Emacs enhances data analysis and programming with R

ESS useR Short Course

A.J. Rossini

Modeling and Simulation
Novartis Pharma AG
Basel, Switzerland

14.06.2006 / useR-2006

Outline

1. Introduction (now, 15 minutes)
2. Using Emacs (45 minutes)
3. Using ESS (60 minutes)
4. Exercise 1: ESS
5. Exercise 2: Sweave
6. Emacs extensions (30 minutes)
7. Emacs Lisp (30 minutes)
8. Discussion and Misc Topics (related Emacs tools, ESS extensions, future designs)

#1: Efficient Program Editing

Edit, and let the computer repeat

- Weird archaic keystrokes are dominant
- Weird archaic keystrokes can make life easier
- Menus and toolbars help Emacs
- Knowing R and data analysis means we can enhance related activities.
- Folding editors, Object explorers

#2: Process Interaction

You enter text for the process, so edit. . .

- R vs. “emacs -f R”
- Getting Help
- Approaches for interacting
- Practical Directory layout

#3: Emacs is Extended and Extensible

Lisp and other hackers are your friends

- Introduction to Lisp and Emacs Lisp
- Lisp extends Emacs
- Design of ESS
- Future Extensions??

Disclaimer

- ESS (and Emacs) is many (different) things to different people.
- Even for R development, the variation in personal philosophies and approaches for use is extremely high (even in the Core development team).
- Religious issues focusing on Emacs should be discussed elsewhere (c.f. `http://www.dina.kvl.dk/~abraham/religion/`)

Demo!

It'll get boring before it gets better. . .

Please note or comment on any part that you'd like covered later.

Overview

- Emacs is the extensible, customizable, self-documenting real-time display editor.
- One of the oldest and yet still most powerful display editors
- The name **Emacs** comes from Editor (or Extensible) MACroS. (source of other amusing acronym expansions)
- Originally written in 1976 as a set of extensions to TECO (Text Editor and COrrector). It has since evolved.

Pros and Cons for Emacs

- Why use Emacs? It is a powerful text editor which can be extended to be a general interface to a computer (everything can be done within it).
- It is highly portable (in its own way) across many platforms.
- Why do not use it? It has a different user interface, which is understandable given its age (little change since 1985).
- It does not follow “modern” keybindings.
- People don’t understand Lisp (otherwise, what they want to do could be done in Emacs)

some people are allergic to it

On-line Resources

Links to links:

- Emacs Wiki: `http://www.emacswiki.org/`
- Emacs `http://www.gnu.org/software/emacs/`
- XEmacs `http://www.xemacs.org/`
- Emacs Lisp Library `http://www.damtp.cam.ac.uk/user/sje30/emacs/ell.html`

The Emacsen Problem

different religions cause stress

There are many variants or dialects of Emacs (*similar to the S language having AT&T's S, Insightful's S-PLUS, and R-core's R*)

- Emacs: "classic version"
- XEmacs: "experiments" (GUIs, dynamic loading, packaging)
- SXEmacs: "splinter" (Unix-focus, streamlining XEmacs)

ESS support is best in order (Emacs, XEmacs, SXEmacs).
Most active ESS developers use Emacs. Users seem to be split 50/50 between Emacs and XEmacs.
No bug reports on SXEmacs...

Installation
the bane of Emacs

Installation can be tricky:

- Linux: Emacs usually can be installed or is preinstalled
- Microsoft: XEmacs is easier to install, Emacs might be better supported.
- Mac OSX: Emacs, AquaEmacs; via Fink, or UCLA package, or. . . (check on the list).

We aren't dealing with this here!

Editing methodology
Emacs resembles modern editors unlike dual-mode VI

- Idea: every keystroke (or sequence of keystrokes) generates a list to evaluate (i.e. command, or lisp function with arguments).
- Pressing “X” evaluates the list (self-insert-command "X") (we’ll talk about Lisp later)
- Modifiers extend the keyboard (CTL, SHIFT, ALT, pressed simultaneously, some such as ESC, META, and others are pressed first, released, and then followed by a key).

Overview
ESS augments Emacs for R

- Emacs Speaks Statistics (or Emacs Statistical System)
- Oldest active statistically-focused OSS project (1989).
- Design goal: a consistent interface unifying batch and interactive data analysis styles.
- Novelty (at the time): process control, unified interface.
- String (stream) operations; “cheap parser”.
- Supports R, S-PLUS, SAS, Stata, LispStat
- Current development team (n=6) spans the US and Europe.

History

- 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 (now Insightful)). 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.

Interactive Programming

Everything goes back to being Lisp

- Interactive programming (as originating with Lisp): works extremely well for data analysis (Lisp being the original “programming with data” language).
- Theories/methods for how to do this are reflected in styles for using ESS.

Good Statistical Analysis is on-line Interactive Programming

Installation

We have to deal with this here, though. . .

- Fetch the ESS package (tar or zip-file)
- unpack
- add a line in **.emacs** to load the **ess-site.el** file.
- restart Emacs (or **M-x load-file /path/to/ess-site.el**).

Review: Introduction

- ESS is a program in Lisp to extend Emacs
- Characteristics: OSS, developed initially to make life easier; continued development supported research in enhanced data analysis interfaces.
- Emacs has strengths and weaknesses; these are usually different than percieved strengths and weaknesses.

Goals

- use of Emacs to edit files
- filesystem interaction

Starting (X)Emacs
trivial but essential

emacs can be used instead of xemacs.

```
xemacs <filename>   run xemacs, edit <filename>
xemacs -nw          run xemacs in _text_ mode
                    (nw="No Window"; useful for
                     slow remote connections)
```

- Once started, no need for more.
- Only run one Emacs session. This is a cardinal rule, do not violate it. (c.f. file-loading section)

Breaking and Quitting!

essential knowledge

- Decide you were wrong while Emacs works: `C-g` (universal break)
- Quit and terminate: `C-x C-c` (will usually prompt you for any files which are modified but not saved)

Understanding Keystrokes.

- `[tab]` is the TAB (indent) key.
- `[return]` is the Return (carriage return, enter) key.
- **C-h** means press control key AND "h" key at same time.
- **ESC-h** means press ESC key THEN "h"
- **M-h** means press ALT key AND "h"
- **M-C-h** means press Meta or Alt while pressing h. 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.

Stopping Activity
you need to know this. . .

C-g stops the current Emacs activity to the best of its ability.

Windows and Frames

- Window: a different view on (usually) a different buffer within the same application
- Frame: a different window/application in the windowing system.

File suffices drive mode setup

Use the right file name formats:

- `.S`, such as `critical_simulation.S`
- `.R`, such as `speedy_simulation.R`

There are other ways to force a file to be edited in a particular mode, but at this point, keep it simple.

Commands

Emacs has a single command-entry mode. There are 2 approaches for commands:

- **M-x (command-name)**
- keystrokes/toolbars/menus (which implement the above)

Learning the Strokes.

- forced practice: painful but fast. Keep a copy of the reference cards nearby! (usually 2-3 days of severe pain, followed by bliss).
- accelerators and menus can help: but they quickly become a crutch.
- Consider working through the tutorial (**C-h t**).

this is a major issue for speed typists converting to/from Emacs usage.

Loading Files

- `C-x C-f` (load new file, **not** deleting old)
- `C-x 4 C-f` (load new file in different window)
- `C-x 5 C-f` (load new file in different frame)

(c.f. basics)

Changing Active Buffer (edit file)

To edit (operate on) another file:

- `C-x b` (switch to new buffer in same window)
- `C-x 4 b` (replaces current file in different window)
- `C-x 5 b` (replaces current file in different frame)

Buffers represent the files you are editing (explain).

Review of Common Keystrokes

Demo and pain

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
```

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 0	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
```

Point and Mark
critical concepts to learn

- Point: where the cursor is.
- Mark: a location that you've decided (actively or inactively) is important.
- Example: Move cursor somewhere. **M-x set-mark** or **C-(space)**.
- Exchange point and mark: **M-x exchange-point-and-mark** or **C-x C-x**.

Review
Emacs

- **C-g** stops emacs activity and returns to edit mode; **C-x C-c** quits
- Try to use the keyboard and not the mouse, menus, or toolbars.
- Keypresses are simply commands.
- Biggest user headache? Differenting commands and text entry
- Buffers represent the files you are editing. Morally, they are the files themselves (with important exceptions).

Overview: ESS

This section should provide an overview for

- editing
- interaction with R
- facilitating help
- reusing transcripts
- Sweave support

and finally, putting it all together

Coding Style
Indentation makes code readable

Comment conventions:

1. # gets moved to the right-side of the line
2. ## is placed at the current indentation level
3. ### is placed flush-left.

Indentation Example

```
    my.x <- rnorm(10)
### This is flush
    ## but this is at level
                                        # this is right
```

Editing, ESS adds. . .

- Automatic fontification, indentation

If R is running in another buffer:

- evaluation of lines (C-c C-n), regions (C-c C-r), functions (C-c C-f), and buffers (C-c C-b)
- help output in other buffers (and via command, C-c C-v)
- Object (data, function) completion
  - `C-c [tab]`
  - `[tab]` (sometimes).
- Filename completion: M-

tab

R editing buffer

Usually named for the file being edited

(mode line says "ESS[S]")

```
C-c C-j         Send line to R
C-c C-n         Send line to R
                and go to _n_ext line
C-c C-r         Send highlighted _r_egion to R
C-c C-b         Send whole _b_uffer to R
C-c C-f         Send _f_unction surrounding
                cursor to R
C-c C-v xxx     get R help on "xxx"
```

Run R

- Within emacs
- At the command line

R process within Emacs (on start)

starting from the command line or launcher

Replace the command **R** with one of:

`emacs -f R`	run (x)emacs, starting R
`xemacs -f R`	session

adds a few nice features to the command interface. (follow with **&** to background the process; `xemacs` might also work)

R process within Emacs (running emacs)

Starting from a running emacs

```
M-x R          Start R process buffer
M-x R-mode     Change mode of current buffer
               for R code editing
```

Evaluate file contents
working from the editing (ESS) buffer

.
From a file of commands:

- Send current line to R: `C-c C-j`
- Send current function to R (assumes the cursor is in the function body): `C-c C-f`.
- Send current region (highlighted) to R `C-c C-r`
- Send whole buffer/file to R `C-c C-b`

Object and File completion

Completion of objects (functions and data):

1. `C-c [tab]`
2. `[tab]` (sometimes).

Dumping Objects
Getting objects back

(for modification, saving, eventual reloading):

1. **C-c C-d** dumps the named object into an editing buffer
2. Examples: today's date for use as a log file; (140606), "lm".
3. if object doesn't exist: create a blank buffer
4. if object exists: insert the object's textual representation (no comments)
5. The file will be stored in /**tmp/username.objectname.R** (unless you want it elsewhere or named something else).

Type commands into the edit file, evaluate them when desired, edit as needed for correctness (and comment mistakes, with reasons for them, for later reminders and documentation).

Commandline History

ESS remembers what you've done

Search command-line history "manually", matching on current input

1. backwards: `M-p`
2. forwards: `M-n`

Complete current line based on command-line history:

1. backwards: `\C-[uparrow]-p`
2. forwards: `M-n`

If you are reviewing old commands, to re-enter it (i.e. cursor on that line) `[return]`.

Philosophy: Files are “truth”

Don’t believe your .RData

- This is a generally decent approach.
- Saving objects in binary format in special files can be a reasonable strategy
- Work from an R file, *submitting* commands to the R process. This saves the source, and allows for structuring of the input, along with having a transcript of the order of commands to understand reproducibility aspects.

Working from R process
from the process (inferior ESS, or iESS) buffer

not usually recommended

1. One improvement is to use a `source()`-like facility. To source a file into R (within ESS): **C-c C-l** (load file)
2. To find what the errors are (if any): **C-c ‘** (backquote).

The latter will tell you what the error is. You'll have to go back to the file (**C-x C-b** or use *Buffers* menu item to bring in back to the screen) and correct it.

Working from R buffer
from the editing (ESS) buffer

Better yet, use a file, and send material from the file straight to the process. This works if the file ends in the proper suffix (and hence, the mode at the bottom should say “ESS”). The following is possible:

- Send current line to R: **C-c C-j**
- Send current function to R (assumes the cursor is in the function body): **C-c C-f**
- Send current region (highlighted) to R: **C-c C-r**
- Send whole buffer/file to R: **C-c C-b**

Philosophy: RData is true

Editing objects without source

- This is a dangerous approach; corruption of the RData file is more difficult to fix using an editor or other external tools than corruption of text files, usually.
- Trusting .RData to have what you've done is rather suboptimal strategy (quite prone to user mistakes)

If you must use a command line. . .

- This style is useful for quick and dirty exercises/testing.
- Transcripts can be cleaned and edited to record the salient points of the sesssion.

Begin ESS/Emacs/R

1. Start XEmacs (in a terminal window, enter **xemacs &**)
2. Start R (**M-x R**).
3. Load a file (**C-x C-f** *filename*)

Data Analysis Example

Demo.

Function Editing

1. demo.R: evaluate.
2. Try editing existing functions: **my.test.function** and **that**, by **C-c C-d this**. Remember that you should be able to complete the name using *ajr.t***SPACE** (i.e. Control-C, then SPACE key). Generally, completion of variables and functions can be done using **C-c [tab]** in the editing buffer, and **SPACE** in the command entry area (at the very bottom of the Emacs screen/window).

Navigating Help Buffers

- To get help on a function (help(“lm”)) without spoiling your input line: **C-c C-v**
- Navigation of sections: “n” next section, “p” prev section.
- evaluation of example: “l” (eval line) “r” eval region.
- searching: “/” isearch forward
- kill-buffer and go *back*: “x”

See menu items

Using Transcripts
for teaching, reproduction, and audits

- Transcript suffix: .Rt
- Stepping through entries:
- Cleaning transcripts: **M-x ess-transcript-clean-buffer**
- Cleaning transcripts: **M-x ess-transcript-DO-clean-region** (even when read-only toggled)

Using Multiple Processes

For when you want to multitask

ESS supports multiple processes. This is useful for

- testing on multiple versions of R
- testing on R and S-PLUS (and S4, if one still has a copy)
- inefficient multiprocessing

Using Remote Processes
loading other people's computers

Since we are capturing/using R text I/O, local or remote is not an issue (running R in an Emacs shell, logged in (shell account via telnet, ssh, rlogin) to another machine)

- **M-x ESS-remote**
- follow instructions

Customizing ESS

See (commented out) example customizations in ess-site.el (assuming that the lisp files are installed:

```
M-x locate-library [return] ess-site [return]
```

will help. Also,

```
M-x customize-group [return] ess
```

helps to review possible customizations (this is an alternative to editing the ~/**.emacs** file for setting customizing variables).

Overview: Sweave

Sweave originated from the Literate Data Analysis philosophy which in itself was the idea of applying Literate Programming to the specialized programming discipline *interactive data analysis*.
Support for editing Noweb files existed within ESS 4 years before Sweave, but Sweave is a “Killer App”.

Documents as Programs

- Every electronic document is a “computer program” (sometimes at a primitive level) for some rendering device or program (such as: Microsoft Word; Postscript printers; Adobe Acrobat Reader; Emacs)
- This is the general theory behind the WWW, as well as the so-called *Semantic Web*.

Literate Programming

A programming methodology devised by Donald Knuth in the late 70s, early 80s, which mixes computer programs doing mathematics with mathematical documentation (formulas, theorems, algorithms).
2 operations:

- Tangling: translating document into undocumented computer source code for compiling.
- Weaving: translating document into documentation for the source code.

The original tool was called Web (predates the *World-Wide-Web*).

Literate Statistical Practice
or *Literate Data Analysis:*

The application of literate programming to statistics; recognizing that statistical data analysis is a specialized programming activity with different characteristics than systems programming or applications programming.
(realization prototype: ESS-Noweb and Sweave).

ESS enhances Sweave

- context specific modes (doc: latex, html, muse; code: R, etc).
- chunk and thread evaluation.
- functions for processing Sweave files.

Evaluation
Chunks and Threads

- eval-chunk (run code inside a code chunk)
- eval-thread (run code inside a series of code chunks)

Accelerated Sweave Development

- ESS has specialized support
- use Makefiles (editor-neutral approach)
- **M-x ess-makeSweave** or **M-n s**
- **M-x ess-makeLatex** or **M-n l**
- **M-x ess-makePS** or **M-n p**
- **M-x ess-makePDF** or **M-n P**
- **M-x ess-insert-Sexpr** or **M-n x**

Review: Using ESS

- ESS for R has 4 modes: editing, interactive, transcript, help.
- Focus on learning to use keystrokes rather than mouse.

Packages and Extensions.

Emacs has many modes:

- language-specific markup (programming, markup)
- editor behavior (spell checking, pending delete, folding, editor emulation (vi, brief))
- IDE capabilities (object explorers, completers)
- database (edb)
- communication (mail, news, www, rdf)
- sub (inferior) process control (telnet, R, . . .)
- publication (text mode conversion)
- version control interfaces
- anything you can program!

Relevant to Data analysis and R

- folding modes (outline; however, the newer version, allout, stomps on R-mode)
- object explorers (ecb)
- documentation modes
  - specialized: latex, html, rtf
  - generic: muse, . . . (idea: simple markup which translates to html, latex, texinfo, etc).

Dired

A directory/file explorer and navigator.

Emacs Code Browser

- ESS integrates to support R
- Extension/customization: through imenu support.
- file-level navigation and objects, editing history of files, files in current directory, directory tree.

(demo with R-examples/FCS.R)

Muse
a mode for documentation and notes

- Wiki-like editing, hyperlinks, within text.
- simple format
- Supports generation of latex, pdf, (x)html, xml.

Planner: Task management
getting organized and tracking activity

- Plain-text task management and organization
- Supports many common time-management approaches (GTD,).
- Extremely customizeable

LaTeX Support
nearly WYSIWYG

- AUC-TeX: latex document support
- BiBTeX-mode and RefTeX: citation management
- X-Symbol: native display of mathematical symbols
- tex-preview: You can see what you will get (YCSWYWG)

Customizing Emacs
Sometimes with Lisp

- Critical components for customizing Emacs: current best practices suggest that you customize via “M-x customize”. (this is a recent (last 5 years) change!).
- However, adding Lisp commands to your **.emacs** file is still very useful (and sometimes required).
- Note that commas denote comments (from the comma until the end of the line).

Lisp

Lisp (LISt Processor) is different than most high-level computing languages, and similar to Emacs, it is very old (1956). Lisp is built on lists of things which might be evaluated:

```
(function data1 data2 data3)
```

or “quoted”:

```
'(function data1 data2 data3)
```

The difference is important – lists of data (the second) are not functions applied to data (the first).

Evaluation

Lisp is built on lists of things which might be evaluated:

```
(function data1 data2 data3)
```

or “quoted”:

```
'(function data1 data2 data3)
(list function data1 data2 data3)
```

The difference is important – lists of data (the second) are not functions applied to data (the first).

Defining Variables
Setting variables

```
(setq <variable> <value>)
```

Example:

```
(setq ess-source-directory
      "/home/rossini/R-src")
```

Defining on the fly

```
(setq ess-source-directory
   (lambda () (file-name-as-directory
         (expand-file-name
           (concat (default-directory)
                   ess-suffix "-src")))))
```

(Lambda-expressions are anonymous functions, i.e. “instant-functions”)

Function Reuse

By naming the function, we could make the previous example reusable (if possible):

```
(defun my-src-directory ()
      (file-name-as-directory
         (expand-file-name
           (concat (default-directory)
                   ess-suffix "-src"))))
```

Example:

```
(setq ess-source-directory (my-src-directory))
```

Programming Emacs Lisp

There are three general approaches to testing, evaluating, and deploying Emacs Lisp code.

- Loading files: `M-x load-file <filename.el>`
- Using the `*scratch*` buffer
- `ielm`, an emacs-lisp REPL

Review the latter two

Playing in the scratch buffer.

1. `C-x b *scratch*`
2. (display-time)`C-j`
3. (setq my-test “asdf”)`C-j`
4. my-test`C-j`

ielm: command-line usage.

1. M-x ielm
2. follow the instructions for more information (similar to using R in interactive mode)

Organization.
Structure of ESS

- How ESS works
- Reading the code
- making extensions

Debugging.

- Set options to “debug on error” and “debug on signal”. Both bring up the stack-trace of the current status of the lisp program (nesting and variable values).
- Look for errors in loading to determine the path taken to the error
- more sophisticated debuggers exist.

Idea generation: SLIME
Lisp IDE suggests ESS improvements

- Use R subprocesses (old idea of DTL).
- Synchronize processes

Future?

- Continued support for R within ESS
- Object and formating support

Summary

- Emacs takes time to learn, but rewards the user.
- ESS enhances R.
- Lisp is worth learning.

- Outlook
  - ESS is the oldest open-source statistics project; yet it can always be improved!
  - Many possible extensions for supporting work modes for Statisticians and Data Analysts. More need implementation.