Comment

C. Weihs

The book describes a software system called Lisp-Stat designed as a statistical software environment particularly for experimentation with new paradigms in interactive statistical graphics. Since other software systems well-known for supporting dynamic graphics, like ISP (1988), SAS-INSIGHT (1991), and (New) S-Plus (1990), suffer from the drawback of not allowing changes in the implemented graphical methods, there is a real need for systems like Lisp-Stat. Indeed, restricting oneself to an existing graphical framework, that is, suppressing ideas for new graphical tools, sometimes not only hinders creativity, but even prevents complete realization of adequate data representations, as we had to accept in our OMEGA-project (Online Multivariate Exploratory Graphical Analysis, see Weihs and Schmidli, 1990a, b). The OMEGA-system is realized in ISP, preventing the implementation of some of our graphical concepts adequately because of restrictions of ISP graphics. At the time the OMEGA-project was started, the most complete and flexible software environments for statistical graphics were the Data Viewer (see Buja, Asimov, Hurley and McDonald, 1988) and plot windows (see Stuetzle, 1988). Both systems were, like Lisp-Stat, based on Lisp, but both were only available on the exotic and expensive Symbolics Lisp machines. The hope that one could easily overcome all such restrictions with Lisp-Stat was the main motivation for me to work through the book. This hope was indeed fulfilled and itself made the book worth reading. But before discussing Lisp-Stat realizations of graphical tools thus far not implementable in OMEGA, I shall give a somewhat personal overview of the contents of the book.

The book starts with a very stimulating tutorial introducing Lisp-Stat's whole range of possible applications in a form that motivates further reading. The illustrated tools range from basic numerical and graphical operations like summary measures and histograms, boxplots and scatterplots, over the generation and modification of data items like lists, to standard dynamic graphics like spinning plots and scatterplot matrices with plot interaction and linking, and to a demonstration of how easy dynamic simulations can be implemented. Additionally, linear and nonlinear regression, maximization, maximum-likelihood estimation and approximate Bayesian computation are demonstrated. Also, the ease of implementing new functions and methods is concisely indicated.

After the tutorial, all chapters but the last describe the Lisp-Stat language by means of syntax explanation, illustrative code and more extensive examples, intended to be interesting in their own right. Chapters 3–5 present all the basic techniques and tools for programming with Lisp-Stat: the definition and usage of variables and functions, data types, data input and output, control structures, code-writing support, probability distributions and statistical and linear algebra functions. The examples include Newton's root finder, symbolic differentiation, a projection operator and robust regression.

So far, nothing special about Lisp-Stat. For example, the notion of a function is realized very similarly as in New S-Plus (1990), using functions not only in place of subroutines or macros, but also allowing functions to be handled as data, being thus able to specify, for example, the mean function of a nonlinear model very easily. Indeed, such "nonstandard" data have always had to be handled in statistical software allowing "real" nonlinear relationships. For example, one of the early systems realizing special data elements for model equations and model equation systems, the IAS-System Bonn, originated at the end of the 1960s (see Kirchen and Weihs, 1984), introduced such elements because of the need of econometric multi-equation models with nonlinearities in the variables as well as in the parameters to be estimated.

Chapter 6 describes something I had not really looked into before: object-oriented programming. The basic idea of object-oriented programming in Lisp-Stat is to build new numerical and graphical tools at the basis of certain *prototypes*. An object is built from a prototype or from another object by copying, by adding specific information like data in locations called *slots* and possibly by changing the methods to work on the data. Methods associated with an object are activated by sending the object a *message* identifying the method. Lisp-Stat contains a number of built-in prototypes delivered to support, among others, regression and the building of graphical windows, menus and dialogs. Nonlinear regression, for example, is supported by the *nreg-*

C. Weihs is a member of the Mathematical Applications unit at CIBA-GEIGY Ltd., R-1008.Z2.22 CH-4002 Basel, Switzerland.

model-proto, which is defined by means of the linear *regression-model-proto*, changing, for example, the method for the computation of estimates, and adding, among others, a slot for holding the model's mean function. Other methods are *inherited*, that is, copied from the *ancestor*. For example, the standard errors of the estimates, leverages and similar quantities are calculated by linearizing the nonlinear model around the estimated parameters. This is realized by defining the x-matrix *slot*, representing the values of the involved regressors in the linear case, to comprise the Jacobian in the estimated parameters in the nonlinear case, and inheriting from linear regression the computation of the above quantities using the x slot. Note that here inheritance realizes the analogies between nonlinear and linear regression analysis; in other cases specialization is realized. In order to start a nonlinear regression, one could build a new object, called an *instance*, from the nreg-model-proto adding the response data, mean-function, initial parameter guesses, and stopping criteria in corresponding slots, and then send the object the *compute*, *coef-estimates* and *coef-standard-errors* messages. However, calling instead the built-in *nreg-model* function with the same arguments would be more easy and more informative. Naturally, the nreg-model function itself was built on the basis of nreg-model-proto.

Chapters 7–10, that is, the rest of the book, describe Lisp-Stat's graphics system. To be more specific, those prototypes are described that Lisp-Stat offers to support the building of graphical windows, menus and dialogs. It is really fascinating to realize how easy window handling can be by using an appropriate set of objects able to respond to messages sent by some hardware specific *window-system* like the Macintosh Toolbox, the Sun View or the X11 system, which are supported by Lisp-Stat. The window-system monitors the user's actions upon windows, for example, moving or resizing windows, popping up windows, etc., and sends corresponding messages to objects affected by such *events*. This then initiates, for example, (re-)drawing the contents of the window or menu. Also, an object-oriented window-system can easily provide a library of standard graphics objects, greatly reducing the number of new methods to be defined when creating new graphical tools by inheriting from the most suitable object in the library. Moreover, new methods are only locally valid for the object they are defined for, that is, they do not influence other instances of the same object. This helps considerably in avoiding confusion.

Graphics windows and dialog windows share certain basic features concerning title, size, location and temporary and permanent removal. The corresponding methods are included in the *window-proto*. Graphical menus are handled separately. Their main properties are that menus are popped up by mouse-clicking a menu button, that they are displayed only as long as the mouse button is not released and that an action is executed corresponding to that *menu-item* highlighted by pointing on it by the cursor when the mouse button is released. The action is realized by sending that object corresponding to the highlighted menu-item the *do-action* message.

With dialog windows, the programmer can give the user the chance to send instructions to a program. There are two kinds of dialogs, one the user has to respond before the program can proceed (*modal dialogs*) and one the user can use very much like a menu appearing in a different form (*modeless dialogs*). Lisp-Stat is offering different standard dialogs of both kinds. A typical modal dialog is providing a message and is asking for OK, or is asking for a choice between different items provided. A typical modeless dialog provides a slider constructed to control, for example, the animation of a power transformation plot. If you want to build a more elaborate dialog, Lisp-Stat offers a flexible set of dialog items.

In order to judge the power of Lisp-Stat's menus and dialogs, let us discuss the construction of OMEGA's user interface in Lisp-Stat. Mainly the following tasks have to be practicable.

- The choice of one action or method out of a list, for example, of the implemented multivariate data analysis methods.
- The building of a subgroup from a list of variables or observations, for example, provided in a dataset.
- Decisions, for example, whether to build another subgroup or whether to store a result.
- Specifications of names, for example, of output data sets for predictions.

In principle all these tasks can be carried out by Lisp-Stat. However, there appear to be some difficulties. At first sight, the first task is a candidate for a menu, and the others are candidates for dialogs. However, OMEGA's menus often include explanations, that is, help texts, and Lisp-Stat's menus do not offer such a facility. But one can use dialog windows instead or write the help text into graphics windows and install a corresponding menu associated to the window. An example for using a modal dialog as a menu will be demonstrated in the

following (see Figure 1 for the graphical dialog):

```
setf text1 (send text-item-proto :new
  "ANALYSIS SELECTION"))
(setf text2 (send text-item-proto :new
  "CLASSIFICATION OF TECHNIQUES"))
(setf text3 (send text-item-proto :new
  "Analysis  |Pre-Information          |Representation of"))
(setf text4 (send text-item-proto :new
  "----------------------------------------------------"))
(setf text5 (send text-item-proto :new
  "PCA-COV|           -                |data variation"))
(setf text6 (send text-item-proto :new
  "PCA-COR|  Scaling unimportant |data variation"))
(setf text7 (send text-item-proto :new
  "CDA      |Object Classification   |class discrimination"))
(setf text8 (send text-item-proto :new
  "       "))
(setf c-item (send choice-item-proto :new
    (list "PCA-COV" "PCA-COR" "CDA" "quit") :value 0))
(defun collect-vals ()
  (send c-item :value))
(setf ok (send modal-button-proto :new "OK"
  :action #'collect-vals))
(setf meth-dialog
(send modal-dialog-proto :new
    (list
      (list text1) (list text2) (list text3) (list text4)
      (list text5) (list text6) (list text7)
      (list text8) (list c-item) (list ok))))
(send meth-dialog :modal-dialog)
```

After the definition of the help text (text1–text8), the choice buttons (c-item) are defined representing the analyses that can be carried out after the dialog. "Pressing" one of these buttons fixes the method. "Pressing" the OK button calls a function (collect-vals), the value of which is the index of the analysis chosen. Afterwards the dialog is removed from screen. The dialog itself (meth-dialog) is defined by using one list of items per line, and sending it the *modal-dialog* message.

Convincing though such a dialog may look, I am not quite certain that a user interface relying on

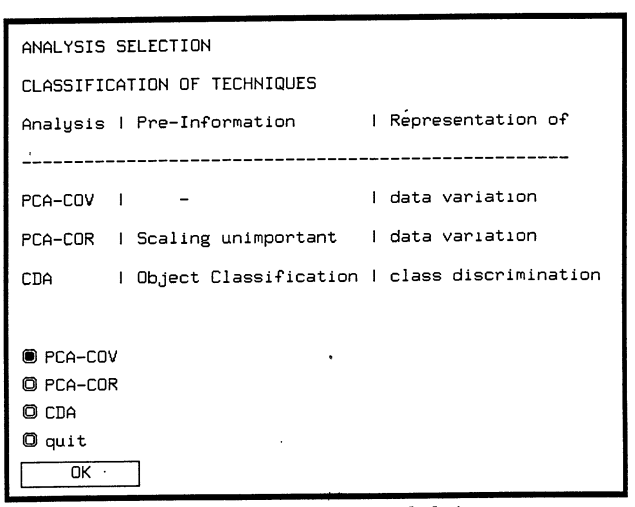

FIG. 1. *Dialog for method choice.*

menus and dialogs would really be an improvement to the line-oriented menus used up to now for OMEGA. For example, in OMEGA, subgroups of variables and observations are often built to be large connected parts of a big list so that many marks in a row would be necessary to identify such a group in a large Lisp-Stat dialog presenting one *toggle-item* for each variable or observation. Would it be not much easier to identify such subgroups by index lists in a response line allowing a sequence notation? Also, our menus asking the user to respond to a question in the next input line do not look that professional, but can also be run batch-like predefining (a part of) the sequence of input lines in a data set. This shortens some analyses considerably and helps with documentation. Using Lisp-Stat's menus and dialogs batch mode appears to be impossible.

Graphics windows inherit from *graph-window-proto*. In such windows, images are drawn by changing the color of the picture elements, the *pixels*. Lisp-Stat supports three drawing and animation modes. In *normal mode* and *XOR mode*, the image is constructed directly on the screen. In normal mode, the effect of drawing on a pixel is independent of its previous color. In XOR mode, coloring reverses the pixel's color. It is true that this is only well-defined in black and white, but even on color displays reversing the color twice in XOR mode leads back to the original color. Thus, if one wants to change the color only temporarily, one does not have to save it when using XOR mode. Lisp-Stat is using XOR mode for brushing and writing point labels. Using *double buffering mode*, flickering can be avoided in animation as long as the copying process from a buffer to the screen is sufficiently fast. With double buffering, the image is first composed off screen and then copied to the screen. This often results in a smoother and more flicker-free motion than with XOR mode.

Drawing in graphics windows can be carried out by sending messages to a window object by means of the interpreter. On the other hand, graphics windows are able to respond on mouse actions, key events and menu actions. Mouse events comprise *motion events* and *click events*. In the first case, an object is tracked by the mouse as it moves; in the second, the object is moved to the location of a mouse-click. A combination of the two approaches is *dragging*, that is, moving while holding down the mouse button. Moreover, different *cursor* symbols are available, like arrow, brush, hand and finger. Also, new cursors can be defined. Movement may also be realized by key events, that is, by hitting predefined keys, or by the window-associated menu. Among other things, such menus

and mouse or key events can be used for starting or stopping *idle* rotation or random walks. Idle actions are running continuously after starting, until they are explicitly stopped.

Obviously, for statistical graphics some more capabilities are needed, for example, connecting points by lines, choosing current variables, adding coordinate axes, centering and scaling and the application of linear transformations like translation and rotation. All this is included in the prototype *graph-proto* for displaying two-dimensional scatterplots in h-dimensional space. Standard mouse modes for statistical graphics are *selecting* and *brushing*. Selecting highlights the points under a rectangle by a mouse-click. Brushing highlights the points under a rectangle, called *brush*, moved with the mouse button down. In both cases, highlighting by default is transient, that is, it only lasts until the next click or until the brush is moved away from the point, correspondingly, but can be made permanent also. Moreover, Lisp-Stat allows the definition of new mouse modes, for example, for labeling a point near the cursor by mouse-clicking. Needless to say, different plots can be *linked* so that an action in one plot leads to corresponding actions in all linked plots. Default linkage is very loose and based on point indices, that is, linking causes the *point states* of observations with the same index to be equal. Point states are *invisible*, *normal*, *hilited*, *selected*. Other characteristics like symbols and colors are not linked. Also, observation indices have to be the same in all plots for the same objects. All these restrictions are overcome by an alternative linking procedure in the very last section of the book. This linking procedure relies on an observation based data representation also allowing linking for different sets of objects in different linked scatterplots. Except the *contents* (points, lines) of the plot, and the coordinate axes, graph windows also provide *margins* to hold plot controls, for example, buttons for fixing the rotation axis. For the window associated menu many useful standard menu items are offered.

Except for the basic 2D-scatterplot, prototypes are offered for scatterplot matrices, rotating plots, histograms and name lists providing a linkable list of point labels. This concludes the description of Lisp-Stat's graphics facilities. The book ends with very interesting examples of more involved applications of Lisp-Stat for developing graphical tools, including the choice of power transformations, plot interpolation, the choice of smoothing parameter, hand rotation, rotation around coordinate axes instead of screen axes, grand tours and parallel coordinate plots.

Judging from the occurrence, in the above contents overview, of so many of the terms central to all discussions of modern statistical graphics, one has the strong impression that Lisp-Stat offers everything we ever dreamt of. Even better, because of the capabilities of the language, it appears to be easy to start exploring new horizons. Let us see whether we would be able to overcome the ISP restrictions for the implementation of OMEGA: Could we realize "lasting control," nonstandard "viewport transformations," "dynamic plot interpolation," histograms linked to scatterplot matrices, linked "parallel views" with partly different sets of objects and "animated graphical testing"?

Lasting control initiates an action, for example, a rotation, which lasts until it is explicitly stopped. In Lisp-Stat this can simply be realized by sending the *idle-on* message to a plot, having defined the *do-idle* method correspondingly (see page 252).

The most important *viewport transformations* not realizable in ISP concern interactive scaling different in the two screen axes, for example, expanding in one direction, shrinking in the other with the same speed. In Lisp-Stat this can be realized by installing a corresponding button at the margin of a scatterplot which can be "pressed" by mouse-clicking (see page 286). Sending the *while-button-down* message to the plot (see page 249), mouse-clicking continuously activates the rescaling as long as the mouse-button is pressed. The actual scaling type has to be implemented in the *adjust-to-data* method (see page 270).

Dynamic plot interpolation is essentially realized in the book (see page 300), except that for OMEGA we are more interested in "convex" interpolation than in "trigonometric" interpolation. The changes should be obvious.

The most general multidimensional ISP views supporting dynamics and linking are rectangular scatterplot matrices. *Parallel views* mean any comparable scatterplots possibly not representable in a matrix structure. In Lisp-Stat any plots can be linked, thus also scatterplots with histograms (see page 46), and parallel views, for example, for comparing projections or predictions from different multivariate analyses.

OMEGA's p%-resampling repetitively uses random p% of the original sample for constructing projections of this subsample and predictions of the rest of the sample. Parallel views for comparing projections from different repetitions, that is, for studying the stability of the projections, thus include partly different sets of objects. In Lisp-Stat p%-resampling can be realized using the *sample* function (see page 31), and the alternative linking strategy (see page 334) may be used to compare the projections from different samples. A projection

value may be stored in an observation object with the index of the corresponding original observation using a variable name reflecting the resampling repetition number. If an observation is not in the p%-sample of this repetition, its value for this variable should be set to "missing." A possible code for "missing" is *nil*. Unfortunately, this code is not accepted by Lisp-Stat's plotting functions. A scatterplot of the projections thus has to be defined by including only those observation objects defining the p%-sample. The linking between plots resulting from different repetitions is guaranteed, since the linking procedure is relying on observation objects not on indices of observations in the different plots.

Let us finish with a small Lisp-Stat program realizing an animated graphical test for independence of two variables. To this end the y-values of a scatterplot are randomly permuted against the x-values as long as the mouse button is held down:

```
(def vx (uniform-rand 50))
(def vy (normal-rand 50))
(def indplot (plot-points vx vy))
(defmeth indplot :do-click (x y m1 m2)
  (flet ((permuty (x y)
           (def wy (sample vy (length vy)))
           (send self :clear :draw nil)
           (send self :add-points (list vx wy))))
    (send self :while-button-down #'permuty nil)))
```

The variables vx and vy comprise 50 uniform and normal random numbers, respectively. After the definition of the plot-object (indplot) for these variables, the method (do-click) is defined, which is continuously activated with a mouse-click inside the plot as long as the mouse button is pressed (while-button-down). The method ignores the location of the mouse-click (x y), but permutes the y-observations by drawing a sample (without replacement) of the same size as the original number of observations (permuty). Just before redrawing (add-points) the plot is cleared. Thus, if you press any mouse button anywhere inside the plot, you will see a sequence of permutations of the original plot as long as the mouse button is not released.

Altogether, using Lisp-Stat we could indeed overcome all the restrictions we had to accept up to now in developing our OMEGA-pipeline at CIBA-GEIGY. This is particularly true since Lisp-Stat is available for our computer environment, Apollo workstations equipped with the X11 window-system. On the other hand, as a classically trained programmer, being at home with Fortran and with matrix languages like ISP, I have to confess that I had difficulties in getting used to this object-oriented language. Possibly it is the language itself, for example, the extensive use of bracketing, or the style the book is written in, but I was never quite happy reading the book and writing test programs. Reading the book I had particular problems with the organization of the nontutorial chapters. Indeed, the author always tries to be as complete as possible in describing the language at the price of sometimes losing a reader who impatiently wants to learn about the next capability offered by Lisp-Stat. For me it would have been much more adequate if the book would have been divided into two parts: an extensive tutorial, demonstrating the use of all the main features of the language, possibly with reference to useful extensions described in the second part, a complete commented syntax reference, which would be much easier to use for a professional than the partly hidden syntax descriptions in the book and which could be avoided by someone who is just interested in an overview.

Nevertheless, Lisp-Stat offers a fascinating environment for statistical computing and particularly for dynamic graphics. I really hope that we hear about many new useful graphical tools developed by means of Lisp-Stat.