpoints; watch window and loggers; interactive debug/undebug of R functions and methods at point; highlighting and easy error navigation and interactive traceback." http://code.google.com/p/ess-tracebug/
2. The R package *edtdbg* "integrates R's debugging facilities with your text editor, featuring code tracking, variable display, quick breakpoint setting, and easy toggling of function debug status." It provides support for both Emacs and VIM. Package available from CRAN mirrors.

R Object Tooltips in ESS

- Watch the movie to see the tooltips for object at point: http://www.sigmafield.org/2009/10/01/r-object-tooltips-in-ess

Rewrite of interface to inferior processes (SLIME)

- ESS uses comint to interface with R processes.
- Better solution, based on sockets, rather than text-based channels, is to use SLIME framework.
- Work in progress (Christophe Rhodes): http://common-lisp.net/c̃rhodes/swankr/

Outline

Final discussion

- What would you like to ask?
- e.g. Can I do X with Emacs / ESS?
- What is the focus for future development of ESS?

Further reading

1. *Intro to emacs lisp* is a great introduction to learning elisp.
2. *Org mode* manual. Warning: it is big! You might prefer the 'compact' guide: http://orgmode.org/guide/index.html

Outline

My contact details

- Stephen Eglen
- sje30@cam.ac.uk
- Tel: 01223 765761
- Web: http://www.damtp.cam.ac.uk/user/sje30/