Interactive data transformation and parameter adjustment: a graphical interface tool

Frederic Udina*

March 22, 1995

Abstract

A tool for user design of a non-parametric transformation is developed in Lisp-Stat. Using the mouse, the user can control the shape of the function graph. Then, the graph is used to compute the required function values. The object oriented approach we use makes easy to include this tool in a Lisp-Stat program. Its use in data analysis and statistical modeling is illustrated by some examples.

This is a draft, please do not quote. Contact the author instead.

*Departament d'Economía, Universitat Pompeu Fabra (e-mail udina@upf.es)

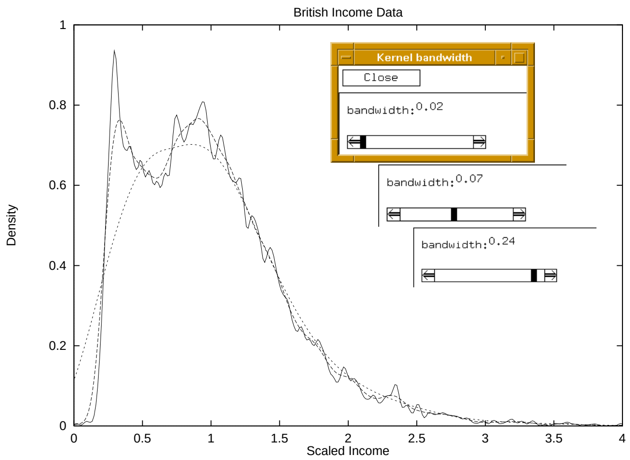

Figure 1: Kernel density estimations for a given data set. The slider used to choose interactively the bandwidth is also shown

1 Introduction

In last years, computer systems for Statistics have evolved from the simple text interface to sofisticated graphical user interfaces. This is specially true for Data Analysis systems. Data Desk and XLisp-Stat Tierney[90] are important examples of this kind of programs, but nowadays any computer system for doing statistics is moving towards this kind of graphical interface.

In such environments, the standard tool for user control of variable quantities or parameters is the 'slider'. By click-and-dragging a thumb nail, the user can choose a value within a given range. This is the way to decide, for example, the offset to be used while displaying a text longer than the window which contains it, or the number of bins a histogram must have. Figure 1 show an example of using a slider to interactively choose the bandwidth for a kernel density estimation.

Several sliders can control several parameters, but some times more than that is needed. If a continuous of values is to be defined by the user in a non-parametric way, a more rich tool is to be found.

In the rest of this paper we describe a general tool to allow the user of a statistical program to specify a function by means of drawing his graph. We provide the user of some *handles* he can drag with the mouse to modify the curve.

In the next section we show three examples of use of interactive transformations in different statistical problems. Section 3 is devoted to describe what the user can do to control and modify the transformation through the window where it is shown and the menu it provides. In section 4 the programmer's point of view is used instead. We briefly explain how a programmer can include our transformation objects in a xlisp-stat program.

2 Examples of use

2.1 Seeing the effect of data transformation

As a first and simple example we start from a lisp-stat object that displays boxplots diagrams for two data sets. We are assuming that we already have a function `boxplot` that creates such an object, and that the resulting object is able to update his data via a method, say `:new-data`.

Then we link it to a transformation object in such a way that the second data set will be the transformed data from the first one.

With a simple lisp function, we get the double window running. The function `trans-data` defined in the following program listing, performs two main jobs: creates the transformation object, and tell the boxplot how to reply to updating calls it will receive from the transformation object. The link between the transformation and the boxplot is created by the key argument in line 11 of the listing.

```
01 (defun trans-data (dt)
02   (let* ((bp (boxplot (list dt dt )))
03   (rang (butlast (get-nice-range (min dt) (max dt) 2)))
04   (transf
05    (make-transformation :initial-transformation #'identity
06         :domain rang
07         :image rang
08         :num-knots 6
09         :title "Transforming data"
10         :interpolation 'mid-point-splines
11         :notice-to-owner (list bp :get-transf-data)))
12     (defmeth bp :get-transf-data ()
13       (send self :new-data
14      (list (first (slot-value 'data))
15    (send transf :transform (first (slot-value 'data)))))))
```

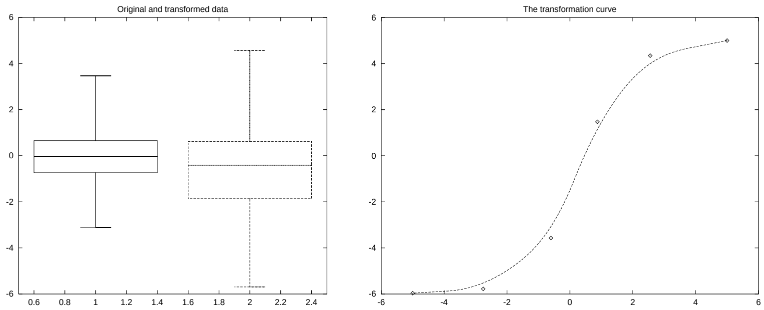

Figure 2: The second boxplot results from applying to the first the transformation defined by the curve on the right figure.

Figure 2 shows the double boxplot diagram with the curve that defines the data transformation. The curve is defined by the diamond-shaped knots placed by the user using the mouse. From the knots, the curve is built using mid-point splines.

2.2 Local bandwidth function in kernel density estimation

Our second example is fully described in Marron and Udina[95]. In the context of kernel density estimation (see Silverman[86] or Scott[92]) we consider using variable bandwidth. The function that gives the bandwidth to be used for each value in the range of estimation is freely defined with the mouse.

Figure 3 shows a location varying bandwidth kernel estimator for the income data. The upper part of the figure shows a kernel density estimate which uses the bandwidth function shown in the lower part. In this paper we suggest making choice of this bandwidth function interactive. Our user interface to the smoothing parameter function is a cubic spline. The function is controlled by the user through manipulation of the knots of the spline, shown as integers in Figure 3b.

2.3 Setting weights in a regression model

Following a suggestion from Raymond Carroll, we also apply our interactive tool to the problem of adjusting the weights in a regression model. We consider a general regression model (notation as in Carroll and Ruppert[88], p. 10)

$$E(y_i) = f(x_i, \beta), \mathrm{Var}(y_i) = \sigma^2 / w_i$$

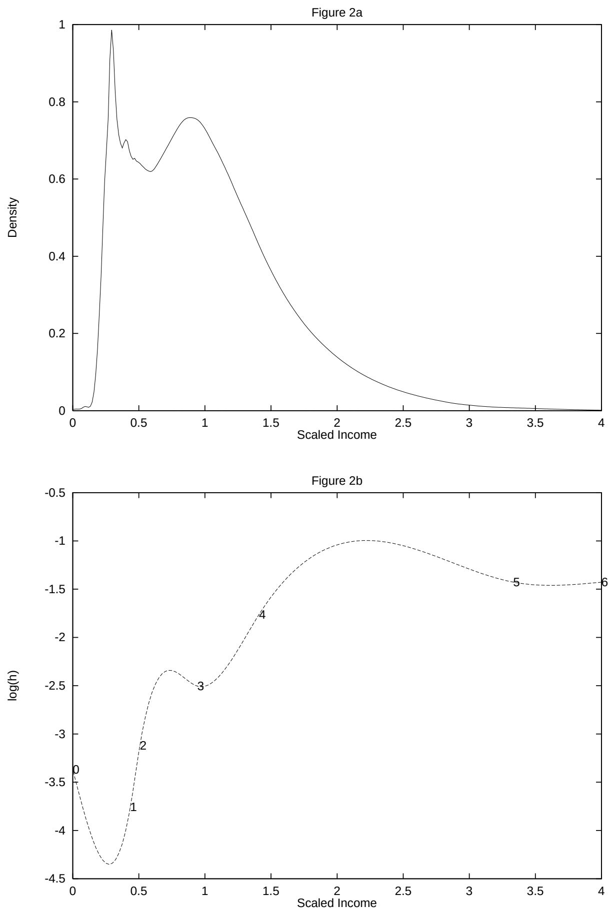

Figure 3: Biweight kernel density estimators for the income data. Location adaptive estimator in (a). Bandwidth function for (a) shown in (b).

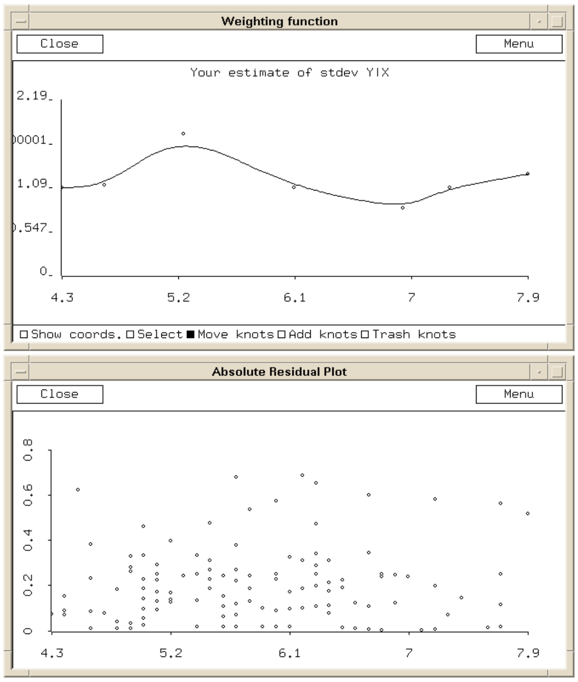

Figure 4: The weighted residuals from a linear regression model are shown in the lower window where weights are computed from the free-design curve in the upper window

where β are the parameters of the model and w_i are the weights in the generalized least squares setting. Usually the weights are modeled as $w_i = g^2(f(x_i, \beta), z_i, \theta)$ for some function g and some explanatory variables z_i (that may include some or all of the predictors x_i).

If the model holds, the weighted absolute residuals $|y_i - E(y_i)|/\sqrt{w_i}$ are identically distributed and so, when plotted display no pattern.

For an specific data set (...to be described...), we show on the screen two vertically stacked windows (see figure 4). While manipulating the weighting curve in the upper window, the user can see the impact of the choice on the lower window, where the absolute weighted residuals are shown with the same horizontal axis.

3 The user's point of view

In this section we describe the actual implementation of the transformation tool used in the examples above.

From a user point of view, a transformation is shown and modified by means of a window. In his window, a transformation object looks like the graph of the corresponding real function. In the window you will see too a few thick dots along or near the curve (see figure 5) that we will call knots. They are for controlling the shape of the transformation graph using the mouse. The actual relationship between knots and the transformation is affected by the interpolation mode in use, as controlled from the Interpolation item in the menu. The transformation window has also a menu than can appear in the menu bar on some systems. In the lower part of the transformation window, you can see a few buttons for selecting the *Mouse Mode*, that is, the effect that mouse clicks on the window will have. They will be described in brief.

3.1 Controlling a transformation with the mouse

You can interact with a transformation window using the mouse. You can pull down the Transform menu and select some of the items you will find there, or you can click (or click-and-drag) directly in the window. The way the window reply to mouse actions depend on the Mouse Mode that is currently selected.

3.1.1 Mouse Modes

You select the mouse mode either using the item named Mouse Modes in the Transform menu, or by directly clicking in the little square boxes in the lower part of the transformation window. The current mouse mode is shown by the highlighted little square box.

These are the usually available mouse modes, and the effect of using the mouse under each of them:

Show Coordinates When clicking in the main area of the transformation window, the coordinates of the mouse position will be drawn together with a pair of lines showing the coordinate position.

Select If you click-and-drag under this mouse mode, you will be able to draw a temporary rectangle. The knots that lie within this rectangle when you release the mouse button will be *selected* (and highlighted to show that they are selected). Then if you select Move knots, you will be able to move all the selected knots at once.

Move knots If you click in one knot and drag the mouse, you will move the knot to a new position. The final position can be restricted in some way, namely, the first and last knots cannot be moved horizontally (they delimit the domain of the transformation, and this is fixed when the object is created). It is possible that you (or some programmer) have decided that the knots have fixed vertical positions, see Knots options below. If you have selected two or more knots using the Select mouse mode, any click-and-drag action over any of the selected knots will move vertically (and only vertically) all the selected knots. A click in any other position will deselect all the knots. Remember that you can Undo the knots change using the menu.

Add knots When this mouse mode is selected, any mouse click in a legal position will create a new knot in that position and the transformation will be recomputed and redrawn. You cannot place new knots outside the transformation domain, nor in the same vertical of a preexisting knot. Remember that you can Undo the knots change using the menu.

Trash knots Clicking near a knot when this mouse mode is active will remove the knot and redraw the transformation graph. You cannot remove the first and last knots. Remember that you can Undo the knots change using the menu.

3.1.2 The Transform menu

When a transformation window is in use on your screen, you will be able to access to his menu, see figure 5 (the menu can appear in the main menu bar or in the window itself, this depend on the windowing system in use). We describe here the items you will find in the menu, and some related questions.

Undo knot change Only the very last change made to the knots can be undone.

Mouse mode This will pop up a dialog for changing the mouse mode. Usually your will change the mouse mode by clicking directly in the little square boxes in the lower part of the window.

Rescale plot This can be useful if some knot change has moved the knot outside the graphing area. Selecting this item will recompute the vertical scale of the graph to include all the knots in the graphing area.

Options This is the standard Options menu item for a LISP-STAT graph window. It is for choosing background colors and other graphic features.

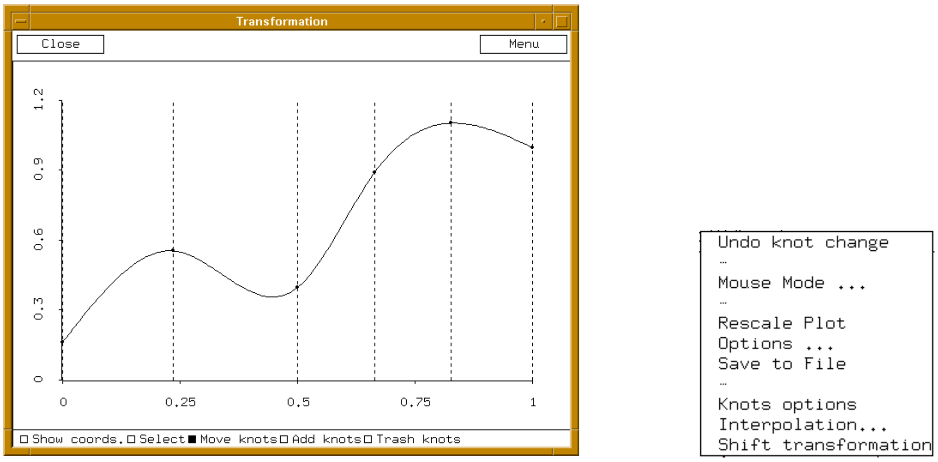

Figure 5: A transformation window with vertically restricted knots, and his menu.

Knots options A dialog window will be presented to decide about three options. Mark the first check box if you want the knots to be forced to remain in his horizontal position. The user will still be able to move the vertical position of each knot, but not the horizontal one (see figure 5). Mark the second check box if you want the knots be moved to equally spaced horizontal positions within the limits of the transformation domain. Finally, the third check box must be marked if you do want the knots be moved vertically to get a constant transformation. The constant value will be the average of the knots heights. In either case, the knots will be moved when you click OK on the dialog.

Interpolation... Here, interpolation means the method used to build the transformation curve from the knots placed by the user. There are four different methods of interpolation implemented. In the Step method, a step function is built using the knots positions, the function being constant in the intervals delimited by the mid points between the horizontal coordinates of the knots. When the Linear method is selected, the transformation is built simply joining the knots with segments. A Cubic spline is a piecewise cubic polynomial. When this option is selected, a cubic spline is computed such that it passes through the knots and in each junction the two polynomials have in common both the function value and the two first derivatives. The last method, Mid-point splines[***NOT FINISHED***], is not properly an interpolation method, the resulting curve not passing through the knots. In this case, the knots become handles to control the curve shape. The transformation is built as a polynomial spline that passes through the first and last knots and also through *the midpoints* of each consecutive knots pair. The slope of the curve in these midpoints is forced to be the same as the segment joining the knots. The advantage of this method of interpolation is that moving a knot does not change the transformation outside the two intervals adjacent to the knot.

Shift transformation This item provides a fast way of modifying the current transformation by adding some number (or multiplying by some number) to the vertical coordinates of all the knots at once. The number can be positive or negative. Be careful not to give a value that can force the transformation outside of his natural limits.

4 The programmer's point of view

The transformation tool is contained in a file named `transfor.lsp`. This file contains code for two object prototypes, `transf-proto` and `transf-window-proto`. Your program will communicate only with the transformation object, a `transf--proto` descendant, while the user will communicate only with the window transformation, a `transf-window-proto` descendant. Creating the window object and the communication between the two objects will be transparent to you (and to the user, of course).

Using transformation objects in a lisp program involves two steps: creating the transformation object and reacting to calls it will send back notifying that the user has changed the transformation. To create the transformation object you call the lisp function `make-transformation`. The more significant argument to this function tells the transformation how to communicate changes made by the user. This is done with the key parameter `notice-to-owner`. It must be followed by a list containing the arguments for a `send` call. So, the destination object must know how update things when the transformation changes.

4.1 Some examples

As a simple example, let's imagine that you have an object named `originator` that works with some data that eventually need be transformed, and you want to give the user the freedom to define the transformation in a non parametric way. Your object would have some method, say `recompute-model` for doing all

```
;suppose there is an object called originator

(load "transfor")

; now, we create the transformation
; here we don't specify all the parameters for the call
(make-transformation ...
          :notice-to-owner (list originator :transf-has-changed)
          ...))

; this method will be called every time the user change the transformation
(defmeth originator :transf-has-changed ()
   (setf (slot-value 'transformed-data)
         (send transf :transform (slot-value 'data)))
   (send self :recompute-model))
```

Figure 6: A typical use of transformation objects.

the computation of the model that depend on data, and some slots, say `data` and `transformed-data`, where this information is stored. Your program will need to contain some pieces of code similar to the listing in figure 6. A more complete call to `make-transformation` can include more parameters in the form of key/value specification as shown in figure 7.

4.2 A brief note about LISP syntax

To facilitate reading and understanding the pieces of lisp code included in the paper we comment here some basics about LISP syntax.

LISP is an interpreted language. The way the interpret works has three steps: read–evaluate–print. The interpret reads an expression, evaluates it and prints (or returns) the result of the evaluation. A LISP expression can be *simple* or *compounded*. Simple expressions are variable names, numbers, strings and similar objects. Compound expressions are surrounded by parenthesis and have a first element that is the name of the function to be called, followed by the argument(s) needed by the function call. As you can see in the following examples, the same syntax template is used to add some numbers,

```
(+ 2 4 67)
```

```
(make-transformation :domain (list someminimum somemaximum)
                     :title "It's my own transformation"
                     :num-knots 3
                     :constrain-knots t
                     :initial-transformation #'log
                     :notice-to-owner
                         (list originator :transf-has-changed))
```

Figure 7: A more refined call to create a transformation object.

for calling a mathematical function,
`(sin 0.36)`
to create a transformation object,
`(make-transformation ...arguments...)`
or to load a file
`(load "transfor")`.

When forming lisp names, any character can be used (parenthesis, quotes, double quotes, comma, semi-colon and spaces are reserved). Is a common convention to use rather long and self-explanatory names for LISP variable and function names, putting a hyphen between words in such names. Also by convention, global variables (that are seldom used) have names surrounded by asterisks, specially if they are for system configuration. Semicolon are for comments, so the rest of the line is ignored by the interpreter when finds a semicolon.

In normal LISP evaluation, the arguments are evaluated first and then the function call is made over it. Thus, an argument can be in turn a compound expression, and this can be nested to any needed level.

LISP provides a very flexible mechanism for defining functions. User-defined functions can have a fixed number of arguments or can be declared to have optional arguments. A special kind of optional arguments are *key* arguments as used in some functions described above, see figure 7 for example. Key arguments are given in the function call as a sequence of pairs of *:key* and *value*. This allows the user to specify some arguments while leaving the function to take the default values for the non-given arguments.

References

R. J. Carroll and D. Ruppert *Transformation and Weighting in Regression* Chapman and Hall, London, 1988.

J. S. Marron and F. Udina (1995) "Interactive local bandwidth choice" *Economics Working Paper Series* **109**, Universitat Pompeu Fabra.

Silverman, B. W. (1986) *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London, 1986.

Scott, D. W. (1992) *Multivariate Density Estimation: theory, practice and visualization*, John Wiley and Sons, Inc., New York, 1992.

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

Soft references

XLISP-STAT is available on `ftp://umnstat.stat.umn.edu/pub/xlisp-stat`

Our programs for transformations are available on
`ftp://halley.upf.es/stat/transfor/`

The reference Marron and Udina[95] is also available on
`ftp://halley.upf.es/stat/papers/intloch.ps.gz`

More information in `http://libiya.upf.es/`