Dynamic and Interactive Graphics in Lisp-Stat

Luke Tierney[*]

Abstract
This paper describes the programmable dynamic and interactive graphics features available in the Lisp-Stat framework and discusses some of the history and motivations behind the design and development of the framework.

Key Words: Linked brushing, parallel coordinate plots, grand tours, computing environments

1. Background

The Lisp-Stat system was designed and developed in late 1980s and early 1990s. The goals of this work were to make available basic interactive and dynamic graphics to researchers and practitioners, and to support experimentation and development of new interactive graphical methods.

My first exposure to dynamic and interactive graphics was in work of Becker and Cleveland on linked brushing in a scatterplot matrix shown in the exhibits area at a JSM in the mid 1980s. This work is described in Becker and Cleveland (1987). The hardware used

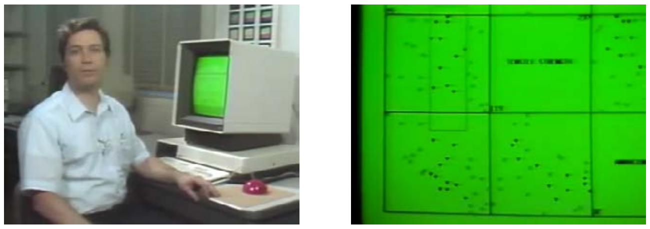


Figure 1: Rick Becker demonstrating linked brushing of a scatterplot matrix, from the ASA Statistical Computing and Statistical Graphics Video Library.

was the ATT Teletype Model 5620 (BLIT) as seen in Figure 1. Other efforts at this time used Lisp Machines or high-end Unix workstations, all out of my price range.

The Apple Macintosh had become available recently and was a more cost-effective option. My initial efforts involved developing two simple, stand-alone Macintosh applications for scatterplot brushing and point cloud rotation. Stand-alone tools need external tools for data preparation. The S language, available to a limited number of universities, provided an excellent integrated framework for data analysis and static graphics, and was of course used by Becker and Cleveland in their work. Something similar was needed to support my explorations in dynamic graphics.

An open source Lisp framework was a convenient choice. I used the XLISP implementation from David Betz, with added Common Lisp (Steele, 1984) features. Some useful features of Lisp include good support for a functional programming style and a macro system for adding new syntax. Lisp is also easy to modify to support vectorized operations, provides good support for developing new object systems, and defines an excellent condition and exception handling framework (Pitman, 2001).

[*]Department of Statistics & Actuarial Science University of Iowa

2. Basic Design and Usage

Like the S language, Lisp-Stat uses a command line interface (CLI) for interactively expressing computations. The command line interpreter is integrated with interactive graphics event processing. A prototype-based object system was developed to represent both graphics and statistical models. The object system supports multiple inheritance to allow a *mixin* style of programming.

The Lisp language uses a simple prefix function call syntax:

```
(log x)
(+ 1 2)
(* (log x) 2)
```

Arithmetic operators do not have a special syntax and are used like any other function.

In Lisp-Stat top level variables are defined using the `def` macro:

```
(def abrasion-loss (list 372 206 175 ...))
```

Some simple summaries are available, such as

```
(mean abrasion-loss)
(median abrasion-loss)
```

as are a number of basic plots. The supported basic plot types (and the functions to create them) are histograms (`histogram`), scatterplots (`plot-points`), 3D point clouds (`spin-plot`), and scatterplot matrices (`scatterplot-matrix`). For example, a scatterplot and a histogram can be created as

```
(plot-points abrasion-loss tensile-strength)
(histogram hardness)
```

The resulting plots are shown in Figure 2.

All plots support a set of basic interactions: identification, selection/brushing, adjusting selection color/symbol, and linking multiple plots. The scatterplot and histogram in Figure 2 are linked so that the points corresponding to the selected lower range in the histogram are also selected in the scatterplot. This shows the conditional distribution of the two variables in the scatterplot given the selected range of the variable in the histogram.

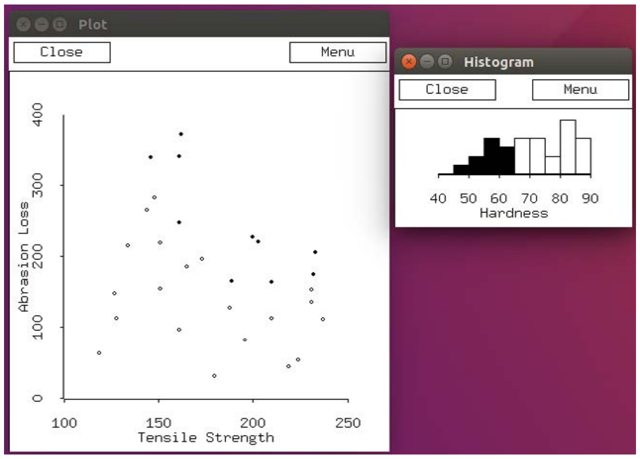


Figure 2: Linked histogram and scatterplot.

All interactive operations can also be done programmatically. For example, these expressions read or set the indices of the currently selected points in a plot `p`:

```
(send p :selection)
(send p :selection (< hardness 70))
```

To ease interaction and programming each plot has its own window and menu. Each plot represents a view on p-dimensional space and supports linear transformations of this space. The dimensions visualized are typically the first dimensions of the transformed space.

3. Custom Animations

The ability to program the system with the Lisp language allows the creation of custom animations in which a parameter is changed through a graphical interface and a plot is updated with the results of a new computation. A simple example, shown in Figure 3, is an animated kernel density estimate in which the slider adjusts the bandwidth and the resulting density estimate plot is updated.

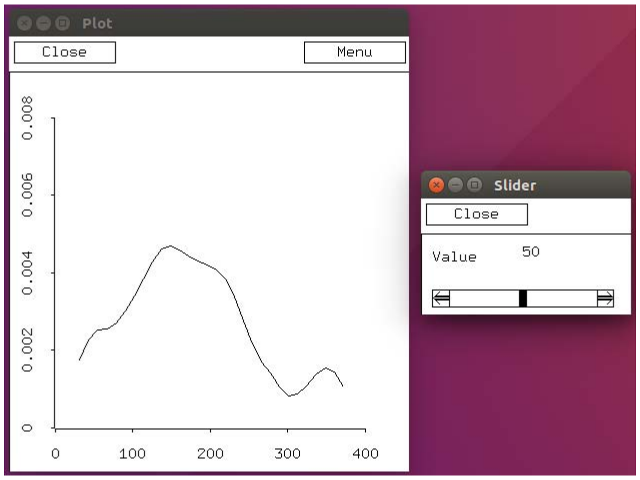


Figure 3: Kernel density estimate plot with the bandwidth controlled by a slider.

The code needed to implement this is very short and is shown in Figure 4. Slider controls can also be incorporated into a plot window as *overlays*.

```
(let* ((s (rseq 20 80 31))
       (p (plot-lines (kernel-dens abrasion-loss
                                   :points 30 :width (first s))))
       (d (sequence-slider-dialog
           s
           :action
           #'(lambda (w)
               (send p :clear :draw nil)
               (send p :add-lines
                     (kernel-dens abrasion-loss :points 30 :width w))
               (pause 2)))))
  (send p :add-subordinate d))
```

Figure 4: Code to implement the kernel density plot with a slider-controlled bandwidth.

4. Customized Interaction

It is also possible to create new modes of interactions by defining a new *mouse mode*, or to customize interactions by overriding some of the methods in the interaction protocol. The *hand rotate* mode for spin plots is an example of a new mouse mode defined in about 30 lines of Lisp code.

An example of a useful customized interaction is to enhance the linked plot of Figure 2 by adding a smooth curve through any highlighted points in the scatterplot. The result is shown in Figure 5. This is accomplished defining a custom `:adjust-screen` method for the scatterplot. The code for this method is again quite short and is shown in Figure 6.

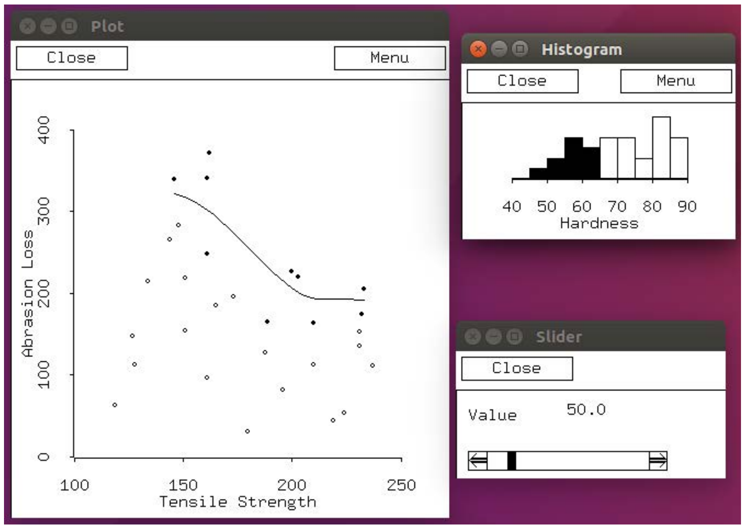


Figure 5: Linked plots with a smooth curve through any highlighted or selected points in the scatterplot.

```
(defmeth p :adjust-screen ()
   (call-next-method)
   (let ((i (union (send self :points-selected)
                   (send self :points-hilited))))
     (send self :clear-lines :draw nil)
     (if (< 1 (length i))
         (let ((x (select x-var i))
               (y (select y-var i))
               (w (send self :kernel-width)))
           (send self :add-lines (kernel-smooth x y :width w)))
       (send self :redraw-content))))
```

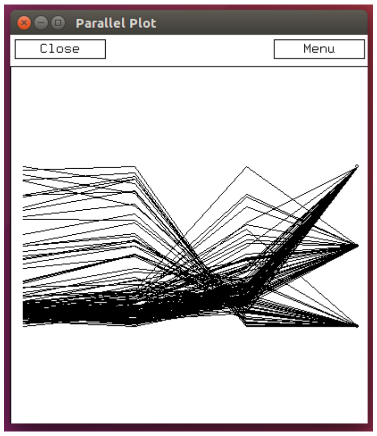
Figure 6: Code for the `:adjust-screen` method to add a smooth curve through any highlighted or selected points in a scatterplot.

5. New Plot Types

New types of plots can be added to the system by defining new plot prototypes. A simple example is a parallel coordinates plot, which can be derived from on the standard scatterplot prototype. Another example is a plot implementing a *grand tour* (Asimov, 1985). The grand tour uses a sequence of rotations and projections to explore a higher dimensional data set. One dimensional projections can be rendered with a histogram, two dimensional ones with a scatterplot, and three dimensional ones with a spin plot that shows a third dimension with depth queueing. The transformation process can be conceptually separated from the projection and rendering process, and this separation can be captured in software using multiple inheritance and a *mixin* style of programming. A grand tour mixin prototype can be created to manage the touring transformation of the p-dimensional data matrix. A standard tour plot of the type implemented in GGobi (Cook and Swayne, 2007) can then be implemented by combining the tour mixin with a spin plot. Combining the tour mixin with a parallel coordinates plot produces a parallel coordinates grand tour (Tierney 1990, Exercise 10.15; Wegman 1992). Figure 7 shows the code needed to create this combination, and Figure 8 shows two views of such a parallel coordinates tour.

```
(defproto parallel-tour-proto '(angle) ()
  (list tour-mixin parallel-plot-proto))

(defmeth parallel-tour-proto :angle (&optional (val nil set))
  (when set (setf (slot-value 'angle) val))
  (slot-value 'angle))

(send parallel-tour-proto :angle .1)

(defmeth parallel-tour-proto :num-tour-variables ()
  (- (send self :num-variables) 1))

(send parallel-tour-proto :slot-value 'scale-type 'variable)

(defun tour-parallel-plot (data &rest args &key point-labels)
  (let ((graph (apply #'send parallel-tour-proto :new (length data) args)))
    (if point-labels
        (send graph :add-points data :point-labels point-labels :draw nil)
        (send graph :add-points data :draw nil))
    (send graph :adjust-to-data :draw nil)
    graph))
```

Figure 7: Code to define a parallel coordinates grand tour prototype in terms of separate parallel coordinates plot and tour mixin prototypes.



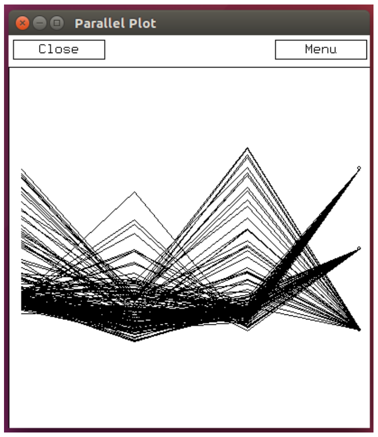


Figure 8: Two views of a parallel coordinates tour of a four variable data set with the fourth, categorical, variable not participating in the tour.

6. Discussion

Some limitations of the computing environments of the time affected the design of the Lisp-Stat system. At the time there were a wide range of graphical user interfaces available on workstations and personal computers, with a range of different conventions. Systems used one, two, or three button mice. Menus were attached to a global menu bar, a menu bar for each window, or a mouse button. While there are still differences between the Mac OS, Windows, and Linux interface conventions, the situation is now much more settled. To allow Lisp-Stat code to be written to work under all supported user interfaces required identifying a reasonable subset of functionality and an abstraction that could be implemented on all platforms. One example is the convention that each graphics window has a single menu associated with it; this menu is then made available according to the user interface conventions of the current OS.

Hardware limitations in capabilities and performance also had an impact. When Lisp-Stat was started on an early Apple Macintosh, color support was not available. Once color support was added, only a very limited number of colors were sometimes available. Lisp-Stat's very limited color support reflects this history. Limited speed of early systems also meant that animations, such as point cloud rotations, could be drawn one frame after another without the need for frame rate control. Faster computers mean frame rate control is needed even for larger data sets. One of the few changes made to the Lisp-Stat source code in recent years is the addition of pauses to control the frame rate.

Other design decisions were based on easing programmability of the system. One example is the decision to give each plot its own top level window. This makes it easy to write code that manipulates both the containing window and its contents, but it precludes the creation of visualizations consisting of multiple plots in a window, sometimes referred to as dashboards. A more flexible design might have allowed for the one-to-one correspondence of plot and window as the default but allowed the two to be decoupled for more flexible arrangements.

Experience with the Lisp-Stat system has amplified the lessons from the S and R frameworks about the benefit of integrating a powerful command line language system with a framework for data visualization. In the context of interactive graphics it is very useful to be able to switch seamlessly between graphical interactions and explorations using tools available in the language framework. The recently developed *Shiny* framework for R allows many interesting interactive visualizations to be constructed within a web browser, but the current design of requiring R to run a server and thus become unavailable for command line interaction prevents this seamless integration. Other technologies such as *plotly* or *D3* that rely on the web browser's *JavaScript* engine for managing interaction with visualizations remove this limitation, but do not support implementing new interactions using statistical functionality available in the command line language. Hopefully these shortcoming will be addressed in the future.

Lisp-stat can be compiled on current Mac OS and Linux systems. I have not tested it in recent Windows systems. It may work there as well. The source code is available at `http://www.stat.uiowa.edu/~luke/xls/xlispstat/current/`

REFERENCES

Asimov, D. (1985), "The grand tour: a tool for viewing multidimensional data," *SIAM Journal on Scientific and Statistical Computing*, 6(1), 128–143.

Becker, R. and Cleveland, W. S. (1987), "Brushing scatterplots" *Technometrics*, 29(2), 127–142.

Michael Bostock, Vadim Ogievetsky, Jeffrey Heer (2011), "D3: Data-Driven Documents," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis).*

Chang, W., Cheng, J., Allaire, J., Xie, Y., & McPherson, J. (2017), `shiny`: web application framework for R. R package version 1.0.0, `https://CRAN.R-project.org/package=shiny`.

Cook, D., & Swayne, D. F. (2007), *Interactive and Dynamic Graphics for Data Analysis: with R and GGobi*. New York: Springer Science & Business Media.

Pitman, K. M. (2001), "Condition handling in the Lisp language family," in *Advances in Exception Handling Techniques*, 39–59), Berlin and Heidelberg: Springer.

R Core Team (2017), "R: A language and environment for statistical computing," *R Foundation for Statistical Computing*, Vienna, Austria, `https://www.R-project.org/`.

Carson Sievert, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec and Pedro Despouy (2017), "`plotly`: Create Interactive Web Graphics via `plotly.js`," R package version 4.6.0, `https://CRAN.R-project.org/package=plotly`.

Steele Jr, Guy L. (1984) *Common LISP: the Language*, Digital Press.

L. Tierney (1990), *LISP-STAT: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*, New York: Wiley.

Wegman, E. J. (1992), "The grand tour in k-dimensions," in *Computing Science and Statistics*, 127–136, New York: Springer.