Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

OBJECT-ORIENTED GRAPHICAL ANALYSIS

Catherine Hurley

Department of Statistics/Computer & Information Systems
George Washington University
Washington, D.C. 20052

ABSTRACT

Object-oriented techniques have long been used for building simulation models, and more recently for animating the simulation as it progresses. This paper describes how object-oriented techniques may be used in the output analysis phase of a simulation, using a system designed for highly-interactive, graphical data analysis.

1 INTRODUCTION

Object-oriented programming methods have been popularized by modern graphical user interfaces. The design of statistical software which takes advantage of such techniques has been the subject of recent research (McDonald 1986, McDonald 1988, Oldford and Peters 1987, Tierney 1990, Pedersen 1991) Emphasis has been placed on software for exploratory data analysis by means of highly-interactive and dynamic graphics (Stuetzle 1987, Buja et al. 1988, Hurley 1987, Buja and Hurley 1990, McDonald, Stuetzle and Buja 1990, Hurley and Oldford 1991). The volume by Cleveland and McGill (1988) is a collection of papers giving a good introduction to the subject.

Modern systems for statistical graphics are

- **Interactive:** plots respond to user input, using a command-style or mouse-driven interface. A good example is point identification, where the analyst uses the mouse cursor to select a point in a scatterplot, causing a label identifying the associated case to be printed.

- **Dynamic:** plots may change over time. Examples are point cloud rotations, where the coordinates of points in a scatterplot move continously over time, and scatterplot brushing where the colors of the points change (see, for example Becker, Cleveland and Wilks 1987, Stuetzle 1987). Dynamic plots are typically interactive, because plot changes are most usefully controlled by the analyst.

- **Extensible:** Fewer systems are programmable, like S (Becker, Cleveland and Wilks 1988) or LispStat (Tierney 1990). Because of their inheritance properties, systems based on object-oriented languages should be particularly easy to extend.

This paper will briefly describe a new object-oriented system for graphical data analysis, and discuss applications to analysis of simulation data. A more complete description of the system is found in Hurley and Oldford (1991). Our implementation is in Common Lisp (Steele 1989) and CLOS (Steele 1989, Keene 1988), an object-oriented extension of Lisp. The graphics package is part of the QUAIL system (for QUantitative Analysis In Lisp) (Oldford et al 1991), available soon for a variety of workstations.

2 OBJECT-ORIENTED PLOTS

Statistical plots are collections of objects such as points, lines, labels and axes. In general, these objects are arranged in a hierarchy- a scatterplot consists of axes, label and a pointcloud which itself consists of points (see Figure 1). Similarly a scatterplot matrix consists of pointclouds and labels, though arranged in a different format. An object appearing in the plot has an associated piece of statistical data, for the scatterplot it's the entire dataset, for a point it's typically a case and for the pointcloud the collection of cases. Each component of a plot we term a *view*, so-called because it provides a graphical representation of some piece of data, called the *viewed object*. A view object contains a reference to its viewed object, and an image of a view is used as a graphical interface to the viewed object. *Simple views* are views like point symbols, axes and labels which do not contain subviews, all other views are *compound views*.

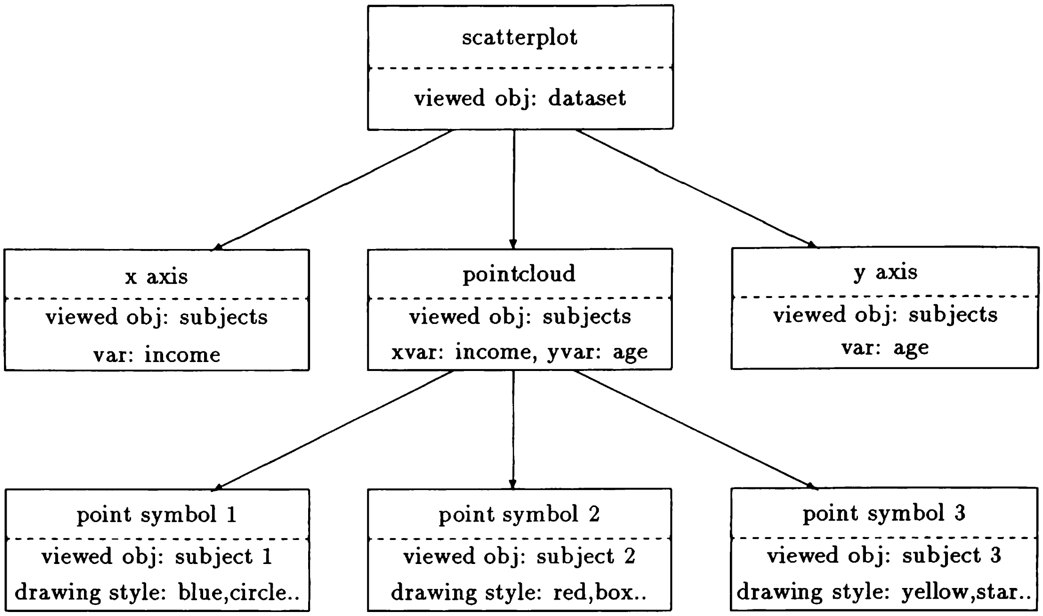

Figure 1: Scatterplot hierarchy

Figure 1 shows the hierarchy for a scatterplot with two axes, and a pointcloud containing three points. In each rectangle, the items listed below the dashed line are some of the attributes (slot names and values) of the view- the viewed object, drawing styles for point symbols and variables for the axes and pointcloud.

2.1 Drawing a Plot

In the terminology of object-oriented programming, there is a `draw-view` generic function (or message) which when applied to a view results in an image appearing on the screen. Compound views are drawn by looping over their immediate subviews and recursively invoking the `draw-view` function on each. As a consequence of the inheritance properties of object-oriented languages, a single `draw-view` method works for all compound views. Simple views are at the bottom of a plot hierarchy and must be drawn explicitly, using the drawing style parameters. Therefore a `draw-view` method must be provided for each class (type) of simple view.

2.2 Constructing a Plot

To construct a scatterplot, the user invokes the `scat-plot` function on the data. A scatterplot object is created, which itself constructs the appropriate subviews (labels, axes and a pointcloud). Then the pointcloud in turn constructs its subviews, the point symbols. View construction is the primary occasion on which information is passed from view to subview-parameters provided by the user are used to initialize the views.

2.3 User Interfaces

Each plot appears in a window on the screen, and every view in a plot is mouse-sensitive. When the user clicks on a window, the 'closest' (in some sense) view is selected and handles the click. Typically the view pops-up a menu, the analyst makes a selection, and the selected operation is invoked on the selected view. The next section demonstrates the variety of operations provided. Note that a view handles mouse-input in the same way regardless of where it appears– a histogram axis offers the same menu choices as does a scatterplot axis, guaranteeing a consistent, easy-to-use interface. Of couse, operations can also be invoked using a traditional command-style interface.

2.4 Using the Toolkit

Since the plotting package is intended to be a *toolkit* for statistical graphics, it is extendible and programmable . New kinds of plots may be built without programming by overlaying and superimposing existing types of views. The programmer can construct new view classes, inheriting from existing views, and/or using them as subviews.

3 APPLICATIONS

The first section illustrates the capabilities of the plot package with a graphical analysis of a simulation experiment. The second section describes interactive techniques for analyzing time-varying response variables.

3.1 A Multi-Factor Experiment

3.1.1 The Experiment

A simulation experiment was conducted to evaluate a new method for variable selection in linear regression (see Thall, Simon and Grier, 1991). The new method used cross-validation in an effort to beat existing methods in situations with considerable non-informative predictor (noise) variables. Four factors were varied as follows, with 1000 replications.

(i) algorithm (TYPE) : cross-validation (CV), stepwise (STEP) and backwards elimination (BACK).

(ii) sample size (N) = 50,100,200

(iii) number of noise variables, (#NOISE), 5,10,20

(iv) correlation of noise variables (CORR), 0 or 0.75.

The response variables were

(i) percent of replications with correct variable selection (%CORRECT).

(ii) percent of replications with zero noise variable selection (0-NOISE).

(iii) average squared error of the fitted values (MSE).

To compare the response variables, I constructed a scatterplot matrix (Figure 2). A second plot (not shown) separating the observations in each block was constructed and *linked* to the scatterplot matrix. In this way I could easily color groups of points in one plot and see the groups color in associated plots. It seems as both CV and BACK groups have a pair of outliers; inspection of the data uncovers a transcription error which exchanged %CORRECT coordinates. Figure 3 shows the corrected data. CV observations separate clearly from the others for both %CORRECT and 0-NOISE. The four remaining clusters separate by TYPE and N.

As one might anticipate, sample size is the main contributor to variation in MSE. Note also that in plots involving MSE, points lie in pairs. Selecting the points for inspection of the associated observations shows that the pairs differ only by CORR. The structure of CV observations seems quite different from the others; either group may be plotted separately by highlighting the points and requesting a blow-up plot.

Now we turn to a more detailed inspection of individual response variables. Figure 4 shows MSE on the y-axis, with observations on the x-axis grouped first by TYPE, and within TYPE by #NOISE. Grayscale color was used to separate by N, and shape for CORR. The plot has a very clear pattern, with CV a clear winner for #NOISE= 20, and correlated predictors having a negative impact on the fits.

We select a change variable operation from the plot menu to switch to %CORRECT on the y-axis. Here the points do not separate clearly by N, so each sample size group was examined in turn by making other groups invisible. Figure 5 shows the plot for N = 100. CV is the clear winner, with BACK beating STEP. %CORRECT decreases as more noise variables are present; CORR is not a contributing factor. These patterns are consistent over N values, and are similar for 0-NOISE.

To summarize, the graphical analysis demonstrates that the new CV algorithm is a big improvement over the popular BACK and STEP methods, in the presence of large numbers of noise variables.

3.2 Time Plots

Time traces are frequently used in discrete-event simulation to show how a response variable changes over time. Here I point out some of the advantages of building interaction into such displays.

- **Controlling the time interval.** In a simulation a variable trace produces a long series, too long to view in a fixed-frame window. With a scrolling window, one could easily scan the entire series by moving the time interval. However more sophisticated controls are necessary to expand and contract the visible time-interval, adjusting the aspect ratio. See Buja et al. (1988) for some interesting examples.

- One way of conducting inference on a variable traced over time attempts to induce approximate normality and independence by batching the raw observations. This may be done graphically by constructing a *slider* which allows the user to manipulate the number of batches. As the batching parameter is changed the time-plot updates automatically to show the new batch means. A normal QQ-plot and an auto-correlation plot could also be controlled by the slider, allowing graphical assessment of normality and independence.

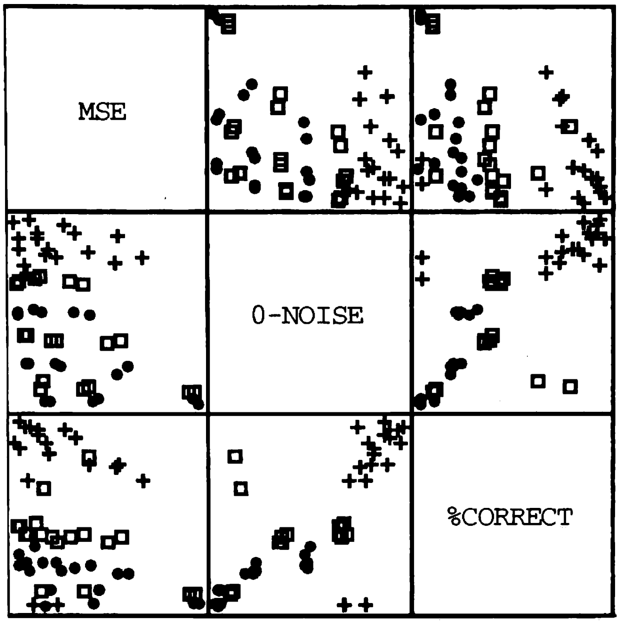

Figure 2: Scatterplot Matrix of Response Variables. Shapes +, □, • used for TYPE = CV,BACK,STEP.

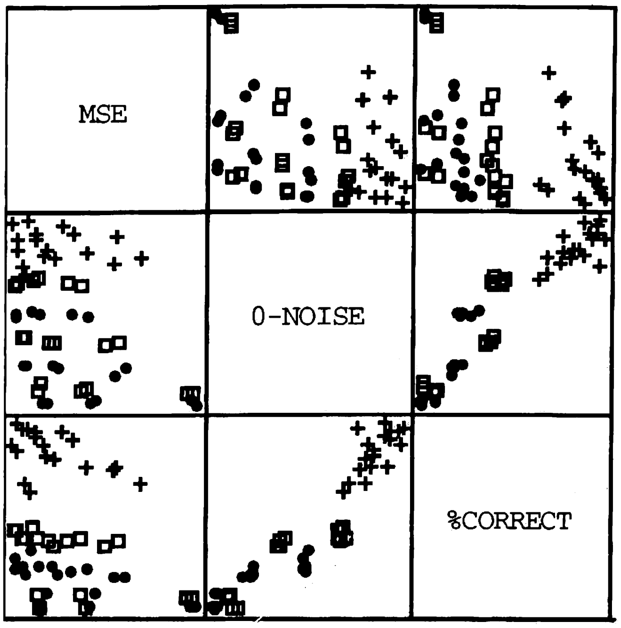

Figure 3: Corrected Response Variables. Shapes +, □, • used for TYPE = CV,BACK,STEP.

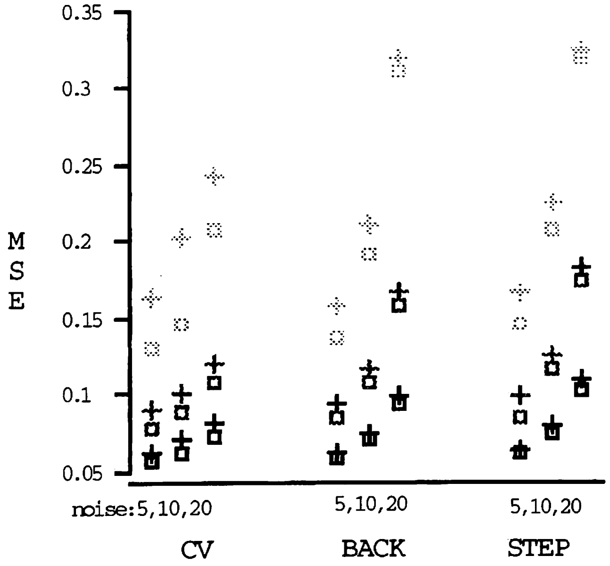

Figure 4: Variation of MSE Across Factors. Observations are separated by #NOISE within TYPE on x-axis. Shapes □, + indicate CORR = 0,0.75; light,dark, black for N = 50,100,200.

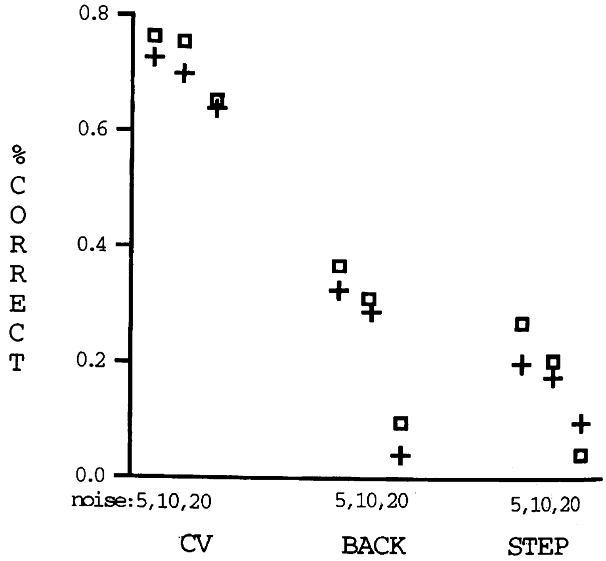

Figure 5: Variation of %CORRECT Across Factors. Observations have N = 100, and are separated by #NOISE within TYPE on x-axis. Shapes □, + indicate CORR = 0,0.75.

ACKNOWLEDGEMENTS

The author thanks David A. Grier for providing the data of section 3.1.

REFERENCES

Becker, R.A, W.S. Cleveland and A.R. Wilks 1987. Dynamic Graphics for Data Analysis. *Statistical Science* 2: 355-395

Becker, R.A, J.M. Chambers and A.R. Wilks 1988. *The New S Language*, Wadsworth and Brooks/Cole.

Buja, A., C. Hurley and J.A. McDonald 1986. A Data Viewer for Multivariate Data. In *Computer Science and Statistics: Proc of the 18th Symposium on the Interface.*

Buja, A., D.A. Asimov,C. Hurley and J.A. McDonald 1988. Elements of a Viewing Pipeline for Data Analysis In *Dynamic Graphics for Statistics*, Cleveland, W.S., McGill, M.E. (eds) Wadsworth and Brooks/Cole.

Cleveland, W.S. and M.E. McGill 1988. (eds) *Dynamic Graphics for Statistics*, Wadsworth & Brooks / Cole.

Hurley, C. 1987. *The Data Viewer: A Program for Graphical Data Analysis.* PhD Thesis and Tech. Report, Statistics Department, University of Washington, Seattle.

Hurley, C. and R.W. Oldford 1991. A Software Model for Statistical Graphics. In *Computing and Graphics in Statistics*, IMA Volumes in Mathematics and its Applications, vol. 36, Buja, A.,Tukey, P. (eds), Springer-Verlag.

Keene, S.E. 1988. *Object-Oriented Programming in Common Lisp*, Symbolics Press and Addison Wesley.

McDonald, J.A. 1986. Antelope: Data Analysis with Object-Oriented Programming and Constraints. In *Proceedings of the A.S.A, section on Statistical Computing*

McDonald, J.A. 1988. An Outline of Arizona. In *Computer Science and Statistics: Proceedings of the 20th Symposium on the Interface*

Oldford, R.W. and S. Peters 1987. DINDE: Towards more Sophisticated Software Environments for Statistics. *SIAM Journal on Scientific and Statistical Computing.*

Pedersen, J. 1991. Situations, Summaries and Model Objects. In *Computing and Graphics in Statistics*, IMA Volumes in Mathematics and its Applications, vol. 36, Buja, A.,Tukey, P. (eds), Springer-Verlag.

Stuetzle, W. 1987. Plot Windows. *Journal of the American Statistical Association* 82(398):466-475.

Thall, P.F., R. Simon and D.A. Grier, 1991. Variable Selection via Cross-Validation, Technical Report, George Washington University.

Tierney, L. 1990. *LISP-STAT An Object-Oriented Environment for Statistical Computing and Data Analysis*, Wiley.

AUTHOR BIOGRAPHY

CATHERINE HURLEY is an assistant professor in the Department of Statistics/Computer & Information Systems at George Washington University. Her research interests are exploratory data analysis and design of software environments for statistical and scientific computing.