Enhancing dynamic graphical analysis with the Lisp-Stat language and the ViSta statistical program

RUBÉN LEDESMA
Universidad Nacional de Mar del Plata, Mar del Plata, Argentina

J. GABRIEL MOLINA
Universitat de València, València, Spain

and

FORREST W. YOUNG
University of North Carolina, Chapel Hill, North Carolina

Presented is a sample of computerized methods aimed at multidimensional scaling and psychometric item analysis that offer a dynamic graphical interface to execute analyses and help visualize the results. These methods show how the Lisp-Stat programming language and the ViSta statistical program can be jointly applied to develop powerful computer applications that enhance dynamic graphical analysis methods. The feasibility of this combined strategy relies on two main features: (1) The programming architecture of ViSta enables users to add new statistical methods as plug-ins, which are integrated into the program environment and can make use of all the functions already available in ViSta (e.g., data manipulation, editing, printing); and (2) the set of powerful statistical and graphical functions integrated into the Lisp-Stat programming language provides the means for developing statistical methods with dynamic graphical visualizations, which can be implemented as ViSta plug-ins.
For example, code that supports data editing and manipulation, file management, and printing, among other functions, must be implemented in order to achieve a ready-to-use application. Such additional programming can involve a great effort on the part of the potential programmer. It is our purpose to show how much of this programming task can be sidestepped by taking advantage of the architecture of the ViSta statistical program.

ViSta is a free-distribution open statistical program, originally launched as a test bed for research and development in statistical visualization (Young & Smith, 1991). From its origins in the early 1990s to the present, ViSta has grown to offer a wide range of statistical methods (e.g., ANOVAs, simple and multivariate regressions, multidimensional scaling [MDS], correspondence analysis, cluster analysis) and, more specifically, innovative graphical methods associated with the execution of these analyses as well as with the visualization of their results.

As an open system, ViSta continues to increase its features as users interested in implementing new methods develop and integrate them into the program. An important feature of the possibility of adding new Lisp-Stat code to the existing ViSta code is that the potential programmer can use the functions already available in this statistical program, avoiding unnecessary repetition. Moreover, with the 6.4 version of ViSta, the integration into the system of a new code has been facilitated through program architecture based on plug-ins. To make his or her developments available in the ViSta environment, the interested developer need only write his or her Lisp-Stat code following a few specific rules and put this code into the ViSta plug-ins folder.

Writing a ViSta plug-in involves creating a very small amount of code beyond that which is already required for writing a model object with Lisp-Stat. Five main steps (described below) can be differentiated in this task. To illustrate these steps, we have included some sections of the code for the log-linear analysis plug-in (Valero-Mora & Young, 2002). All of the code for this plug-in can be found in the p0002.4-logllplg file in the ViSta plug-ins directory and in the associated files in the loglin subdirectory. This is a good reference for the developer who wishes to build a new plug-in; in addition, a detailed description of how ViSta plug-ins must be written is offered in the program documentation (Young, 2002). All of the code for this plug-in can be found in the p0002.4-logllplg file in the ViSta plug-ins directory.

### Step 1: Bound seven key variables for the interaction between ViSta and the plug-in environment

The following example demonstrates the case in which the name of the item is programmed to appear in ViSta's menu when the constructor function is invoked, and the slot accessor methods, which make it possible to read and write the information held in the object slots. On the other hand, the specific methods for computing and visualizing the results of the analysis should also be defined at this point. The code for a slot accessor method of the plug-in code for the log-linear analysis is shown below.

```lisp
(defun loglinear-analysis
  (&key (data *current-data*))
  (load *loglin-plugin-ctor-file*)
  (loglinear-analysis :data data))
```

### Step 2: Define the plug-in loader function, which is loaded when ViSta is run

Its primary purpose is to load the plug-in's code the first time the user needs to use the plug-in. The loader function is a small piece of code contained in the plug-in interface file (the file is placed in the plug-ins directory).

```lisp
(defun loglinear-analysis
  (&key (data *current-data*))
  (load *loglin-plugin-ctor-file*)
  (loglinear-analysis :data data))
```

### Step 3: Define the plug-in constructor function, the entry point to the plug-in's code

The function issues the :ISNEW message, which in turn constructs a new instance of the plug-in and then uses the new instance to perform the analysis called for. The constructor function must be located in a new file called the plug-in constructor file. Because the loader and constructor functions have the same name, and because the code read in by the loader function contains the constructor function, the loader function modifies itself to become the constructor function (i.e., the loader function is self-modifying). The architecture of the constructor function is as is so that code to do the analysis is not loaded until it is needed.

```lisp
(defun loglinear-analysis
  (&key (data *current-data*))
  (send vista-loglinear-proto :new *loglin-menu-item-name* data nil))
```

### Step 4: Define the prototype of the analysis model, which will be instanced each time the analysis is executed by the program user

When instanced, the resulting object will have the slots defined in the specification of the prototype, which are set to hold the relevant data for the analysis model implemented (17 slots in the example below). The prototype is required to inherit from the ViSta prototype, called `vista-analysis-plugin-object-proto`.

```lisp
(defproto vista-loglinear-proto
  (models-adjusted-list vars curr-model ncat counts
terms level-labels data-matrix data-object sorted-labels model be-hierarchic? use-logs? excluded-categories indicators-list order-of-variables
dependent-variable)
() vista-analysis-plugin-object-proto)
```

### Step 5: Define the methods associated with the prototype established in the previous step

These include some basic methods, such as the `isnew` method, which proceeds to create a new instance of the plug-in object when the constructor function is invoked, and the slot accessor methods, which make it possible to read and write the information held in the object slots. On the other hand, the specific methods for computing and visualizing the results of the analysis should also be defined at this point. The code for a slot accessor method of the plug-in code for the log-linear analysis is shown below.

```lisp
(defun loglinear-analysis
  (&key (data *current-data*))
  (load *loglin-plugin-ctor-file*)
  (loglinear-analysis :data data))
```
As an example of a development that uses the LispStat language within the ViSta environment, we present two ViSta plug-ins that include statistical visualizations with dynamic graphics. The first performs some classical psychometric item analyses, which implement some of Wainer’s (1989) suggestions about this subject; the second performs metric MDS, a procedure designed to analyze distance-like data (e.g., proximities, dissimilarities) between pairs of objects and plots it in a low-dimensional Euclidean space (Young, 1985). Both plug-ins can be installed with the respective installers, freely available from www.visualstats.org, along with other ViSta plug-ins that provide a number of statistical analyses, such as homogeneity analysis, log-linear analysis, and multivariate regression.

Classical Item Analysis

In this section, we describe the features of a ViSta plug-in that supports three classical psychometric models traditionally considered in test development: internal consistency (Cronbach’s alpha), split-half reliability, and test dimensionality based on principal components analysis. All three are supported by dynamic graphics that provide a visual and highly interactive interface. In the present article, we do not describe all three methods in detail; instead, we focus on the procedure that supports Cronbach’s alpha analysis, a classical procedure in the psychometric literature as well as in the software supply. Our purpose is to illustrate, on the one hand, what kind of visual statistical environment with dynamic graphics can be created using Lisp-Stat and ViSta, and, on the other hand, how it differs from other classical computer methods used to perform Cronbach’s alpha analysis.

Internal consistency analysis based on Cronbach’s alpha, like the other psychometric methods considered in the plug-in, takes as basic input a multivariate numerical data set in which each cell represents the score (or just the response) of a subject (rows) to an item (columns). The data set can be generated using the data sheet and the edition features available in ViSta, or it can be imported to ViSta from a text or Microsoft Excel file containing the data to be analyzed. The data set can contain kinds of variables other than item responses or scores, since, before proceeding with the analysis, ViSta enables the user to choose the specific variables to be analyzed.

To proceed with the analysis, we must first select the icon on the ViSta WorkMap that represents the data to be analyzed (in our example, the TestData icon in Figure 1). After selecting the Item/Test analysis item in the ViSta Analyze menu and then choosing the “Cronbach’s alpha” option, a new icon appears in the ViSta WorkMap, representing the analysis that is being done (the ScaRel icon in Figure 1), with another one below it (ScaRel-TestData) representing the results of applying the analysis to the data. As is standard in ViSta, two ways of visualizing the results are available to the user: (1) a text report with the results of the analysis applied to data—that is, the way in which many classical programs offer results—and (2) a dynamic graphical visualization of the results of the analysis.

The latter is presented as a spreadplot (Young, Valero Mora, Faldowski, & Bann, 2000), a ViSta graphical concept consisting of linked list and plot windows that shows different aspects of the statistical analysis results. As we will next see for our example, the windows in a spreadplot work together in a dynamic way so that changes by the user in one window automatically produce changes in all windows linked to the one with which the user is interacting.

Visualizing Cronbach’s Alpha Analysis

Each of the three psychometric methods in the item analysis plug-in has its own spreadplot, but to give an example of the dynamic graphical visualizations that can be built with ViSta and Lisp-Stat, we describe the spreadplot for Cronbach’s alpha analysis. Cronbach’s alpha is one of the forms most widely used in practice to assess the reliability of a test, this psychometric model being based on the analysis of the consistency (correlation) among the test item scores.

In order to obtain the visual output for Cronbach’s alpha analysis, the program user should click, in the WorkMap, on the small right icon within the analysis results icon (ScaRel-TestData in Figure 1). Alternatively, the user can also select the Visualize model item in the ViSta Model menu. The specific spreadplot for Cronbach’s alpha analysis will appear, showing the results for the data set analyzed (see Figure 2).

The spreadplot implemented for Cronbach’s alpha analysis includes the following six windows:

1. The list of items (Figure 2, upper left panel) lists the names of all test items (variables) in the data set. This list works as a control panel: The items selected constitute the input to recompute the analysis. The spreadplot is initialized by default with all items considered as selected, but when the program user chooses a set of items in this list by clicking on them, the spreadplot responds by adapting to the selected items.

2. The list of observations (Figure 2, lower left panel) shows the labels of all observations (subjects) in the analyzed data set. It allows identification of the subjects in other windows of the spreadplot as well as the exclusion of some of them and an automatic recomputation of the analysis. The spreadplot is initialized by default with all subjects considered as selected.

3. Reliability plot (Figure 2, upper middle panel): The horizontal line in this plot represents the Cronbach’s alpha value for all items in the list-of-items window. The curved
line shows how this reliability index would increase as the test length increases \( n \) times according to Spearman–Brown’s prophecy formula. If a set of items is selected from the list of items, as is the case in Figure 2, the curve is updated, the starting point of this curve showing the recomputed Cronbach’s alpha value for the selected items.

4. Test score distribution plot (upper right panel): By default, this plot shows the histogram of the subjects’ test scores, which are obtained for each subject as the sum of his or her scores in the test items. This plot is recomputed automatically when a subset of items is selected in the list-of-items window, making it easy to explore how the exclusion of one or more items affects the test score distribution. On the other hand, if a case is selected in the list-of-observations window (e.g., the first subject on this list in Figure 2), a horizontal section of one of the bars is outlined, showing the corresponding score of the selected case.

5. Item score distribution plot (lower middle panel): Using a side-by-side box plot, this window shows the distribution of item scores for all items selected from the list of items (for all the items, by default). Like the plot previously described, this plot will be recomputed and redrawn as soon as selections are made in the list of items or the list of observations. This plot can be especially useful for exploring and comparing the score distributions of different test items. Dots in each item box plot represent the observations and, when one (or more) observation is selected in the list-of-observations window, a line links the dots representing the scores of this observation in each item.

6. Item-test scatterplot (Figure 2, lower right panel): In this plot, the observations are represented as points in a 2-D space. The horizontal axis represents the test scores, and the vertical axis the item scores for the selected item in the list-of-items window (by default, the first on the list). The plot allows the exploration of the relationship between an item and the total test scores, giving a visual description of the discrimination power of the selected item.

Any window in a spreadplot can be maximized as well as restored to its standard size. Windows can also be copied to the clipboard or printed. Other minor features are omitted, but the program user can obtain more specific information from the on-line help system available in ViSta (Help menu).

**Metric Multidimensional Scaling**

In this section, we describe the features of another ViSta plug-in that supports metric MDS, a data analysis and visualization technique for analyzing distance-like data and displaying a low-dimensional Euclidean space that summarizes the distance information in the data (Young, 1985). MDS is based on iterative computational algorithms so that successive solutions are computed that progressively become closer to the correct final solution.
finally stopping when a criterion is reached. However, a caveat with these computational algorithms is that they are not guaranteed always to reach the best global solution and may get stuck in a so-called local minima solution. We will show how this problem is combated in this plug-in by means of graphical interactive techniques specially designed for this purpose (McFarlane & Young, 1994).

Once this MDS plug-in is installed in ViSta, it can be executed in a way similar to the psychometric analysis procedure described above. Analogously, results will be presented to the program user through ViSta double approximation to the results of the analysis: (1) a text report that includes the coordinates of the stimuli in the dimensions, information about the relative importance of each dimension, the fit of the metric MDS model, and other related results; and (2) a spreadplot offering a dynamic graphical visualization of the most relevant analysis results (see Figure 3 for a classical data set in the MDS literature, providing the distances between some US cities).

The spreadplot associated with MDS contains six linked windows: (1) a scatterplot matrix for the stimulus points in the number of dimensions obtained according to the criterion set by the data analyst, (2) a spin plot showing a 3-D view of the stimulus space, (3) a list window with the labels of the stimuli, (4) a scatterplot showing a 2-D view of the stimulus space, (5) a scree plot showing the relative fit (importance) of each dimension, and (6) a stress plot of the fit of the MDS model to the dissimilarity data, where a stress value is shown for each iteration.

We will focus on the fourth of these windows because it provides a special mouse mode called drag point, which allows points (stimuli) to be moved to new locations (space coordinates). When a point is moved using drag point, which must be activated through the mouse button in the overlay of this plot window, the model recalculates itself and presents the new results graphically. This happens because plots in the MDS spreadplot are algebraically linked, so that the corresponding elements in the matrix of coordinates automatically update after a point is moved and all plots are changed to reflect the newly revised model. A couple of advantages of this interactive graphical MDS modeling feature are that (1) it provides convenient starting points for the stimuli on the basis of theoretical reasons and (2) it can be used to test the position of certain points in order to determine alternate locations for them. The latter advantage provides a way to overcome problems of local minimum, which arise when the points arrive at a configuration that cannot be improved by moving the points in any direction in the next step; nevertheless, if more steps were to be given, the points could be substantially enhanced. An analogy for this situation is to have to go down a hill in order to arrive at a mountain to be climbed. These kinds of problems have plagued the practice of MDS, because the first solution obtained by
the MDS computational algorithm should not be accepted as the only possible solution, since there may be others that work just as well or even better. These other solutions may be easier to interpret than the standard solution and can be achieved through direct (graphical) manipulation of the positions of the stimuli according to the researcher’s knowledge of the subject being analyzed.

**Discussion**

Lisp-Stat and ViSta provide developers with a fairly easy way to implement statistical applications that include dynamic graphics. As a resource already set in the ViSta environment, new spreadplots can be written and integrated in ViSta as plug-ins, offering a way of building computerized data analysis tools that can use the resources of a consolidated statistical package such as ViSta. Although this strategy can eliminate the need to program from scratch an environment based on dynamic graphics, the potential developer should be conversant with the Lisp-Stat programming language; he or she should also refer to the ViSta documentation in order to know how the spreadplots are built and to become familiar with the rules that allow a program to be integrated in ViSta as a plug-in. As derived benefits of this option, the potential developer of graphical environments for data analysis can use the many computational and graphical functions already available in Lisp-Stat, as well as the ViSta environment, as a statistical package in which new developments can be integrated and distributed in a straightforward way. Dynamic graphical visualizations intended to perform item analysis and MDS, such as those presented in this article, are supposed to improve the effectiveness of the data analyst’s task. However, this kind of hypothesis should be tested through empirical research.

**REFERENCES**


(Manuscript received May 26, 2004; revision accepted for publication January 10, 2005.)