Emacs Speaks Statistics: A Universal Interface for Statistical Analysis

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

February 16, 2001

Keywords: Statistical Analysis, Programming, User Interfaces, SAS, S-PLUS, R, XLispStat, Stata

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 an attempt to increase programmer productivity and efficiency. The increased speed in commercial 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 analytic tools are targeted for particular styles of statistical analysis. This suggests that efficiency work habits can require switching between data analysis tools. One example is that general purpose tools such as R (Ihaka and Gentleman, 1996) are not as easy to use performing Bayesian analyses as special tools such as BUGS (D.J. Spiegelhalter and Best, 1999). On the other

*Department of Biostatistics, University of Washington and Fred Hutchinson Cancer Research Center, Seattle, WA, USA
†Seminar for Statistics; ETH Zurich; Zurich, Switzerland
‡Technische Universität Wien, Vienna, Austria
§Temple University; Philadelphia, PA, USA
¶Medical College of Wisconsin, WI, USA

hand, these specialized tools can not be efficiently used to handle general analyses and graphical summaries. This suggests a potential efficiency gain through the use of using multiple computational tools for complex statistical analyses. However, most of these tools have their own idiosyncratic user interfaces, which can undo the efficiency gain.

Emacs Speaks Statistics (ESS) is intended to provide a single point of interface for statistical computing tasks using a keyboard. The primary user tasks which ESS focuses on for optimization and efficiency 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, for example C or Fortran; or it might be run by an interpreter, for example, SAS, S-PLUS R, or Python. 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.

Very few statistical packages have identical command-line interfaces; this is in addition to the differences between the actual macro or programming languages. This means that it might be necessary for a statistical analyst to learn 2 or 3 different text-based 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 (MathSoft, 2000), and R (Ihaka and Gentleman, 1996); XLispStat (Tierney, 1990) including the extension of Arc (Cook and Weisberg, 1999) and ViSta (Young et al., 1992); SAS (Institute, 2000); Stata (StataCorp, 1999); Omegahat (Temple Lang, 2000); and can be extended in at least a limited way to most statistical packages which can be controlled from a command-line.

The rest of this introduction will discuss the proposed development environment and how it can be used. Sections 4 and 5 will discuss how this is realized for two very distinct statistical languages, the S family and SAS. The history of this project will then be discussed, and this paper will conclude with a discussion of future work in this area, including Literate Statistical Analysis and other planned enhancements.

1.1 Emacs

Emacs (Stallman, 2000) is a mature, powerful, and easily extensible text editor which is freely available, under the terms of the Free Software Foundation's Copy-left agreement, for a large number of environments. These include frequently used statistical computing platforms such as Apple Corporation's Macintosh platform, and Microsoft-based operating system. Emacs has been around for many years, and in the past was considered as one of the primary options for text editing on many Unix systems.

Most programming and documentation tasks fall under the realm of text editing. This work can be enhanced by common IDE features such as contextual highlighting and recognition of special reserved words for the programming language being used. In addition, editor behaviors such as folding, outlining, and bookmarking can assist with maneuvering around a file. One task which is not in this realm is word processing, which focuses on producing a presentation of a document, in addition to text editing. While Emacs is not a word processor, it does share some features.

ESS provides Emacs with an easily extensible and uniform interface to text handling for statistical packages. The capabilities can be extended in an orthogonal manner to include other Emacs packages for assisting with documentation (LaTeX; SGML, XML, and XSLT; Noweb); version control (RCS,CVS,SCCS, PRCS); and remote editing via FTP or secure mechanisms such as SSH/SCP. It handles the interface to both source code and transcripts contextually, providing syntax highlighting, bookmarking features, interfaces to directory structure, and command-history.

Emacs provides a sophisticated development environment for programming in traditional compiled languages such as C, 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. Finally, it assists with project tasks such as documentation of changes, design, and version control.

Other 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 an interface to other common text-based tools such as spell checking programs. It can have its interface re-mapped to resemble that of other text-editors, such as ed, vi, wordstar, and brief.

The interface to Emacs is by keystrokes on all systems, as well as by mouse input on systems which support it, including both windowing and non-windowing environments. The mouse-based interface, through menus and toolbars, also tries to facilitate the learning of keystroke-based short-cuts.

Emacs was one of the first tools available with a programmable extension language, which is a dialect of Lisp (Chassell, 1999; Graham, 1996). There are many examples of specialized extensions for assisting and facilitating common editing tasks. This includes strange ones such as connections with spreadsheets, database front and backends, forms entry, and even interactive games such as tetris. However, important tasks such as programming Makefiles, scripts, and compiled code are definitely not neglected.

Emacs also has an internal help system and tutorial for its use. Appendix A provides sample sessions to help familiarize one with approaches for using Emacs with ESS.

The above reasons make Emacs a reasonable choice as a starting point for providing a universal interface for data analysis and programming. ESS is an attempt to provide a functional and extensible common interface to multiple packages.

2 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. Another approach is taken by DataDesk (Velleman and Pratt, 1989), which provides a visual and tightly linked interface.

2.1 Command line interfaces

The classic command-line interface is still available for statistical packages on most Unix and mainframe computers. This generally consists of typing a single line at a time, with some form of wrap-around for those lines which are extremely long in terms of character counts. For example, S-PLUS, R, Stata, XLispStat, and SAS still have command-line interfaces, though they are slowly disappearing. For example, it is difficult to determine how to use it under SAS, and the Stata interface is depreciated towards use of a newer GUI. ESS assists command-line interfaces by providing a comprehensive interface layer on top of the command-line, so can fail to integrate well with non-command line interfaces.

2.2 Graphical User Interfaces

The advent of the Apple Macintosh and Microsoft Windows operating systems have encouraged a partial standardization of interfaces for statistical packages. Many packages written for both IBM-based and Apple-based personal computers have descended from the spreadsheet data-table approach. Generally, these feature a cases-by-variables spreadsheet representation of the data and use pull-down menus and dialog boxes for data analysis activities. Depending upon their nature, numerical results can be saved on the spreadsheet. Software that takes this approach includes SPSS, Minitab, S-PLUSfor Windows, DataDesk, ViSta, and many others.

2.3 ESS

ESS provides a front-end text-based user interface to a variety of interactive statistical programs and interpreted statistical languages. There are various levels of support depending on their capabilities and needs. Because of it's 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 XLisp-Stat 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.

3 Features and Use

ESS provides a number of features for statistical programming beyond those provided by Emacs, as 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 execution. 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.

ESS also facilitates the editing of source code by providing a means for loading and error-checking of code for S, XLispStat, and SAS. This allows for one form of crude source-level debugging.

iESS, the mode for controlling interactive processes, allows for command-line editing and saving history, as well as recalling and searching for previously entered commands. File completion exists, and for for S languages, there exists object and function completion. Transcripts are easily recorded and editable into a coherent activity log, and there is a good interface to the help systems for S, XLispStat, and Stata. In addition, there is the facility to edit and run programs on remote machines which potentially could be quite different platforms than the local machine.

Transcript manipulation is another important task that ESS facilitates. Once a transcript log is generated, perhaps saving an iESS buffer, transcript-mode allows for a quick means of reproducing results, which is useful for demonstration of techniques as well as for reconstruction of data analyses. In particular this allows for the recording and saving of transcripts, manipulating and editing saved transcripts, as well as

re-evaluating commands from transcript lines.

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.

4 Using ESS with the S family of languages

We give, as one application, how ESS can be used with the S language, and its implementations via S-PLUSand R. Since ESS originated as S-mode, this indicates why it is so strong for programming S. We give examples for editing files, communicating with the interactive S process, handling transcripts, and getting help.

4.1 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; the 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; the ability to switch between processes which would be the target of the buffer (see previous); the ability to request help from an S process for variables and functions, and to have the results sent into a separate buffer; and completion of object names and file names.

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

4.2 Inferior ESS processes

iESS (inferior ESS) is the mode for interfacing with active statistical processes (programs). The processes are referred to as inferior since they are controlled by Emacs. The inferior ESS modes 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 and file names; interactive history mechanism; and transcript recording and editing.

Generally, the statistical processes are started by running a function, using the Emacs function call `M-x`, for example by `M-x S+3`. This assumes that you have access to the desired statistical package, which doesn't come with ESS. Generally, these start up the package with a particular set of command-line arguments which are tailored for ESS. However, some packages such as R have 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):

```
--max-vsize=10000 --max-nsize=5000
```

Then that R process will be started up with those particular arguments.

One critically useful command family is that of the `S+elsewhere` and `ESS-elsewhere` commands, which allow for the use of a remote machine to execute the program or analysis. The idea of `S+elsewhere` is that we open a telnet, rlogin, or ssh connection to another machine, call the buffer `*S+elsewhere*`, and then run S on the other machine in that buffer. It is still possible to bring up visual displays from the remote system locally, using the X11 windowing system. Note that this holds for Unix, Microsoft, and Apple platforms being used for the local system.

4.3 Handling and Reusing Transcripts

The general features which are provided for transcript handling are a mode for editing transcripts which understands the form of S transcripts; 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; and the ability to switch between processes which would be the target of the buffer (for the above).

4.4 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).

4.5 Philosophies for combining S with ESS

There are two primary philosophies for using ESS with the S family of languages. The first, which is preferred by the current group of developers, is how ESS is initially configured. In this approach, 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. This implicitly assumes that one might want to use written code and analyses on possibly different versions of the S language and allows for better portability as well as external version control for source code.

The second, depreciated view, is that S objects are real. Source code is a temporary realization of the objects, and that dumped buffers should not be saved. We strongly

discourage this approach, which is a natural result of assuming that the S process will always be the same. However, it can be realized by adding the following lines to ones `.emacs` configuration file:

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

This "old" view was based the permanent storage of objects approach in the original S implementations, which used a separate disk files for each object, and saves a small amount of disk space.

5 Using ESS with SAS

The SAS module in ESS (ESS[SAS]) is currently undergoing development but is still functional and pretty stable. While ESS (originally S-mode) was initially designed for use with S and S-PLUS it has been extended to support other languages, including SAS. The editing of SAS files is based on the stable, old SAS mode by Cook (ftp://ftp.biostat.wisc.edu/pub/cook/sas-mode/sas.tar.gz). Those editing features and new advanced features are part of ESS[SAS]. The user interface with ESS[SAS] is similar in behavior to the interface for S (unlike Cook's SAS mode) and to the windowing interface that SAS itself uses.

5.1 Editing SAS Files

ESS[SAS] is the mode for editing SAS 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, or a single line to an inferior SAS process, if one is currently running; ability to switch between processes which would be the target of the buffer (for the above); ability to save and submit the file you are working on as a batch SAS process with a single keypress and to continue editing while it is running; the capability of killing the batch SAS process through the shell buffer or allow the SAS process to keep on running after you exit Emacs; single keypress navigation of .sas, .log and .lst files (.log and .lst files are automatically refreshed with each keypress).

ESS[SAS], the mode for editing SAS language files, is automatically turned on when editing a file with a ".sas" suffix (or other, if specified in ess-site). The batch processing keypress commands are enabled by default to use the same function keys that the SAS Display Manager uses. The interactive capabilities of ESS require you to start an inferior SAS process with M-x SAS (described below).

At this writing (ESS release 5.2), the indenting and syntactic highlighting are usually correct, though the SAS language is rather complex, so this isn't certain.

5.1.1 SAS and TAB

There are two options for how ESS will interpret the [TAB] key with respect to SAS code. The TAB key is bound by default to sas-indent-line. This function is used to syn-

tactically indent SAS code so PROC and RUN are in the left margin, other statements are indented 4 spaces from the margin, continuation lines are indented 4 spaces in from the beginning column of that statement. This is the type of functionality that Emacs provides in most programming language modes. This functionality is equivalent to uncommenting the following line in ess-site.el:

```
;;; (setq ess-sas-edit-keys-toggle 0)
```

ESS provides an alternate behavior for the TAB key that makes it behave as it does on non-emacs terminals, i.e. move the cursor to the next tab stop. The alternate behavior also provides a backwards TAB key: C-TAB. This functionality is obtained by uncommenting the following line in ess-site.el:

```
;;; (setq ess-sas-edit-keys-toggle 1)
```

Under the alternate behavior, the TAB key is bound to tab-to-tab-stop and the tab stops are set by default at multiples of 4.

5.2 SAS process interaction

There are two ways to interact with SAS. The first facilitates the common Unix interface technique of editing SAS programs and submitting them as batch files. The second attempts to emulate the 3-level terminal window approach.

5.2.1 SAS Batch processing

The description of the setup for function keys for batch processing of SAS files is unavoidably more complex than we wish it were. The actual use of the function keys is simple. There are five distinct options.

The first option is to have function keys in ESS[SAS] modes do whatever they normally do in other emacs modes. Many users will have defined some of the keys [f2]-f[8] in their `.emacs` or `_emacs` file, or will have installation-wide definitions in their site-start.el file. By default, ESS does not override those definitions. This default is intended to make ESS[SAS] mode behave similarly to other emacs modes.

Users who are primarily familiar with SAS, and who are learning emacs as a way to approach SAS, will likely want to duplicate the function key capabilities that SAS Institute provides with its Display Manager. There are still 4 distinct options for how this can be done. SAS provides different function key definitions with its PC and Unix products; ESS can use either. The ESS[SAS] function key definitions can be active in all buffers (global) or limited (local) only to buffers that have SAS-related file extensions: .sas, .log, .lst, and "Type-1" which defaults to .txt. The distinction between local and global appears subtle. If you want the ESS[SAS] definitions to work when you are in *shell* or when editing files other than the file extensions that SAS recognizes, you will most likely want to use the global definitions. If you want your function keys to understand SAS batch commands when you are editing SAS files, and to behave normally when editing other files, then you will choose the local definitions.

The option can be chosen both site-wide and by individual users.

The function keys act as shown in Table 1.

Unix	PC	Action
F2	F2	**Refresh**: revert the buffer with the file if the file on disk is newer than the file currently in the buffer.
F3	F8	**Submit**: save the current .sas buffer (which may be the .sas file associated with the .lst or .log file you are actually looking at) to a file and submit the file to a background SAS job.
F4	F5	**Program**: switch buffer to .sas file.
F5	F6	**Log**: switch buffer to .log file, ”refresh” and goto next error message, if any.
F6	F7	**Listing**: switch buffer to .lst file and ”refresh”.
F7	F4	**Type-1**: switch buffer to Type-1 (defaults to .txt) file and ”refresh”.
F8	F3	**Shell**: switch buffer to shell.

Table 1: SAS Function Key Options

Keys [f3]-[f8] mimic SAS Display Manager keys. One other key has been provided for convenience. F2 performs the ”refresh” operation on the current buffer. ”refresh” compares the buffer date stamp with the file date stamp and replaces the buffer with the file if the file is newer. This is the same operation that is automatically performed when Log, Listing, or Type-1 are pressed. Type-1 takes you to a file with a user-specified extension; .txt by default. You can over-ride the default, by specifying a different extension in your .emacs file:

```
(setq ess-sas-suffix-1 '.txt')
```

5.2.2 Inferior SAS processes

iESS (inferior ESS) is the mode for interfacing with active statistical processes (programs). To start up iESS[SAS] mode, use:

```
M-x SAS
```

We plan to add the ability to request help from a process for variables and functions, and to have the results sent into a separate buffer, as well as the completion of object names and file names.

5.2.3 SAS Interface

The default command used by the Submit function key (F3 or F8) to submit a batch SAS job is simply ”sas”. If necessary, you can over-ride this in your `.emacs` file something like one of the following:

```
;; (setq ess-sas-submit-command "nohup nice sas") ;; Unix
;; (setq ess-sas-submit-command "c:/progra~1/sas/sas.exe") ;; Windows
;; (setq ess-sas-submit-command "invoke SAS using program file") ;; Mac
```

Note that when specifying a path on Windows, avoid spaces (that is use ”progra 1”, not ”Program Files”) and use forward slashes ”/”. There may be occasions when you want to run a particular buffer under a different version of SAS or specify different options on the command line. In that case, you can override `ess-sas-submit-command` in your local buffer with file variables or other means.

There is a built-in delay before a batch SAS job is submitted when using a Unix shell under either Unix or Windows. This is necessary in many cases since the shell might not be ready to receive a command. This delay is currently set high enough so as not to be a problem on any system. But, there may be cases when it needs to be set higher, or could be set much lower to speed things up. You can over-ride the default in your .emacs file by (the default of 5 seconds is shown):

```
(setq ess-sleep-for 5)
```

5.3 Using Transcripts

Transcripts for SAS are not yet available. However, the basic intent is that a marked section of a log file from a previous SAS run can be resubmitted to SAS. The user would highlight a region from ’PROC’ to ’RUN;’ and then send it to the inferior SAS process with the [RET] key. ESS would automatically clean the region (remove line numbers) and send the entire region over as a single request to SAS.

5.4 ESS interface with SAS Design Philosophy

ESS[SAS] mode was designed to aid the user in writing and maintaining input command files, such as myfile.sas, for SAS. These are files containing SAS statements. In a batch environment such files would be submitted to SAS by the operating system command:

```
sas myfile.sas
```

In a SAS window environment, these files would be brought into the ”SAS: PROGRAM EDITOR” window and then submitted with the ’Local’ ’Submit’ menu commands.

The *SAS:1.log* buffer in ESStr mode corresponds to the file myfile.log in SAS batch usage and to the ”SAS: LOG” window in the SAS window environment. All commands submitted to SAS, informative messages, warnings, and errors appear here.

The *SAS:1.lst* buffer in ESSlst mode corresponds to the file myfile.lst in SAS batch usage and to the ”SAS: OUTPUT” window in the SAS window environment. All data related printed output from the PROCs appear in this window.

The iESS [SAS:1] buffer exists solely as a communications buffer. Files are edited in the myfile.sas buffer. The C-c C-r key in ESS[SAS] mode is the functional equivalent of bringing a file into the ”SAS: PROGRAM EDITOR” window followed by the ’Local’ ’Submit’ menu commands. The user should never use this buffer directly.

The ESS[SAS] mode was written with two primary goals.

- Using Emacs, a window environment becomes available for dial-up users who do not have access to the SAS window environment.

- The authors prefer the Emacs environment for editing and managing input and output files, even on computer systems which run the SAS window environment.

A secondary goal, to allow for a reasonable interface under slow connections, was also realized. That is, with an X-windows terminal connected by ppp at 14400 baud to a Unix system running SAS, iESS[SAS] interaction with SAS was hundreds of times faster than the SAS window system. The savings come because the ESS windows are subunits of a text-based terminal window, rather than the remotely managed graphical windows provided by SAS. The speedup is based on SAS timings from the log files. The SAS windows times include window management and communications times as well as calculation times, while the iESS times include only the calculations by the SAS computing engine.

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.

There exist packages for various Linux distributions, as well, including Debian, RedHat, Mandrake, and SuSE.

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 originated from the extension for programming and process interface in Emacs supplied by S-mode (D. Bates, F. Ritter, et.al.; M. Meyer; 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.

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, 1994), 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 the use of Literate Programming methodologies (Knuth, 1992; Ramsey, 1994), which we will refer to as Literate Statistical Analysis (Rossini and Lunt, 2001). The tools include the use of Noweb (Ramsey, 1994) 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.

Robert Chassell. *Programming in Emacs Lisp: An Introduction*. Free Software Foundation, 2nd edition, 1999.

R. Dennis Cook and Sanford Weisberg. *Applied Regression Including Computing and Graphics*. John Wiley & Sons, August 1999.

A. Thomas D.J. Spiegelhalter and N.G. Best. Winbugs version 1.2 user manual. Technical report, MRC Biostatistics Unit, 1999.

Paul Graham. *ANSI Common Lisp*. Prentice Hall, 1996.

Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.

SAS Institute. Sas statistical software: Release 8.0, 2000. Cary, NC: SAS Institute.

Donald E. Knuth. *Literate Programming*. Number 27 in CSLI Lecture Notes. Center for the Study of Language and Information, 1992.

MathSoft. S-plus statistical software: Release 5.1, 2000. Seattle, WA: MathSoft.

Norman Ramsey. Literate programming simplified. *IEEE Software*, 11(5):97–105, September 1994.

A.J. Rossini and Mark Lunt. Literate statistical analysis. Technical report, University of Washington, Biostatistics, 2001.

Richard M. Stallman. *GNU Emacs Manual, for Version 20.7*. Free Software Foundation, 13th edition, 2000.

StataCorp. Stata statistical software: Release 6.0, 1999. College Station, TX: Stata Corporation.

Duncan Temple Lang. The omegahat environment: New possibilities for statistical computing. *Journal of Computational and Graphical Statistics*, 9(3), September 2000.

Luke Tierney. *Lisp-Stat: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*. John Wiley & Sons, New York, 1990.

Paul F. Velleman and Paul Pratt. A graphical interface for data analysis. *Journal of Statistical Computation and Simulation*, 32:223–228, 1989.

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.

A Scenarios for Use

The next 2 sections discuss specific use patterns for ESS under 2 very different statistical language systems, S and SAS.

A.1 Scenarios for using S

We present some basic suggestions for using ESS to interact with S. These are just a subset of approaches, many better approaches are possible. Comments for the intent of 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-x 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-x 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*.
```

A.2 Scenarios for using ESS with SAS

We present one scenario for using ESS to interact with SAS. 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 "##").

Batch SAS (-unix-keys illustrated, pc-keys in the comments)

```
   ## Find the file you want to work with
C-x C-f myfile.sas
   ## myfile.sas will be in ESS[SAS] mode

   ## Edit as appropriate, then save and submit the batch SAS job.
   ## -pc- F8
F3

   ## The job runs in the shell buffer while you continue to edit
   ## myfile.sas.  If you are running a Unix shell under Unix or
   ## Windows, the message buffer will notify you with a shell
   ## notification when the job is complete.  If so, then you
   ## will also have the option of terminating the batch job
   ## before it is finished.
   ## -pc- F3
F8

   ## In any case, you may want to visit the .log while the process
   ## is still running (unix only) or when it is done and check for
   ## error messages
   ## (you will be taken to the next error message, if any).
   ## The .log will be refreshed and you will be placed in it's buffer.
   ## -pc- F6
F5

   ## Now, refresh the .lst and go to it's buffer.
   ## -pc- F7
F6

   ## If you wish to make changes, go to the .sas file with
   ## -pc- F5
F4
   ## and make your editing changes.  The go back to the submit instruction.
```

```
   ## Find the file you want to work with
C-x C-f myfile.sas
   ## myfile.sas will be in ESS[SAS] mode

   ## Edit as appropriate, and then start up SAS with the cursor in
   ## the myfile.sas buffer
M-x SAS

   ## Four buffers will appear on screen:
   ## myfile.sas  in  ESS[SAS]        mode # your source file
   ## *SAS:1*     in  iESS [SAS:1]    mode # ESS communication buffer
   ## *SAS:1.log* in  Shell [] ESStr  mode # SAS log information
   ## *SAS:1.lst* in  Shell [] ESSlst mode # SAS listing information

   ## If you would prefer each of the four buffers to appear in its
   ## own individual frame, you can arrange for that.  Place the
   ## cursor in the buffer displaying myfile.sas.  Enter the
   ## sequence:
C-c C-w

   ## The cursor will normally be in buffer myfile.sas.
   ## If not, put it there:
C-x b myfile.sas

   ## Send regions, lines, or the entire file contents to SAS
   ## (regions are most useful).  A highlighted region will normally
   ## begin with the keywords 'DATA' or 'PROC' and end with the
   ## keyword 'RUN;'
C-c C-r

   ## Information appears in the log buffer, analysis results in the
   ## listing buffer.  In case of errors, make the corrections in the
   ## myfile.sas buffer and resubmit with another C-c C-r

   ## PROC GPLOT graphs will normally be produced in a postscript
   ## file and be viewed later.  Include the lines
/* required gsoptions for batch files */
/* comment out these lines for interactive use on X-terminals*/
filename grafout 'temp.ps';
goptions device=ps gsfname=grafout gsfmode=append gaccess=sasgastd;
   ## in myfile.sas.

   ## PROC PLOT graphs can be viewed in the listing buffer.  You may
   ## wish to control the vertical spacing to allow the entire plot
   ## to be visible on screen, for example, by
proc plot; plot a*b / vpos=25;

   ## At the end of the session you may save the log and listing
```

```
## buffers with the usual C-x C-s commands.  You will be prompted
## for a file name.  Typically, the names myfile.log and mfile.lst
## will be used.  You will almost certainly want to edit the saved
## files before including them in a report.  The files are
## read-only by default.  You can make them writable by the Emacs
## command C-x C-q.

## At the end of the session, the input file myfile.sas will
## typically have been revised.  You can save it.  It can be used
## later as the beginning of another iESS[SAS] session.  It can
## also be used as a batch input file to SAS.

## The *SAS:1* buffer is strictly for ESS use.  The user should
## never need to read it or write to it.  Refer to the .lst and
## .log buffers for monitoring output!
```

Here is a typical myfile.sas:

```
title 'Analysis of Case 0502';

data case0502;
    infile '/home/public/stat/Data/case0502.asc'
           firstobs=2;
    input percent code;
run;

proc glm;
  class code;
  model percent=code /ss1;
run;
```