

Emacs Speaks Statistics: A Universal Interface for Statistical Analysis

A.J. Rossini* Martin Mächler† Kurt Hornik‡
Richard M. Heiberger§ Rodney Sparapani¶

January 16, 2001

Keywords: Statistical Analysis, Programming, User Interfaces

Abstract

We discuss Emacs Speaks Statistics (ESS), a user interface for statistical programming based on Emacs, intended for many statistical programming languages. It falls in the programming tools category of Integrated Development Environments (IDEs). We discuss how it works, why one would consider using it, and extensions which increase the programming efficiency for statistical programming.

1 Introduction

Integrated Development Environments (IDEs) combine features and tools in a single interface in the attempt to increase programmer productivity and efficiency. The increased speed in software development time over the last decade can be partially attributed to the use of IDEs and similar Rapid Application Development (RAD) tools and methodologies. In the field of statistics, programming is an important skill which can be augmented and enhanced by the right tools and environment. This is especially important with the rise of computational tools such as resampling methods such as the bootstrap and jackknife, as well as simulation methods such as Markov chain monte carlo (MCMC).

One issue that quickly arises is that different tools have different strengths, and optimal use can require switching back and forth between tools for data analysis. For example, general purpose tools such as R (Ihaka and Gentleman, 1996) and S-PLUS are limited in scope for Bayesian analyses, which tools such as BUGS are useful for. From another point of view, specialized tools such as BUGS are not as easy to construct more

*Department of Biostatistics, University of Washington and Department of Biostatistics, Fred Hutchinson Cancer Research Center, Seattle, WA, USA

†Seminar for Statistics; ETH Zurich; Zurich, Switzerland

‡Technical University of Vienna; Vienna, Austria

§Temple University; Philadelphia, PA, USA

¶Department of Statistics, Duke University; Durham, NC, USA, Medical College of Wisconsin, WI

general analyses and graphics as S-PLUS and R. This leads to an efficiency gain when using multiple tools for both theoretical and applied statistical research.

The goal for the Emacs Speaks Statistics (ESS) project is to provide a single point of interface for text-based statistical computing. The primary tasks that will be considered are statistical coding and interactive data analysis. Statistical coding is the writing of computer code for data analysis. This code might be compiled into an object file using a compiler, or might be run by an interpreter. The task of entering of commands for interactive data analysis is similar. The primary difference is that the results of a small set of commands are of critical interest for review in the latter, but the results of all commands are of interest in the former. Both of these tasks can be present at the same time, for example in the use of compiled Fortran code for optimization, which is being called from an interpreted language, such as S-PLUS, R, or XLispStat, containing the objective function to optimize.

One natural issue that follows is that not all of the languages or statistical packages have similar interfaces. This means that it might be necessary for a statistical analyst to learn 2 or 3 different text-interfaces for editing and executing code. Furthermore, it might be necessary for the analyst to consider dialects of a language, for example under the S family of languages, there is R, S-PLUS, the original S implementation, as well as different versions of all three. Testing code meant for use by others necessitates having it work in multiple versions, so being able to quickly test means having quick access to all versions; this might mean having to be able to access programs which can only run on completely different sets of machines. For example, one might want to be connected to multiple R processes. Reasons for this include verifying behavior on different versions of the same software, test and run scenarios where one process is doing long-term processing while the other is doing short-term testing, simulations, running multiple processes on multiple machines from the same place.

ESS provides an interface which mitigates some of the problems noted above. It provides an editing interface, an interface to statistical processes, and additional tools which can be useful for both statistical software development and data analysis. It works with common statistical software including the S family of languages including S (Becker et al., 1988; Chambers and Hastie, 1992; Chambers, 1998), S-PLUS **CITE MathSoft**, and R (Ihaka and Gentleman, 1996); XLispStat (Tierney, 1990) including the Arc **CITE Cook and Weisberg** and ViSta (Young et al., 1992) extensions; SAS **CITE SAS Institute**; Stata **CITE: Stata Corporation**; Omegahat **CITE: Omega Project**; and can be extended in at least a limited way to most statistical packages which can be controlled from a command-line.

In the rest of this introduction, we will discuss the environment of interest and how it can be used. Sections 3 and 4 will discuss the implementation, features, and use with S family of languages and SAS, respectively. The history and advanced features will be then discussed, and this paper will conclude with a discussion of future work in this area, including Literate Statistical Analysis as well as important features.

1.1 Emacs

Emacs is a mature, powerful, and extensible text editor which has been freely available for over 15 years and is available for a large number of platforms, including those

currently most frequently in use for statistical computing, including Unix, Apple Corporation's Macintosh, and Microsoft OS platforms. While it does not fall into the realm of word processors, relying on external programs for that, it does have similar features.

Most programming and documentation falls under the realm of text processing. This is natural given that it is being worked on from the perspective of the developer and not the end user. This work can be enhanced by common IDE features such as contextual highlighting and recognition of special reserved words for the programming language or statistical program/package of interest.

The main point is that with the ESS extensions, Emacs acts as a uniform interface to the statistical programming code, the statistical program/process, documentation tools (L^AT_EX, SGML, XML, Noweb), and revision control tools (RCS, CVS, SCCS, PRCS), on both local and remote machines. It handles the interface to both source code and transcripts using context (syntax highlighting, bookmarking features, command-history).

In addition, Emacs is a sophisticated development environment for programming in traditional compiled languages such as C and Fortran as well as non-traditional byte-compiled and interpreted languages such as Perl, Java, and Python. It provides interfaces to build tools such as make as well as debugging tools through the use of external debuggers such as JDebug, GDB, and DBX.

Extensions to Emacs allow it to act as a World-Wide-Web browser, a highly sophisticated mail and news reader, a shell/terminal window with history, and as many other common text-based tools. It can be re-mapped to act as many other text-editors, such as ed, vi, wordstar, and brief. In addition, the above plus its ability to interface with other processes makes it an ideal platform for providing a universal interface to statistical packages. The use of emacs-lisp as an extension language has also facilitated the programming.

Emacs is equipped to assist with programming in Fortran, C, C++, Java, and many more languages. It has an interface to build and debugging tools including make and many common debuggers such as dbx, gdb, and jdb.

Along with this, Emacs provides an interface to revision control systems for maintaining version history and a change-log for files.

Because of the above, the choice of Emacs as the basis for a universal interface is a natural one.

AJR: Need an Introduction to EMACS

We assume that you are familiar with Emacs terminology and syntax: file, buffer, region, description of keys etc. If not, please read the New Users guide (found in the info pages, "C-h i" (by pressing control h, i) or Tutorial, "C-h t").

To find the key-sequences for commands, view the keymap (C-h b) or view help for the current mode (C-h m).

One of the strengths of Emacs is that every activity can be facilitated by keystrokes. This has the advantage of eliminating the slowness and potential danger of mouse usage, but also has the disadvantage of requiring memory of location and behavior.

For small projects, it can be useful to have a single repository (file or directory with a few files) for documenting code which produces datasets and analysis results. This is facilitated by being able to edit files (different or the same) and being able to specify different modes for handling programming code for the different statistical languages.

1.2 Other Statistical User Interfaces

For the purposes of the user interface, recent and traditional interfaces for statistical packages and languages can generally be classified into 3 forms. There is the command-line interface that most of the packages have available; this is the interface that ESS needs for interfacing at the process level. There is also the spreadsheet/MDI interface employed by both spreadsheet packages as well as most Apple and Microsoft-based statistical packages. In addition, there have been other one-time implementation interfaces, which include graph-based interfaces as implemented in ViSta (Young and Lubinsky, 1995), the SAS terminal interface, which divides the terminal window into 3 screens, and possibly others.

1.3 ESS

AJR: History of ESS.

ESS grew out of the programming and process interface of the Emacs S-mode (D. Bates, et.al; D. Smith). The extension to a language-independent generic interface was prompted by the success of R, and the need for an R-mode. This led to a merger with the SAS-mode (T. Cook), and the refactoring of the S-mode codebase to accommodate multiple languages in a flexible way. A detailed history can be found in section 7.

AJR: Current State of ESS

ESS currently supports a number of statistical languages, with various levels of support depending on their capabilities and needs. Because of its history, ESS supports the S family of languages extremely well; these include recent versions of S, S-PLUS, and R. SAS is also well supported, but to a lesser extent. The lack of objects in SAS has prevented the use of object completion facilities. Stata and XLispStat (and the XLispStat extensions, ARC and ViSta) are marginally supported, providing mainly syntax highlighting and process-interfacing. Note that even though we refer to it as marginal support, it is still the basic functionality that the majority of ESS users take advantage of. This appears to be sufficient to differentiate from typical package-provided interfaces.

2 Features and Usage

ESS provides a number of features for statistical programming, beyond those generally provided by Emacs, which were listed in section 1.1.

For the interface with the programming language code as well as the interface with the statistical program, this includes syntax highlighting to denote assignment, reserved words, presence of strings. For the programming code, to assist with providing a clear presentation, customizable automatic indentation is possible, with the customization relating to how one would want to indent groups of expression. For the interface with the statistics package/program, there is also a means of searching the command-line history for previous commands as well as editing past commands for current use. These are connected by the ability to send, within Emacs, lines, functions, regions, and whole edited buffers from the programming code to the statistical program/package for exe-

cution. In addition, it is possible to have Emacs complete file names from the current underlying working directory.

In addition to the above, for languages in the S family, including S (versions 3 and 4, developed at Bell Labs), S-PLUS(all versions), and R (all versions), there is object-name completion of both user and system based functions and data. There is also the ability to dump and save objects (user and system generated) into text files in a formatted manner for editing, as well as to reload after possible editing back into the statistical package/process.

Editing source code (S, LispStat, SAS)

- Loading and error-checking of code

Interacting with the process (S, LispStat, SAS)

- Command-line editing
- Searchable Command history
- Command-line completion of S object names and file names
- Quick access to object lists and search lists
- Transcript recording
- Interface to the help system

Transcript manipulation (S3, S+3, S4, R, XLispStat)

- Recording and saving transcript files
- Manipulating and editing saved transcripts
- Re-evaluating commands from transcript files

ESS also provides an interface for writing help files for R functions and packages. It provides the ability to preview, as well as handling embedded R source code in the same manner as ESS normally handles source code, including syntax highlighting as well as the ability to submit code to a running ESS process, for example R or S-PLUS.

3 Using ESS with the S family of languages

3.1 Editing Files

ESS[S] is the mode for editing S language files. This mode handles:

- proper indenting, generated by both [Tab] and [Return].
- color and font choices based on syntax.
- ability to send the contents of an entire buffer, a highlighted region, an S function, or a single line to an inferior S process, if one is currently running.
- ability to switch between processes which would be the target of the buffer (for the above).

- The ability to request help from an S process for variables and functions, and to have the results sent into a separate buffer.

- completion of object names and file names.

ESS[S] mode should be automatically turned on when loading a file with the suffixes found in ess-site (*.R, *.S, *.s, etc). However, one will have to start up an inferior process to take advantage of the interactive features.

3.2 Inferior ESS processes

iESS (inferior ESS) is the mode for interfacing with active statistical processes (programs). This mode handles:

- proper indenting, generated by both [Tab] and [Return].

- color and font highlighting based on syntax.

- ability to resubmit the contents of a multi-line command to the executing process with a single keystroke [RET].

- The ability to request help from the current process for variables and functions, and to have the results sent into a separate buffer.

- completion of object names and file names.

- interactive history mechanism

- transcript recording and editing

To start up iESS mode, use one of

M-x S+3

M-x S4

M-x R

(for S-PLUS 3.x, S4, and R, respectively. This assumes that you have access to each). Usually the site will have defined one of these programs (by default S+3) to the simpler name:

M-x S

Note that R has some extremely useful command line arguments, -v and -n. To enter these, call R using a "prefix argument", by

C-u M-x R

and when ESS prompts for "Starting Args ? ", enter (for example):

-v 10000 -n 5000

Then that R process will be started up using "R -v 10000 -n 5000".

New for ESS 5.1.2 (and later): "S-elsewhere" command

The idea of "M-x S-elsewhere" is that we open a telnet (or rlogin) to another machine, call the buffer "*S-elsewhere*", and then run S on the other machine in that buffer. We do that by defining "sh" as the inferior-S-elsewhere-program-name. Emacs sets it up in a "*S-elsewhere*" iESS buffer. The user does a telnet or login from that buffer to the other machine and then starts S on the other machine. The usual C-c C-n commands from myfile.s on the local machine get sent through the buffer "*S-elsewhere*" to be executed by S on the other machine.

3.3 Handling and Reusing Transcripts

- edit transcript
- color and font highlighting based on syntax.
- resubmit multi-line commands to an active process buffer
- The ability to request help from an S process for variables and functions, and to have the results sent into a separate buffer.
- ability to switch between processes which would be the target of the buffer (for the above).

3.4 Programming Language Help

When viewing help pages, there's a new menu [ESS help] and useful 1-letter shortcuts for navigation and more, such as

n,p move to **n**ext or **p**revious help section

s x move to help section code “x” where x is e.g. u for “Usage”, or e for “Examples”.
You can see a full list

h fast ‘hyperlink’ to other help pages

l (after “s e”) send examples **l**inewise to S for evaluation, also **C-c C-r** for sending a whole region. This is particularly useful in R which guarantees that all examples are directly executable (possibly after `library(*)` for non-base packages).

3.5 Philosophies for using ESS

The first is preferred, and configured for. The second one can be retrieved again, by changing emacs variables.

1. (preferred by the current group of developers): The source code is real. The objects are realizations of the source code. Source for **every** user modified object is placed in a particular directory or directories, for later editing and retrieval.
2. (older version): S objects are real. Source code is a temporary realization of the objects. Dumped buffers should not be saved. We strongly discourage this approach. However, if you insist, add the following lines to your .emacs file:

```
(setq ess-keep-dump-files 'nil)
(setq ess-delete-dump-files t)
(setq ess-mode-silently-save nil)
```

The second saves a small amount of disk space. The first allows for better portability as well as external version control for code.

3.6 Scenarios for use

We present some basic suggestions for using ESS to interact with S. These are just a subset of approaches, many better approaches are possible. Contributions of examples of how you work with ESS are appreciated (especially since it helps us determine priorities on future enhancements)! (comments as to what should be happening are prefixed by "##").

- Data Analysis Example (source code is real)

```
## Load the file you want to work with
C-x C-f myfile.s

## Edit as appropriate, and then start up S-PLUS~3.x
M-x S+3

## A new buffer *S+3:1* will appear. Splus will have been started
## in this buffer. The buffer is in iESS [S+3:1] mode.

## Split the screen and go back to the file editing buffer.
C-x 2 C-x b myfile.s

## Send regions, lines, or the entire file contents to S-PLUS.
## For regions, highlight a region with keystrokes or mouse
## and then send with:
C-c C-r

## Re-edit myfile.s as necessary to correct any difficulties. Add
## new commands here. Send them to S by region with
C-c C-r
## or one line at a time with
C-c C-n

## Save the revised myfile.s with C-x C-s.

## Save the entire *S+3:1* interaction buffer with C-c C-s. You
## will be prompted for a file name. The recommended name is
## myfile.St. With the *.St suffix, the file will come up in ESS
## Transcript mode the next time it is accessed from Emacs.
```

- Program revision example (source code is real)

```
## Start up S-PLUS~3.x in a process buffer (this will be *S+3:1*)
M-x S+3

## Load the file you want to work with
C-x C-f myfile.s

## edit program, functions, and code in myfile.s, and
```



```

    ## send revised functions to S when ready with
C-c C-f
    ## or highlighted regions with
C-c C-r
    ## or individual lines with
C-c C-n
    ## or load the entire buffer with
C-c C-l

    ## save the revised myfile.s when you have finished
C-c C-s

```

- Program revision example (S object is real)

```

    ## Start up S-PLUS~3.x, in a process buffer (this will be *S+3:1*)
M-x S+3

    ## Dump an existing S object my.function into a buffer to work with
C-c C-d my.function
    ## a new buffer named yourloginname.my.function.S will be created with
    ## an editable copy of the object. The buffer is associated with the
    ## pathname /tmp/yourloginname.my.function.S and will
    ## almost certainly not exist after you log off.

    ## Enter program, functions, and code into work buffer,
    ## and send entire contents to S-PLUS when ready:
C-c C-b

    ## Go to *S+3:1* buffer, which is the process buffer, and examine
    ## the results.
C-c C-y
    ## The sequence C-c C-y is a shortcut for: C-x b *S+3:1*

    ## Return to the work buffer (may/may not be prefixed)
C-x C-b yourloginname.my.function.S
    ## Fix the function that didn't work, and resubmit by
    ## placing the cursor somewhere in the function and
C-c C-f
    ## Or you could've selected a region (using the mouse, or keyboard
    ## via setting point/mark) and
C-c C-r
    ## Or you could step through, line by line, using
C-c C-n
    ## Or just send a single line (without moving to the next) using
C-c C-j
    ## To fix that error in syntax for the "rchisq" command, get help
    ## by
C-c C-v rchisq

```

- Data Analysis (S object is real)

```

    ## Start up R in a process buffer (this will be *R*)
M-x R

    ## Work in the process buffer.  When you find an object that needs
    ## to be changed (this could be a data frame, or a variable,
    ## or a function), dump it to a buffer:
C-c C-d my.cool.function

    ## Edit the function as appropriate, and dump back in to the
    ## process buffer
C-c C-b

    ## Return to the R process buffer
C-c C-y
    ## Continue working.

    ## When you need help, use
C-c C-v rchisq
    ## (which you can do using command name completion)
    ## but entering: help("rchisq") or ?rchisq
    ## will bring up the help file in a separate buffer
    ## --- currently only for *R*.

```

4 Using ESS with SAS

5 Advanced Usage

- Switching the engine
- S+elsewhere

6 Obtaining ESS

ESS is primarily located at ess.stat.wisc.edu, and is available by FTP or through the World-Wide-Web (WWW). A basic installation consists of downloading the package, unpacking, and then adding a line to the emacs initialization file (`.emacs` or `_emacs`), pointing to the lisp subdirectory of the unpacked archive.

7 History

ESS (originally S-mode) was initially designed for use with S and S-PLUS(tm); hence, this family of statistical languages currently has the most support. We denote by S, any of the currently available members (or “dialects”) of the family, including S 3.x, S 4.x, S-PLUS3.x, S-PLUS 4.x (incl. S-PLUS 2000), S-PLUS 5.x, and R. In addition, we

denote by Emacs, one of the GNU family of editors, either Emacs (as developed and maintained by the Free Software Foundation) or XEmacs (which is a derivative work).

ESS is a remarkable example of how open-source products can continue to develop beyond what their initial authors planned. ESS started as S-mode, an Emacs extension which was developed in 1991 as a simple tool for editing program files for S and S-PLUS. In 1994, Rossini extended S-mode to support XEmacs, which was a forked version of Emacs. At the same time, Rossini extended a successful SAS-mode written by Tom Cook to work with XEmacs. In 1995, S-mode was merged into a uniform S-mode for Emacs, XEmacs, and supported S, S-PLUS, and R. During 1996 and 1997, this was extended to incorporate the SAS-mode as well as to provide a generic means for configuring ESS to accommodate changed as well as new statistical languages.

Most of this was primarily done under Unix. However, in 1998, thanks to an example of interprocess communication using Microsoft's DDE by Brian Ripley, Richard M. Heiberger provided interfaces for S-PLUS 4.x and then S-PLUS 2000.

Most of what has been done recently is debugging and tuning. New features in the plans include robust extensions for Literate Data Analysis using Noweb (Ramsey, 199x?), as well as extensions for XML-based Literate Statistical Analysis.

8 Remarks and Extensions

There are two active areas of extensions for user environments. One is to enhance the capabilities of the IDE for statistical practice; this includes implementing such common IDE features as object browsers as well as clean up the interface.

The other exciting extension is towards quasi-Literate Programming methodologies (Knuth, 1987?; Ramsey, 199x?), which we will refer to as Literate Statistical Analysis (Rossini and Lunt, 2001). The tools include the use of Noweb (Ramsey, 199x?) and an additional step towards the use of an XML authoring environment for statistical analysis. Literate Data Analysis based on Noweb, is a means of documenting a statistical analysis plan and procedure. Literate Statistical Analysis is intended to be a round-trip environment for both design and analysis.

The future use of XML is to accommodate the growing use of WWW-based services for document publishing as well as information conversion. By marking up the document with XML, it is possible to display subsets of the document contextually according to the intent of the document. Unlike Literate Data Analysis, which provides a single document for code and analysis plan, Literate Statistical Analysis is a round-trip cycle, using document modification through the Document Object Model interface (W3 Consortium). This is described more in (Rossini and Lunt, 2001).

References

Richard A. Becker, John M. Chambers, and Allan R. Wilks. *The S Language; A Programming Environment for Data Analysis and Graphics*. Wadsworth & Brooks/Cole, Pacific Grove, 1988.

- John M. Chambers. *Programming with Data; A Guide to the S Language*. Springer-Verlag, New York, 1998.
- John M. Chambers and Trevor J. Hastie. *Statistical Models in S*. Wadsworth & Brooks/Cole, 1992.
- Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.
- A.J. Rossini and Mark Lunt. Literate statistical analysis. Technical report, University of Washington, Biostatistics, 2001.
- Luke Tierney. *Lisp-Stat: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*. John Wiley & Sons, New York, 1990.
- Forrest W. Young, Richard A. Faldowski, and Mary M. McFarlane. Vista: A visual statistics research and development testbed. In *Computing Science and Statistics. Proceedings of the 24rd Symposium on the Interface*, pages 224–233. Interface Foundation of North America (Fairfax Station, VA), 1992.
- Forrest W. Young and David J. Lubinsky. Guiding data analysts with visual statistical strategies (disc: P251-260). *Journal of Computational and Graphical Statistics*, 4: 229–250, 1995.