Dynamic and Interactive Graphics in Lisp-Stat

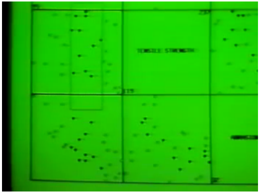
Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

July 31, 2017

Background

- This talk describes work in the late 1980s and 1990s for
  - making available basic interactive and dynamic graphics;
  - supporting experimentation and development of new methods.
- My first exposure to dynamic and interactive graphics was in work of Becker and Cleveland on linked brushing in a scatterplot matrix



- The ideas were described in “Brushing scatterplots” (Becker and Cleveland, 1987, Technometrics)

Background

- The hardware used was the ATT Teletype Model 5620 (BLIT)
- 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 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.
- Something similar was needed to support dynamic graphics.

Background

- An open source Lisp framework was a convenient choice.
- I used the XLISP implementation from David Betz, with added Common Lisp features.
- Some useful features of Lisp:
  - Supports a functional programming style;
  - Macro system for adding new syntax;
  - Easy to modify to support vectorized operations.
  - Easy to develop new object systems.
  - A good exceptional condition handling system;

Some Basic Design Features

- A command line interface (CLI) for interactively expressing computations;
- Integrating the command line with interactive graphics event processing.
- Prototype-based object system for graphics and models.
  - Multiple inheritance to support *mixin* style of programming.
- Plots represent views on p-dimensional space.
  - Support linear transformations of space.
- Each plot has its own window/menu.

Basic Usage

- Lisp prefix function call syntax:
```
(log x)
(+ 1 2)
(* (log x) 2)
```
- Defining a variable:
```
(def abrasion-loss (list 372 206 175 ...))
```
- Summaries:
```
(mean abrasion-loss)
(median abrasion-loss)
```
- Some plots:
```
(plot-points abrasion-loss tensile-strength)
(histogram hardness)
```

Basic Dynamic Graphics

- Standard plot objects:
  - histogram – histogram
  - scatterplot – plot-points
  - 3D point cloud – spin-plot
  - scatterplot matrix – scatterplot-matrix
- Standard interactions:
  - identification
  - selection/brushing
  - adjusting selection color/symbol
  - linking multiple plots
- Interactive operations can also be done programmatically.

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

Custom Animations

- Kernel density estimate with a slider control:

```
(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))
```

- Slider controls can also be incorporated into a plot as *overlays*.

Customized Interaction

- New interactions can be created by defining a new *mouse mode*.
- The *hand rotate* mode for spin plots is defined in about 30 lines of code.
- Response to changes in linked plots can be customized by defining a custom :adjust-screen method.
- A method to fit a smooth line to the currently highlighted or selected points in a 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))))
```

New Plot Types

- New plot types can be created as new prototypes.
- A simple example is a parallel coordinates plot.
- Prototypes can inherit from one or more prototypes.
- This supports a *mixin* style of design.
- A grand tour mixin can be created to change the transformation of the p-dimensional data matrix according to a touring algorithm.
- A standard tour plot can be constructed from this mixin and a spin plot.
- A parallel coordinates tour can also be built fro the tour mixin and the parallel coordinates plot.

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))
```

Discussion

- Some historical constraints:
  - unsettled user interface conventions (mouse buttons, menus, ...);
  - limited color range;
  - speed.
- Some design decisions:
  - new window for every plot;
  - one plot per window
- Some lessons:
  - integration with a powerful language CLI is very valuable;
  - creating a good set of software bulding blocks is very helpful;
  - being able to switch between language CLI and interaction is very useful (current limitation of shiny approaches);
  - programming callbacks in language is helpful (current limitation of *JavaScript* approaches)

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

http://www.stat.uiowa.edu/~luke/xls/xlispstat/current/