Emacs Speaks Statistics: A Universal Interface for Statistical Analysis

A.J. Rossini Martin Mächler Kurt Hornik Richard M. Heiberger
Rodney Sparapani *

August 21, 2024

Abstract

Emacs Speaks Statistics (ESS) is a user interface for developing statistical applications and performing data analysis using any of several common statistical programming languages. ESS falls in the programming tools category of Integrated Development Environments (IDEs), which are approaches for developing and visualizing computer programs. We discuss how it works, the advantages of using it, and extensions for increasing statistical programming efficiency.

Keywords: Data Analysis, Programming Tools, User Interfaces, SAS, S-PLUS, R, XLISPSTAT, STATA

1 Introduction

Integrated Development Environments (IDEs) are computer programming environments which combine features and tools in a single interface with the intent of increasing programmer productivity and efficiency. In the last decade, the use of IDEs and other Rapid Application Development (RAD) tools has shortened the development time of complex software systems which use programming languages such as Visual Basic, Java, C++, Python, and Smalltalk. Computer programming, in the context of both data analysis and computer program development, is an important statistical skill which can be augmented and

*A.J. Rossini is Research Assistant Professor in the Department of Biostatistics, University of Washington and Joint Assistant Member at the Fred Hutchinson Cancer Research Center, Seattle, WA, USA (E-mail: rossini@u.washington.edu); Martin Mächler is Senior Scientist and Lecturer in the Seminar for Statistics, ETH Zurich, Zurich, Switzerland (E-mail: maechler@stat.math.ethz.ch); Kurt Hornik is Professor in the Institut für Statistik, Wirtschaftsuniversität Wien and the Institut für Wahrscheinlichkeitstheorie und Statistik, Technische Universität Wien, Vienna, Austria (E-mail: Kurt.Hornik@r-project.org); Richard M. Heiberger is Professor in the Department of Statistics at Temple University, Philadelphia, PA, USA (E-mail: rmh@temple.edu); and Rodney Sparapani is Senior Biostatistician at the Medical College of Wisconsin, Milwaukee, WI, USA (E-mail: rsparapa@mcw.edu)

enhanced by the right tools and environment. This is especially true with the increased use of simulation-based statistical procedures including resampling and Markov Chain Monte Carlo (MCMC) sampling.

Statistical software tools are intended for either general data analysis or for specialized forms of statistical analyses. The specialized tools can be orders of magnitude more efficient for generating data analyses. This is balanced by the inability of the specialized tools to perform a wide range of common data analysis operations. Tightly coupled inter-operability between these programs rarely exists, while the need to switch between tools occurs much more often. For example, general purpose tools such as R (Ihaka and Gentleman, 1996) do not perform Bayesian analyses as easily as tools such as WinBUGS can (Spiegelhalter et al., 1999). On the other hand, these specialized tools lack breadth in the range of analyses and graphics that can be generated. For this reason, BUGS is distributed with applications providing loose inter-operability for simplifying the transfer of output to R or S-PLUS for processing the results. Although the interfaces for WinBUGS and R are similar, they are distinct enough to create tension for the analyst.

Emacs Speaks Statistics (ESS) (Rossini et al., 2001) is an extension package for the Emacs editor which provides a single interface for a variety of statistical computing tasks. ESS is optimized for statistical coding and interactive data analysis. Statistical coding is the writing of computer code for data analysis. This code might be in a compiled language, such as C or FORTRAN, or it might be in an interpreted language such as S-PLUS, SAS, R, XLISPSTAT, PERL, or Python. Entering commands for interactive data analysis is a similar activity. In either case, text is written in a computer language and sent to a computer program for evaluation. The primary difference is that the results of a small set of commands are of critical interest for review in the analysis phase, but the results of all commands are of interest in the coding phase. Both of these tasks can occur simultaneously, for example in the use of compiled C code for optimization, which is called from an interpreted language such as R, where the objective function to optimize is written.

Simple conflicts between interfaces are exemplified by different keystrokes for editing tasks such as copy and paste, beginning of line, highlighting regions. These are sometimes the most aggravating because our fingers are trained for a primary interface convention which is continually interfered with. ESS solves this problem by providing a uniform keyboard interface.

Complex conflicts between interfaces can involve the coordination of several data files, multiple statistical software packages, and the corresponding source code in each of these languages, combined for a single analysis. ESS, as part of Emacs, has tools which assist in this, including support for version and source code control systems, tools for accessing programs or files on remote machines, and interfaces to documentation systems including LaTeX and XML. In addition, Emacs can assist with, or be programmed to perform, many tasks related to data cleaning, management, and editing.

It can be useful to have multiple statistical processes running simultaneously, either on a single machine or a variety of machines. This capability assists with code design and testing across multiple versions of statistical software packages, as well as large scale numerical simulations. For example, one might want to be connected to multiple R processes of different versions in order to verify behavior on different versions of the same software; multiple processes of the same version to perform test-and-run scenarios where one process is doing long-term processing while the other is doing short-term testing; simulations; and to distribute the load over a variety of remote machines.

ESS mitigates some of the problems noted above. It provides a well designed and documented programming editor, an interface to statistical processes, and provides, through Emacs, additional tools which aid both statistical software development and data analysis. It works with common statistical software, including the S languages (which include S (Becker et al., 1988; Chambers and Hastie, 1992; Chambers, 1998), S-PLUS (Insightful Corp., 2001), and R (Ihaka and Gentleman, 1996)); XLISPSTAT (Tierney, 1990) and its extensions Arc (Cook and Weisberg, 1999) and ViSta (Young et al., 1992); SAS (SAS Institute, 2000); STATA (StataCorp, 1999); Omegahat (Temple Lang, 2000); and BUGS (Spiegel-

halter et al., 1999). ESS can be extended to accommodate most statistical packages which can be controlled from a command-line.

The present paper provides historical and non-technical information on ESS and its capabilities. A general introduction and usage instructions can be found in Heiberger (2001). Details on the programmer API for ESS as well as examples of its use are included in package documentation that comes with the ESS source. The next section introduces common forms of statistical user interfaces. This is followed by a discussion of ESS and Emacs. Specifics on the realized interfaces for the S family and SAS are discussed in sections 4 and 5 respectively. The conclusion brings up related work and future extensions.

2 Statistical User Interfaces

Human interfaces for statistical packages and languages can generally be classified into three forms, the command-line interface, the graphical user interface, and a catch-all category for specialized interfaces which only have one or two examples. Command-line interfaces (CLIs) have historically been used by many packages. This is the interface that ESS uses for control and provides a limited interface for. Modern statistical packages have generally used graphical user interfaces (GUIs), sometimes exclusively. In addition, there have been many single-implementation interfaces, including flow-chart guides, implemented as graph-based interfaces in ViSta (Young and Lubinsky, 1995); the SAS terminal interface, which divides the terminal window into three screens; and the coherent implementation provided by DataDesk (Velleman and Pratt, 1989), which has a tight link between data and graphics.

Computer users strongly resist changes to their interface. This can be observed in the choice of editors such as vi and emacs; mail user agents such as mail, Pine, Elm, Netscape Mail, and Microsoft Outlook; word processors such as Microsoft Word and Corel WordPerfect; and other commonly used tools. The user's choice has little to do with theory, and more to do with compatibility with previous usage and with perceived size of the learning curve for a new package. However, sometimes the hardest interfaces to

master are the most efficient in the long run—after the initial investment in time.

2.1 Command line interfaces

The CLI is available for many statistical packages, particularly those that were originally developed under Unix, MS-DOS, and mainframe computers. The CLI is the successor to the IBM punch-card based batch interface. The CLI consists of entering commands one line at a time. Current examples are S-PLUS, R, STATA, and XLISPSTAT. Note that the STATA CLI is deprecated in favor of a new GUI, and this direction seems to be a general trend. We note that SAS does not have a true CLI. The two possible options for SAS are a prompt-less input device which sends a combined log and list-file text stream directly back to the same device, and second, the classic three-window terminal interface which is being deprecated in favor of a GUI version.

2.2 Graphical User Interfaces

The advent of Apple Macintosh (Mac) and Microsoft Windows (Windows) operating systems have encouraged a partial standardization of interfaces for statistical packages. Many packages written for both Mac and Windows personal computers have adopted a data display descended from the spreadsheet data-table approach. Generally, these feature a cases-by-variables spreadsheet representation of the data, along with pull-down menus and dialog boxes for data analysis activities. Depending upon their nature, numerical results can be saved as well as displayed on the spreadsheet. Statistical software packages that take this basic approach include SPSS, Minitab, S-PLUS (4.x for Windows and 6.x for Unix and Windows), DataDesk, and ViSta.

Most statistical software, including spreadsheets, use the Multiple Document Interface (MDI). This user interface style encloses multiple views of text, figures, and toolbars enclosed by the surrounding application. The use of MDI seems to be more a function of the availability of GUI toolkits for Mac and Windows applications, which promote such interfaces, rather than a consistent choice based on theories

for user-interface design. The common approach taken by statistics packages for MDI is to wrap menus and toolbars around a region containing representations of documents. These representations usually include one or more frames possibly featuring display graphics, a command-line window, a source code editing window, or possibly a spreadsheet displaying, for example, a cases-by-variables view of the data set or a drill-down contingency table interface. This approach is employed by both spreadsheet packages such as Microsoft Excel as well as most Mac and Windows statistical packages including S-PLUS and SPSS.

The Single Document Interface (SDI) provides a single frame for each document, view, or figure created. Unix-based interfaces tend to be SDI-based, due in part to the nature of the X11 windowing system—with its flexibility in allowing a single application to place frames and windows across multiple displays on potentially multiple computers—and in part to the lack until recently of available GUI toolkits which worked across different Unix platforms. Examples of statistics packages using the SDI are the command-line interfaces for STATA, S-PLUS, XLISPSTAT, or R under Unix and the RGUI interface for R under Windows. In each of these the terminal window is considered to be one document and the graphics displays to be the other parts of an overall GUI.

3 The ESS user interface

ESS extends Emacs by providing a number of features for statistical programming. This is done by extending the editor to provide additional useful features, and in the case of interactive statistical programs, sitting "in front of" their CLI and intercepting and modifying I/O as needed. We describe the exact features more in this section.

Figure 1 provides an example of how it looks when being used with XEmacs. This screen-shot shows both SAS and R code being examined at the same time, with an R interactive process being controlled from within XEmacs.

Figure 2 shows ESS running in NTemacs 21.0 on Microsoft Windows 2000. We are currently displaying 6 buffers. `highlight.s` illustrates several uses of syntactic highlighting. The most glaring one is the bright purple indicator for the unbalanced parentheses. We also see color choices for keywords (`if`), comments, and quoted strings. The string on the last line was not properly terminated and we are immediately warned by the string color staying on through (what we think is) the end of the line.

The two buffers `transcript-before.st` and `transcript-after.st` show transcript editing. A single ESS command converted the before buffer into the after buffer by removing all lines that do not begin with a prompt character.

The last three buffers show that a single S language source file can be used with two (or more) executing processes. In this example we sent over first a subset of the line in `tmps.s` and then the entire line to the instance of S+4 running in an inferior `iESS(Sqpe)` buffer. Then we switched the connection to send the same line to the instance of R running in the `iESS(R)` buffer. The dialog about the process switch appears in the message line.

3.1 Features and capabilities

Since ESS originated as S-mode, which provided an interface for programming and process control under GNU Emacs for S-PLUS version 3, ESS strongly supports the S family of languages; these include recent versions of S, S-PLUS, and R. SAS is also well supported. STATA and XLISPSTAT (and the XLISPSTAT extensions, ARC and ViSta) are supported with the basic functionality of syntax highlighting and process-interfacing. ESS and its interface are fully discussed in the next section.

Syntactic indentation and color/font-based source code highlighting. The ESS interface includes a description of the syntax and grammar of each statistical language it knows about. This gives ESS the ability to edit the programming language code, often more smoothly than with editors distributed with the languages. The process of programming code is enhanced as ESS provides the user with a

clear presentation of the code with syntax highlighting to denote assignment, reserved words, presence of strings, and comments. ESS has customizable automatic indentation, with the customization based on the syntactic structure of groups of expressions. ESS knows the structure of transcripts of the executing session and provides syntactic highlighting for transcripts. ESS interacts well with the executing statistics language/program and provides means for searching the command-line history for previous commands and editing them for current use.

Partial code evaluation. Emacs can send individual lines, entire function definitions, marked regions, and whole edited buffers from the window in which the code is displayed for editing to the statistical language/program for execution. Emacs can complete partially typed file names by referring to the current working directory. Emacs sends the code directly to the running program and receives the printed output back from the program. This is a major improvement over cut-and-paste as it does not require switching buffers or windows. The response is received immediately in an editable Emacs buffer.

Object name completion. In addition, for languages in the S family (S developed at Bell Labs, S-PLUS, and R) ESS provides object-name completion of both user- and system-defined functions and data. ESS can dump and save objects (user- and system-generated) into text files in a formatted manner for editing, and reload them (possibly after editing) back into the statistical language/program.

Source code checking. ESS facilitates the editing of source code by providing a means for loading and error-checking of small sections of code for S, XLISPSTAT, and SAS. This allows for source-level debugging of batch files.

Process interaction. Emacs has historically referred to processes under its control as "inferior", accounting for the name inferior ESS (`iESS`) to denote the mode for interfacing with the statistical pack-

age. The output of the package goes directly to an editable text buffer in Emacs. This mode allows for command-line editing and saving history, as well as recalling and searching for previously entered commands. Filename completion is available. In addition (currently only for S languages), there exists object-name and function-name completion. Transcripts are easily recorded and can be edited into an ideal activity log which can then be saved. There is a good interface for handling and intercepting calls to the internal help systems for S, XLISPSTAT, and STATA.

Interacting with statistical programs on remote computers. ESS provides the facility to edit and run programs on remote machines in the same session and with the same simplicity as if they were running on the local machine. The remote machine could be a very different platform than the local machine.

Transcription Editing and Reuse. Once a transcript log is generated, perhaps by saving an '`iESS`' buffer, transcript-mode assists with reuse of part or all of the entered commands. It permits editing and re-evaluating the commands directly from the saved transcript. This is is useful for demonstration of techniques as well as for reconstruction of data analyses. There currently exist functions within ESS for cleaning transcripts from S languages back to source code by finding all input lines and isolating them into an input file.

Help File Editing (R). ESS also provides an interface for writing help files for R functions and packages. It provides the ability to view and execute embedded R source code directly from the help file in the same manner as ESS normally handles code from a source file. `Rd` mode provides syntax highlighting and the ability to submit code to a running ESS process, either R or S-PLUS.

3.2 Emacs

Emacs (Stallman, 2000) is a mature, powerful, and easily extensible text editing system which while traditionally used under Unix, is freely available under the GNU General Public License for a large number of platforms, including Unix, Mac, and Windows. Emacs shares some features with word processors, and more importantly, shares many characteristics with operating systems. Most importantly, Emacs can interact with and control other programs either as subprocesses or as cooperating processes.

Recent versions of Emacs commands provide both a GUI and terminal interface which can respond to both keyboard and mouse usage. The mouse-based interface, through menus and toolbars (toolbars available currently with XEmacs), tries to facilitate the learning of keystroke-based short-cuts. User-defined additional menus and toolbars can be constructed as needed.

Emacs provides facilities that go beyond simple insertion and deletion: viewing two or more files at once; editing formatted text; visual comparison of two similar files; and navigation in units of characters, words, lines, sentences, paragraphs, and pages. Emacs knows the syntax of each programming language. It can provide automatic indentation of programs and highlight with fonts or colors specified syntactic characteristics. Emacs is extensible using a dialect of Lisp (Chassell, 1999; Graham, 1996). This means that new functions, with user interaction, can be written for common and repeated text-editing tasks.

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 appropriate to the programming language in use. In addition, editor behaviors such as folding, outlining, and bookmarks can assist with maneuvering around a file. Type-setting and word-processing, which focus on the presentation of a document, are tasks that are not pure text-editing. Emacs shares many features with word-processing programs and cooperates with document preparation systems such as (LaTeX; SGML, XML, and XSLT; and Noweb).

The capabilities can be extended in an orthogonal manner to include other Emacs packages for assisting with documentation as noted above; version control (RCS, CVS, SCCS, PRCS); and remote editing via FTP or secure transport mechanisms such as ssh and scp. Emacs handles the interface to both source code and transcripts contextually, providing syntax highlighting, bookmarking features, interfaces to directory structure, and command-history. 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. The Emacs keyboard and mouse interface can be re-mapped to resemble that of other text-editors, such as ed, vi, wordstar, and brief.

The above reasons suggest that Emacs is a reasonable starting choice for providing a universal interface for data analysis and programming. ESS extends Emacs to provide a functional, easily extensible and uniform interface for multiple statistical packages.

3.3 History of ESS

ESS is an example of how open-source products can continue to develop beyond what their initial authors planned. The ESS codebase was brought to life in 1989 initially as S-mode, to provide extensions to Emacs for editing S and S-PLUS files. This was originally written by Doug Bates, Ed Kademan, Frank Ritter, and Mike Meyer. David Smith was the next primary maintainer, and his most important contribution was to enhance the interface to the S-PLUS command-line interface and to allow for control of multiple processes simultaneously. In 1995, A.J. Rossini extended S-mode to support XEmacs, an open-source editor which was derived from Emacs. At the same time, extensions from ETH, written by Martin Mächler, were folded into what was to become the primary codebase. By 1996 a uniform S-mode, available for both Emacs and XEmacs, supported S, S-PLUS, and to some extent R. Kurt Hornik was instrumental in enhancing support for R, in particular providing Rd mode.

The other ancestor of the ESS codebase was a successful SAS-mode written by Tom Cook. The initial extension was done by A.J. Rossini in 1995 to work with XEmacs, and this provided the initial impetus to provide a uniform codebase for statistical programming.

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 (Cook, 1995), and the refactoring of the S-mode codebase to accommodate multiple languages in a flexible way.

During 1996 and 1997, Richard M. Heiberger further incorporated SAS-mode into ESS and designed the inferior ESS mode for SAS. In 1997, the grand redesign of the internals occurred, where by a generic means for configuring ESS for new statistical languages was implemented.

Most of the original work was done for Unix-based statistics packages. The first example of interprocess communication for Windows using DDE was constructed in 1998 by Brian Ripley. Based on this work, Richard M. Heiberger provided interfaces for Windows versions of S-PLUS.

In 1998, Rodney Sparapani designed the SAS batch interaction mode. Sparapani and Heiberger developed SAS batch support including function key behavior that followed SAS Institute's function key definitions, developed a more comprehensive and efficient syntax highlighting mechanism using the Emacs Font-lock mode, extended the `*ESS-elsewhere*` functionality to include SAS processes, and provided automated error message lookup in the log file. This provides a powerful development environment for SAS.

4 Using ESS with the S family of languages

ESS originated as S-mode and has historically provided strong support for the S languages. This section provides examples for editing files, communicating with the interactive S process, editing transcripts, and getting help.

4.1 Editing Files

`ESS[S]` is the mode for editing S language files. This mode handles proper indenting of multi-line code and functions (automatically when the code is entered and under user control while editing); color and font highlighting 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; the ability to switch between several inferior S processes (possibly on different machines); the ability to request help from an S process for variables and functions, and to have the help results sent into a separate buffer; and completion of object names and file names.

`ESS[S]` mode is automatically turned on for files with the extensions '`.R`', '`.r`', '`.S`', '`.s`', '`.ssc`', '`.q`'. An inferior process must be running to take advantage of the interactive features such as object- and file-name completion and help.

4.2 Inferior ESS processes

`iESS[S]` is the mode for interfacing with active S statistical processes. In effect, this mode replaces the S-PLUS or R Commands window with an active buffer running inside Emacs. This mode provides an interactive history mechanism; transcript recording and editing; the ability to resubmit the contents of a multi-line command to the executing process with a single keystroke. It also provides the same editing features as the editing mode `ESS[S]`.

The statistical processes S and R are started by using the Emacs function call `M-x S` or `M-x R`. This call opens up a buffer and starts the S process inside that buffer.

`iESS[S]` uses standard input/output to communicate with the S program, except with the GUI interface for S-PLUS 4 and 6 for Windows where standard input/output is not available. ESS currently provides two options on Windows. ESS can use the DDE (Dynamic Data Exchange) protocol to communicate with the GUI. The editing features, including the sending of commands to S-PLUS, are as

described above. The printed output does not come back to an Emacs buffer, but instead appears in the S-PLUS Commands Window. Transcripts must be physically copied to an Emacs buffer to get the transcript editing features. Full GUI capability is available by switching to the S-PLUS window. Or ESS can directly access Sqpe, the S-PLUS computing engine, using exactly the same procedures as are used for Unix. With this procedure interactive graphics and other GUI features are not available. In the near future, we anticipate merging these approaches for S-PLUS 6 by using the `connections` class of the new S4 engine. We will then have both interactive GUI and transcript recording and editing on the Windows platform.

One useful extension in ESS is relaxation of the requirement that the statistics program be available on the local machine. ESS provides transparent execution of a statistics package on a remote machine from within the local Emacs. The `S+elsewhere` commands assist with transparent execution of a statistics package on a remote machine. The idea of `S+elsewhere` is that we open a telnet, rlogin, or ssh connection to another machine from within a buffer called '`*S+elsewhere*`', and then run S on the other machine in that buffer. All the editing and interaction features described on the local machine work equally well on the remote machine. The relevant commands get sent through the buffer '`*S+elsewhere*`' to be executed by the program on the other machine. The interaction, including all the unique features of working with ESS, appears to the user exactly the same as if the program were running on the local machine. With X-windows running on the local machine, it is even possible to bring up visual displays from remote Unix systems on a Windows local machine.

4.3 Editing and Reusing Transcripts

ESS Transcript mode is designed for editing transcripts, constructed for example by saving an '`iESS`' buffer. This mode understands the form of S transcripts; it does color and font highlighting based on syntax; permits resubmitting multi-line commands to an active process buffer; has the ability to request

help from an S process for variables and functions; and the ability to switch between processes which would be the target of the resubmissions. The S language transcripts can be "cleaned" by finding all input lines and isolating them into a stand-alone input file that can be used as input to another S job or as a model for further program design.

4.4 Programming Language Help

There is a new menu, [ESS help], for viewing S help pages. In addition, additional keyboard shortcuts (using single keystrokes) are turned on, including:

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

s *x* move to help **s**ection code "*x*" where *x* is e.g. '`u`' for "Usage", or '`e`' for "Examples". You can see a full list by entering '`s ?`' from within an ESS Help buffer.

h fast '**h**yperlink' 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.

4.5 Philosophies for Combining S with ESS

There are two primary philosophies for using ESS. The first is that the source code (the file '`myfile.s`', for example) is real, and objects in S are temporary realizations of the source. This is the approach preferred by the current group of developers and the default when ESS is initially installed. A project is constructed by writing S language source code for *every* user defined function or data object which is considered as the primary means for generating the objects. The code is stored in one or more text files stored in a project directory or directories. The code is read into S for a temporary realization as an object. This approach allows for better portability, where the user might want to use the written code and

perform the same or related analyses on different versions of the S language. This approach also permits external version control for S language source code.

The second, deprecated view, is that S objects are real. The text version of the objects (source code, although we can't use that term with this philosophy) is a temporary realization of the objects. ESS is then used to edit, but not save, the objects. Dumped buffers (a text representation of the underlying S object) should not be saved in this philosophy. We strongly discourage this approach. It is a natural result of assuming that one will always use a single S process and that transportability is not important, assumptions we are not willing to make. Details for activating this configuration are in the documentation distributed with ESS.

5 Using ESS with SAS

`ESS[SAS]` mode is used for editing SAS files, initially based on the earlier SAS-mode (Cook, 1995). ESS provides two user interfaces for executing SAS programs. Both have similarities to the windowing interface that SAS itself provides. One is similar to batch processing of SAS programs and the other to the interactive behavior of SAS Display Manager. In addition, SAS files can be edited in `ESS[SAS]` mode in Emacs and then pasted into the SAS Display Manager Program Editor window. All three procedures can be used with SAS running either locally or remotely.

5.1 Editing SAS Files

`ESS[SAS]` is the mode for editing SAS language files. `ESS[SAS]` is automatically turned on when editing a file with a '`.sas`' suffix. `ESS[SAS]` mode provides all the flexibility of the Emacs editor for editing and searching in each file and for simultaneously working with multiple SAS files. For example, while writing a SAS program, a user may need to construct SAS macro files, consult other SAS programs/listings/logs ('`.sas`', '`.lst`', '`.log`') and/or non-SAS files documenting one or more SAS data sets.

ESS[SAS] mode provides the following services: proper indenting, generated by both [Tab] and [Return]; color-coded syntax highlighting of '.sas' and '.log'; the choice to submit the contents of an entire buffer, a highlighted region, or a single line to an inferior SAS process, if one is currently running; switching between processes which would be the target of the buffer (for the above); to save and submit a whole file or a highlighted region as a batch SAS process with a single keypress; to continue editing a SAS program while SAS is processing it; to kill the batch SAS process through the shell buffer; the capability of allowing the SAS process to continue after you exit Emacs; single keypress navigation of '.sas', '.log' and '.lst' files (in addition, '.log' and '.lst' files are automatically refreshed, and an automated error message search is performed for '.log'); single keypress viewing of SAS permanent data sets.

Preconceptions and Configuration. People using ESS and SAS together come from two incompatible starting positions. Some are Emacs users who are adding SAS to their repertoire, and expect the keyboard to behave as it does in any other Emacs mode, along with a few new keys to interact with SAS. Others are SAS users who have discovered the power of editing their SAS source files with Emacs. They expect the keyboard to behave as it does in SAS Display Manager and to gain powerful editing and interaction capabilities by using Emacs.

The two places where the keyboard must be configured are the [TAB] key and the function keys. The batch processing keypress commands use the same function keys as the SAS Display Manager.

5.2 SAS process interaction

There are three methods for interacting with SAS. We describe running SAS with the Display Manager, batch processing from ESS, and running SAS in an inferior ESS buffer.

The ESS methods for interacting with SAS are based on the SAS Display Manager's procedures. The two reasons to use ESS rather than the Display Manager are that one might prefer Emacs editing of

source, and ESS provides immediate access, through the Emacs search and edit commands, to the output in the listing file.

ESS and Display Manager It is possible to use ESS for editing SAS files with `ESS[SAS]` mode and then pick up the file or a region and drop it directly into the SAS Display Manager Program Editor window.

Batch Processing. Batch Processing works when Emacs is running on a local machine and SAS is running on either a local or remote machine. The local machine must be directly connected to the network, possibly by ppp. Even with the remote connection, it behaves exactly as if the SAS process were local. The remote behavior requires Emacs' remote-editing facilities, which are obtained by retrieving the file by '`/username@remote.machine.address:/directory-path/myfile.sas`'. When the file is submitted for batch processing, it will automatically be saved back to the remote machine and be exactly where the remote SAS will find it.

`iESS` Processing. ESS can run an interactive SAS session in several '`*shell*`' buffers, either locally on a Unix machine or remotely from any local machine to SAS running on a Unix host. Highlighted lines in the '`myfile.sas`' buffer are sent incrementally to the SAS session running in an `iESS[SAS]` buffer. The '`.log`' and '`.lst`' file information is received back incrementally in two other buffers. This process simulates SAS Display Manager behavior on text-only terminals over slow network connections, such as international internet access as well as telephone lines.

5.3 Design of `ESS[SAS]` mode

`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*' or 'myfile.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*' or 'myfile.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.

6 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, tool-tips, and interfacing cleaning. The other is to target appropriate potentially useful programming methodologies for transfer to statistical practice.

Literate Programming methodologies (Knuth, 1992; Ramsey, 1994) are a natural fit for statistical practice. We refer to the application to statistical analysis as Literate Statistical Practice (Rossini, 2001). The tools used are Noweb (Ramsey, 1994) and either LaTeX, HTML, or XML for documenting and explaining the analysis. This approach to programming encourages the use of a literary documentation style to explain the programming code for the data analysis. The program can then be extracted from the documentation text for realizing the statistical analysis.

Important IDE extensions which should be implemented in future versions include class browsers, analysis templates, tool-tips, and similar features. Class browsers can be thought of as a tree or outline

for presenting datasets, variables and functions in the context of what they represent; this allows for rapid and appropriate inspection. Analysis templates would allow statistics centers and groups to provide standardized templates for initiating an analysis. While most IDE features have been developed for object-oriented languages, the above also can apply to non-object oriented programming.

ESS is one of the first IDEs intended for statisticians. It provides an enhanced, powerful interface for efficient interactive data analysis and statistical programming. It is completely customizable to satisfy individual desires for interface styles as well as being extensible to additional statistical languages and analysis packages.

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.

Tom Cook. SAS mode for Emacs, 1995. `ftp://ftp.biostat.wisc.edu/pub/cook/sas-mode/sas.tar.gz`.

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

Richard M. Heiberger. Emacs speaks statistics: One interface — many programs. In Kurt Hornik and Friedrich Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing (DSC 2001)*. Technische Universität Wien, Vienna, Austria, 2001. `http://www.ci.tuwien.ac.at/Conferences/DSC.html`, ISSN 1609-395X.

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

Insightful Corp. S-plus statistical software: Release 6.0, 2001. Seattle, WA: MathSoft.

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

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

A.J. Rossini. Literate statistical practice. In Kurt Hornik and Friedrich Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing (DSC 2001)*. Technische Universität Wien, Vienna, Austria, 2001. `http://www.ci.tuwien.ac.at/Conferences/DSC.html`, ISSN 1609-395X.

A.J. Rossini, Martin Mächler, Kurt Hornik, Richard M. Heiberger, and Rodney Sparapani. ESS (emacs speaks statistics), 2001. `ftp://ess.stat.wisc.edu/pub/ESS/`.

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

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

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 Obtaining ESS

ESS (Rossini et al., 2001) 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. Unix users, but not Windows users, may choose to do an optional installation by executing the Makefile that comes with the ESS package.

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

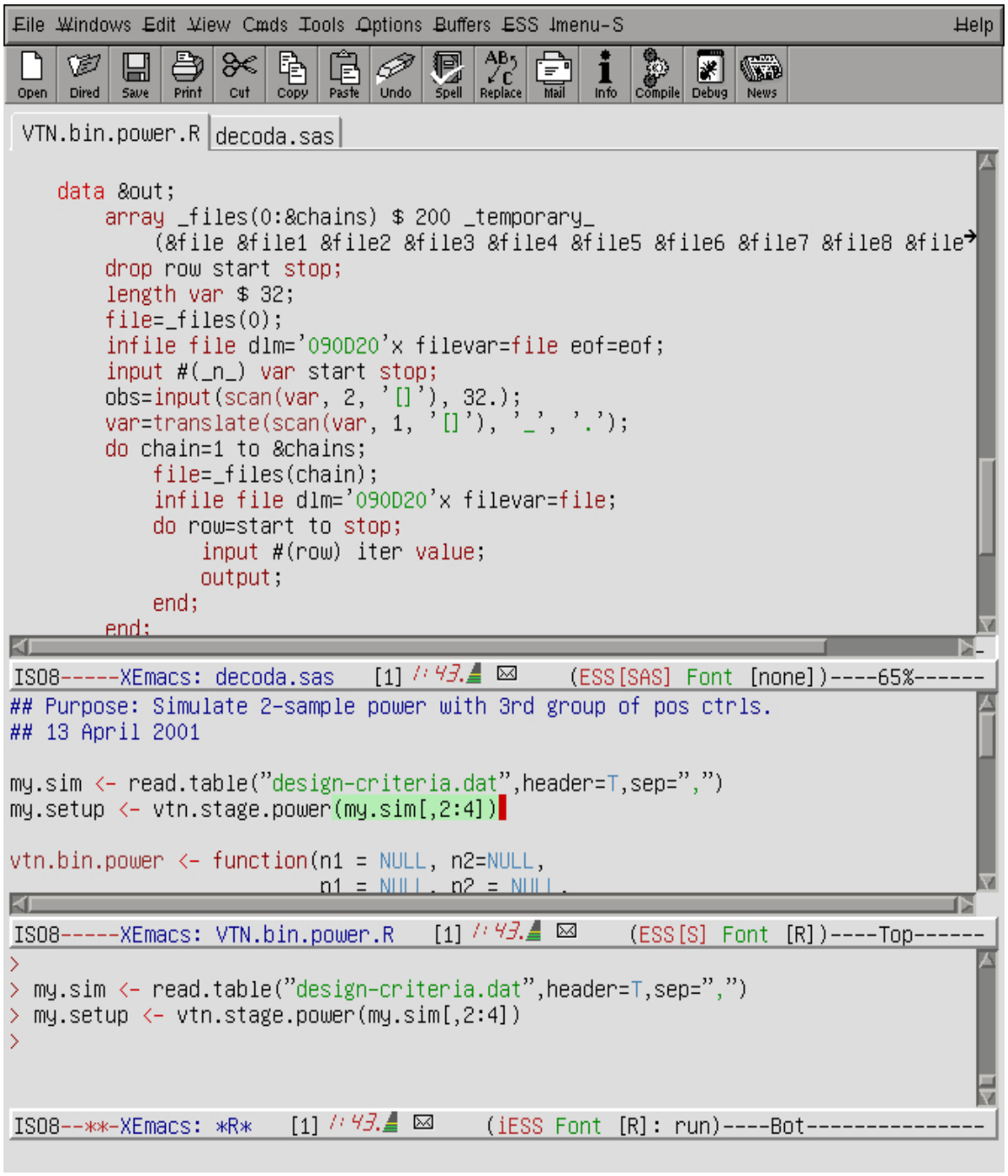

Figure 1: XEmacs and ESS editing SAS and R code and R evaluation on Unix

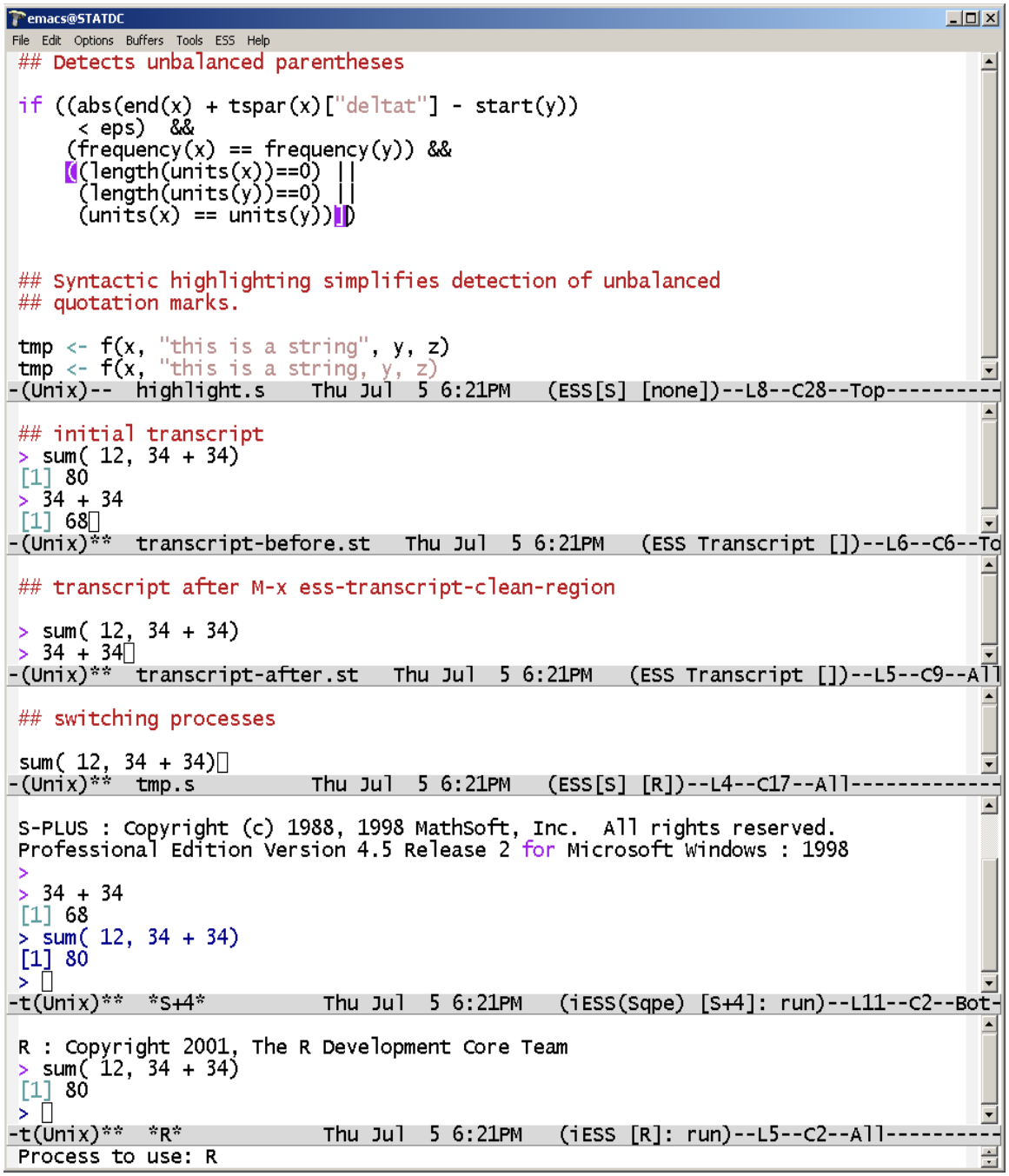

Figure 2: Emacs and ESS running R and S-PLUS on Microsoft Windows