Interactive Graphics for Kernel Density Estimation

Frederic Udina*

July 28, 1995

Abstract

We describe a computer implementation of kernel density estimation techniques in a highly interactive graphic environment. Adjusting parameters and changing options can be done in a fast and easy way, getting a rich graphical response. This requires fast computation methods. Different methods proposed recently are revised, implemented and compared. Fixed and variable bandwidth techniques are included, and also are some automatic bandwidth selection methods.

*Departament d'Economía, Universitat Pompeu Fabra (e-mail udina@upf.es) This work has been partly supported by a CESCA (Centre de supercomputació de Catalunya) grant. Useful comments from J. S. Marron and M. P. Wand have improved parts of the original manuscript.

Contents

1 Introduction

In recent years there has been a great deal of development in nonparametric methods for density estimation. Most of these methods are based in intensive computing techniques and so, when implementing them in a computer, it is important to use efficient algorithms and powerful computers. But getting fast estimates is not enough. In the techniques involved, there are many options to choose, and some parameters can take any value in a (un)known range. Despite valuable efforts done in this very active research field, there are no definite responses to these choices, and the user of a software package must be able to modify the parameters and options in a highly interactive way, i. e., having simple access to the choices and getting complete graphic feedback as important as can be computer power.

Having all this in mind, we present here an implementation of nonparametric kernel density estimation in a software package based on interactive graphics. It is an object-oriented approach built upon XLISP-STAT, a statistical oriented dialect of the Lisp language.

In this section we give a short overview of the general aspects of KDE software, paying attention to the object-oriented approach we use. In the next section we review some aspects of kernel density estimation theory relevant to the rest of the paper and to practical application of the techniques. Section 3, will provide some introduction about creating and using KDE objects so the reader can get an idea about it. We will cover the two very different ways KDE software can be used: his graphical interface can be used with the mouse by a user without any lisp knowledge, while usual LISP-STAT users can include KDE objects in his own programs. So we will describe first the graphical aspects of KDE objects, their window appearance and the way to drive them via its own menu and the dialog windows it creates for getting the user's commands. Then, a more technical approach is presented, showing how to use the objects from the keyboard or from a program file. In the appendices we give a more detailed information about the transformation object that is used in KDE software but can also be included in other statistical software. Finally we give some information about XLISP-STAT, the system that supports our development, and the steps to follow to get it all running in different computer environments.

1.1 Introduction to KDE software

KDE objects are pieces of software for kernel density estimation computation, visualization, and exploration. They are implemented in XLISP-STAT, a powerful lisp dialect developed by Luke Tierney (Tierney[90]) for statistical analysis

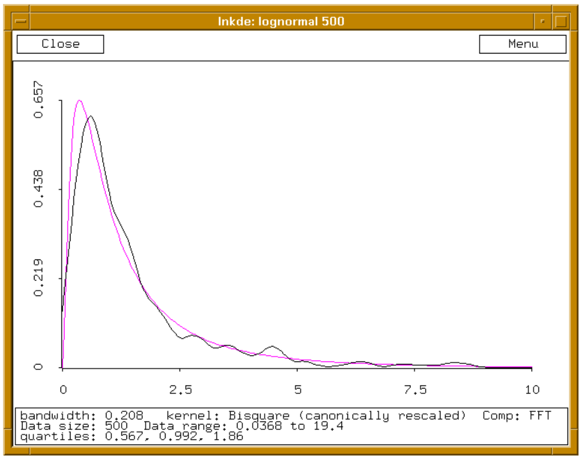

Figure 1: Typical aspect of a KDE window

and graphic display (see last section for more on XLISP-STAT).

From a user point of view, KDE software is able to take all the needed decisions to display a first density estimate starting from a data set only (see figure 1. Then it gives the user a very easy to use interface for adjusting interactively all the parameters: the range of values the estimation must cover and the number of points to be evaluated, the kernel function and the bandwith to use, wether to include or not some additional graphic information together with the density estimation (see figure 6). The user can also choose the computation method to apply and try some methods for automatic bandwidth selection. It is also posible to apply variable bandwidth techniques adjusting the local bandwidth using the mouse (see figure 4). A simple bootstrap mechanism can give some insight about the variability of the estimation, and a parametric density function can be included as reference or to generate new samples.

From a LISP-STAT user point of view, KDE software is built with LISP-STAT objects and can be used from the keyboard or from a lisp program following the usual conventions. A lisp object is a structure that contains data as well as the procedures (usually called methods) to deal with the data. The object-oriented paradigm is very useful to build complex structures and allows easy building of user interfaces in graphical environments. A KDE object, beside being able to apply our algorithms to its data, contains also a window for displaying graphical results and for interaction with the user via its own menu and dialog windows.

More concretely, a KDE object contains

statistical data The current KDE objects presented here allow only for univariate density estimation so its purely statistical data are simply a list of numbers. The object contains also other related information, such as the kernel type or the bandwidth to use, the range of x values of interest, etc...

methods In the object-oriented paradigm, a method is a function the object knows how to apply to its own data. There are mainly three kind of methods in a KDE object. *Data manipulation methods* allow the object to complete the internal data needed to achieve the computations. *Statistical methods* are for computing the estimates or some of its parameters. Finally, a KDE object has a lot of *interface methods*, including the graphic ones for visualization of the estimates and other graphs.

graphical window A KDE object is actually a descendant of a graphic object, so it inherits all the hardware and software needed for displaying its information and interact with the user.

A KDE object can also contain the description of a theoretical density function that will normally be the target for our estimates, and a random number generator to observe resampling effects. This is useful to explore the estimating capabilities of the methods in study. When the object contains the theoretical density, there are some extra methods available, for example, to compute the bandwidth for minimizing distance between theoretical and estimated density functions.

2 Kernel Density Estimation

In this section, we will summary the more remarkable facts about kernel density estimation and about how to actually compute the estimates.

Starting from a data set $X_1, \ldots X_n$, we assume that they come from a set of i.i.d. random variates. Our goal is to estimate the common density function f in a *nonparametric* setting. This means that we don't restrict ourselves to a parametric family of possible estimators.

The kernel density estimate is defined by

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i \in 1 \ldots n} K(\frac{x - X_i}{h}) \tag{1}$$

To apply this definition, a function K, called the *kernel function*, and a *bandwidth* h must be selected. For a fixed kernel function, increasing the bandwidth implies more smoothing, less variability and more bias in the estimation, and decreasing it produces the inverse effects. In figure 2 a very simple example

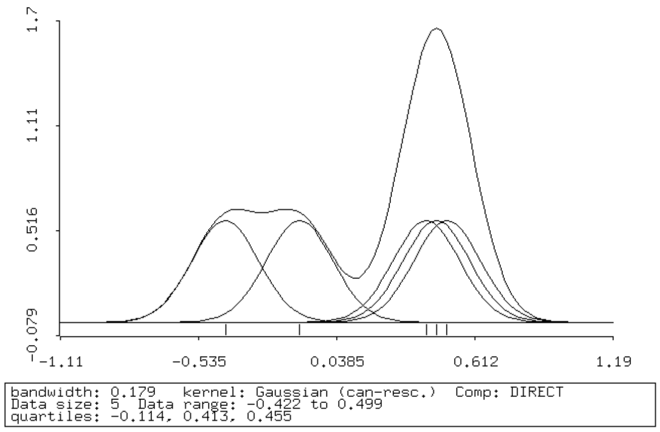

Figure 2: A decomposed kernel density estimation for 5 data points

is shown. The data set consist in only five data points, they are shown in the lower part of the graphic. For each data point, a copy of the kernel function is drawn (in a 1/5 scale for clarity) and then the sum of the five kernel functions give the density estimate. (To get such a figure on screen and to be able to interactively change the bandwidth in use and add new points, you can load the file `kdeaddpt.lsp` typing `(load "kdeaddpt")` while running KDE).

2.1 Choice of kernel

The choice of the kernel function is known not to have great effect on the practical performance of the estimator. It has obvious consequences on the properties of the estimate itself: if the kernel function is continuous, or differentiable, or has compact support, or other properties, the estimate, being a finite linear combination of translated and rescaled copies of the kernel function, will have the same properties. For example, to insure that our estimate $\hat{f}_h$ is really a density function, we will usually use kernel functions that are themselves density functions (positive and with integral over the whole real line equal to one).

The default kernel function in KDE is the biweight kernel. See table in figure 3 for a list of the kernel functions available in the current implementation.

2.1.1 Canonical rescaling

Ideally, the effect on the estimation of the kernel and the bandwidth must be clearly separate. It is clear instead that from a kernel function K, a whole family

Name	Equation $K(t)$	Code
Uniform	$1/2 \cdot 1_{[-1,1]}$	`u`
Triangular	$(1-\|t\|)_+$	`t`
Bartlett-Epanechnikov	$3/4\,(1-t^2)_+$	`a`
Biweight	$15/16\,(1-t^2)_+^2$	`b`
Triweight	$35/32\,(1-t^2)_+^3$	`w`
Gaussian	$1/\sqrt{2\pi}\,e^{-(1/2)t^2}$	`h`
an order 4 kernel	$75/16\,(1-t^2)_+ - 105/32\,(1-t^4)_+$	`r`

Figure 3: The kernel functions available in KDE.

of kernel functions can be produced by scaling the given K, say $K_\delta(\cdot) = K(\cdot/\delta)/\delta$. All the members of such a family can give exactly the same estimates for suitable chosen bandwidths, so they are completely equivalent from the point of view of density estimation. So, by rescaling the kernel one can vary the bandwidth keeping fix the value of h. On the other hand, when comparing estimates built upon different kernel functions, it is often the case that for the same value of the bandwidth one has very different degree of smoothing.

To avoid these undesirable effects, Marron and Nolan[88] proposed the use of a canonical representative of each family of rescaled kernel functions. The representant is chosen by the condition

$$\int K_\delta(x)^2 dx = \left[\int x^2 K_\delta(x) dx\right]^2 .$$

Using canonical rescaled kernels ensures you will get the same amount of smoothing on your data with the same bandwidth and different kernels.

KDE software give the choice to use or not canonical rescaling when choosing the kernel. See section 3.1.1.

2.1.2 Higher order kernels

This is a very technical issue. As explained in Silvermann[86], pag. 66, it is possible to reduce the pointwise bias of the estimation using kernel functions that are no longer density functions, they take negative values in part of his domain. The estimate we obtain cannot be a density function, but bias is reduced. You can see the graph of the fourth order kernel included in KDE selecting it in a KDE window and selecting also the `Draw kernel sample` item in the menu. We have included it mainly to be able to apply Devroye's Double Kernel method (see next subsection). Please, note that different order kernels cannot be compared through the canonical rescaling mechanism.

2.2 Choosing the bandwidth

Choosing the bandwidth is the real clue to performance of kernel density estimates. While a too big bandwidth gives less variance, it can dramatically increase the bias of the density estimate. If the bandwidth in use is too small, the opposite effect will be produced. There are many opinions favouring a subjective choice of the smoothing parameter (see, for example, SILVERMAN[86], p. 44 or SCOTT[92], p. 160 ff.). It is for sure the best strategy for data analysis or exploratory purposes. The software we present here can be a great help to visualize changes in the estimation while changing the bandwidth. KDE provides the *Bandwidth slider* mechanism for interactive choice of the bandwidth (see figure 8).

But in any case, both if an automatic bandwidth is to be selected, or an exploratory technique is to be allowed, a range of interest for the bandwidth must be computed based only on the data, and this is a decission with some inherent risk. One can find some criteria for this decission in PARK AND TURLACH[92], section 2 or PARK AND MARRON[90], section 4. Both papers base its decision in the already known targeted density function. This is not our case, so we must take a very conservative approach. Currently we are using the interval $[h_0^\star/9,\, 4h_0^\star]$, where $h_0^\star$ is the $\textsc{mise}(h)$ minimizer with reference to $N(0,1)$ (see the comments above about the rule of thumb below).

2.2.1 Automatic selection of a bandwidth

However, in applications and also for objective reference automatic bandwidth selectors are of real interest. In the last years there has been a great research effort to develop new algorithms. We have selected some among them, following applicability, simplicity and computability criteria. The first selector, the simpler, is already used in KDE when a new object is created from a fresh data set. It is not a very good automatic selector but it is simple to understand and fast to compute.

But before to discuss it and the other automatic or objective selectors, there is an important issue to introduce: what is the goal for an automatic selector? Our main goal is to estimate f by $\hat{f}_h$. So our *best* h must minimize some discrepancy criterion $\mathcal{D}(f, \hat{f}_h)$. Typically used discrepancy criteria are ISE, MISE, IAE and MIAE. They are defined by

$$\mathrm{IAE}(h) = \int_{\mathcal{R}} |f - \hat{f}_h|, \qquad \mathrm{ISE(h)} = \int_{\mathcal{R}} (f - \hat{f}_h)^2,$$

MIAE and MISE being the expected values of those quantities. Most of the good selectors target the MISE as the quantity to minimize. Least Squares Cross-Validation is designed to minimize ISE but has a curious problem in doing it: if h_{LCSV} denotes the random variable resulting of computing the bandwidth following this rule, and h_{ISE} is the bndwidth that really minimizes ISE (supposed f is known), then the two random variables have the same expected value, but are negatively correlated. This is the main argument against trying to minimize ISE and in favour of targeting MISE: this is what Park-Marron, Sheather-Jones and HSJM selectors do. The only automatic selector based in IAE minimization we implement is Devroye's Double Kernel method.

Normal based rule of thumb. This is the more simple rule for automatic bandwidth selection. It consists in computing the asymptotically optimal bandwidth (as stated by PARZEN[62]) taking for all the quantities refering to the unknown target the corresponding quantities computed for a suitably scaled normal density. The scale parameter used here is the interquartile range, as recommended by SILVERMAN[86], p. 46, or SHEATHER[92]. In summary, this is computed using the Parzen's formula

$$h_{\mathrm{ROT}} = \left\{\mu_2(K)^{-2} R(K) R(f'') n^{-1}\right\}^{1/5}$$

where the functional R is integral of squared function and μ_2 is the second order moment. Please note that if the kernel in use is canonically rescaled (see previous section) then $\mu_2(K)^2 = R(K)$ and Parzen's formula takes the remarcable simple form

$$h_{\mathrm{ROT}} = \left\{R(f'') n^{-1}\right\}^{1/5},$$

and does not depend on the kernel actually used. The so-called *rule of thumb* consist in taking as f (the unknown density function) the normal density. Then, in terms of the interquartile range λ, the optimal bandwidth is given by

$$h_{\mathrm{ROT}} = 1.0113\lambda \left\{\mu_2^{-2} R(K) n^{-1}\right\}^{1/5},$$

or, if the kernel is the gaussian one, $h_{\mathrm{ROT}} = 0.79\lambda n^{-1/5}$.

Following SILVERMAN[86], formula 3.31, one can use a more robust version of this estimator, to improve the mean integrated squared error when estimating skewed or bimodal densities. This robust version is also computed in KDE.

Least squares cross-validation This automatic method for asymptotically minimizing the ISE was introduced by RUDEMO[82] and BOWMANN[84]. For many years it has been the method of reference, but recent work has shown that other methods performs better from the bias point of view and much better in reducing the variance. See Park and Turlach[92] for a detailed description of the algorithm.

Park-Marron Plug-in method In Park and Marron[90] this estimator is defined as the largest solution (if it exists) of the equation obtained by plugging-in a better estimator of $R(f'')$ in the Parzen's formula. The authors say that they know of no cases where the solution does not exist. Our implementation of this method is done via simple iteration of the Park-Marron formula. It is much faster than the grid-search method as explained in Park and Turlach[92], but will eventually need a good (large) starting value.

Sheather-Jones Plug-In method In Sheather and Jones[91], an improvement over Park-Marron method is described that has some theoretical and practical advantages. The methodology is similar in that an equation must be solved but in this case there are less numerical problems. In KDE, we have implemented a binned version of the method, and solve the equation by a simple iteration method, by the moment. It must be stressed that Sheather-Jones method is recognized by the experts in the fields as the best automatic method available, and this is shown in the simulation work of Cao, Cuevas and Gonzalez-Manteiga[94] and also in the big bunch of simulations contained in J. S. Marron's computer.

Hall-Sheather-Jones-Marron method This automatic bandwidth selector is described in Hall, Sheather, Jones and Marron[91]. We have implemented it for two main reasons: it requires no iterations and no grid search minimizing, and has a remarkable theoretical asymptotical order of convergence of $n^{-1/2}$. It consist on the same idea of plugging estimates into the Parzen's formula, but the techniques used are more sofisticated.

Devroye's double kernel method The idea of the method is simple and very elegant. It is known that using higher-order kernels improves the bias of the kernel density estimation in some order of magnitude respect to n, the sample size. Devroye[89] argue, and show, that if we take two density estimations f_{nh} and g_{nh} built on kernels of different order, the value of h that minimize the distance between the two estimations is a good bet when targeting the minimizer of the distance from f_{nh} to the unknown density. In our implementation, following some author advice, we use the setting of the first example in Devroye[89], pag. 554. We compute the distance between estimates for 70 bandwidth values chosen) in the bandwidth interval of interest (using logarithmic spacing), and then we show the values in a separate window for allowing the user to decide if the minimum achieved is the appropriate one (in some cases double local minima can appear, and it is not possible to select between them in an automatic and fast way).

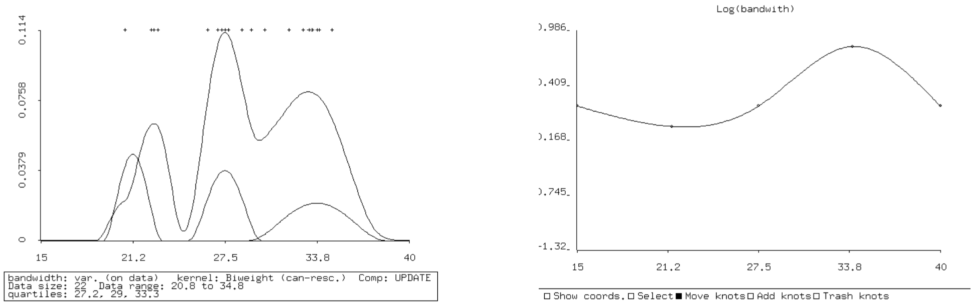

Figure 4: Variable bandwidth is applied to a data set in the left figure. In the right, the bandwidth function is shown with the handles the user can move with the mouse to change the function's shape.

2.2.2 Variable bandwidth

Some authors argue that for a better estimation of the target density, the best strategy is to apply different bandwidths to different parts of the range of interest. So, in regions where the curvature of the density is larger, a smaller bandwidth can be more suitable, and viceversa.

In order to allow the user experimenting with these ideas, we have implemented the two main ways of variable bandwidth, see Marron and Udina[95]. If in the original definition 1 we want to change h by a function, we can make h depend on x or on X_i. In the first case, we have

$$\frac{1}{h(x)} \sum_{i=1\ldots n} K(\frac{x - x_i}{h(x)})$$

and we talk about bandwidth variable on x values. This gives an estimator for f that is not itself a density function in general but has some appealing properties (see Terrel and Scott[92] for a general discussion on variable bandwidth estimators. The other way for varying the bandwidth is to give a different scale h_i to the kernel function in each data point X_i, $i = 1 \ldots n$. We have

$$\sum_{i=1\ldots n} \frac{1}{h_i} K(\frac{x - x_i}{h_i})$$

and we will talk about bandwidth *variable on data position.*

KDE allows, via the Computing method menu item, to use either of this two methods of variable bandwidth, and using it in a really interactive way. By using the Local Bandwidth Controller, see figure 4, the user can control with the mouse the amount of smoothing to apply in each part of the estimation. In the figure, three kernel curves are shown to give an idea of the different bandwidth in use in different regions of the estimate.

2.3 Computing Kernel Density Estimates

Recently, some work about fast computation of kernel density estimation has been done starting with the seminal paper of HÄRDLE and SCOTT[92]. We will refer also to FAN and MARRON[93], HALL and WAND[94] and WAND[94].

There are three main approaches to practical kernel density computation: direct computation, binning methods, and updating methods.

Direct methods are very inefficient when implemented in a computer, and are in fact impracticable in a regular computer if the sample size is medium to large (say some hundreds) and the language in use is not a very low level one such as FORTRAN or C. But direct methods apply directly the definition and thus have the advantage that there are no decisions to take in the implementation (only the kernel and the bandwidth to use), and there are no special requirements on the data. It is clear, also, that in the (very unusual) case in which a single point density estimation is needed, the direct method is the appropriate choice. For these reasons, one can find in some high level languages a routine call to compute direct kernel estimation, XLISP-STAT provides it and KDE uses it in the 'direct' computation method.

Binning methods are based in some sort of rounding the data and working from then on with equally spaced grid points and some weight-counts assigned to each one of them. More precisely, let $g_1, \ldots, g_G$ the bin centers around which we round the data. To bin the data means substitute our data $X_1, \ldots, X_n$ by the *counts*, $c_1, \ldots, c_G$. The simplest method of computing these counts is defining c_i as the number of data points included in the interval centered at g_i. This is called *simple binning* and is used in KDE software for building the histograms. But as HALL and WAND[94] show, and FAN and MARRON[94] recommend, *linear binning* is better for computing kernel estimators. Linear binning consist in splitting the weight of each data point between the two nearest grid points, according the following definition, depicted in figure 5:

$$c_i = \sum_{j=1 \ldots n} \left(1 - \frac{|g_i - X_j|}{\delta}\right)_+$$

where δ is the bin width or distance between grid points.

Once the data are binned, we will compute $\hat{f}_h(x)$ (see equation (1)) only for

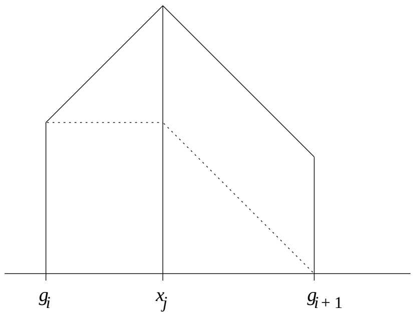

Figure 5: The unit weight at each data point X_j is split in two parts accounted to the two nearest grid points, the nearest getting more weight.

the grid points, using the approximation

$$\hat{f}_h(g_i) \simeq \frac{1}{h} \sum_{j=1\ldots G} c_j K(\frac{g_i - g_j}{h}) = \sum_{j=1\ldots G} c_j w_{i-j}(h)$$

where $w_k(h) := K(k\delta/h)/h$. From this expression it is clear that after doing the data binning step, the only remaining steps are 1) computing the kernel weights w_k and 2) perform a discrete convolution. Because most of the w_k are zero, the number of kernel evaluations to be done is really low, and so in this step 2 is where the main computational saving is achieved (Note that k ranges in $\pm G$ and $w_{-k} = w_k$). When the finite convolution is computed directly, the method is called WARP (weighted average of rounded points, see Härdle and Scott[92]). Alternatively, one can apply discrete Fourier transforms via some Fast Fourier Transform (FFT) algorithm, see Silverman[86] or Wand[94].

Updating methods are based in a simple idea: when computing the density estimation for several points, passing from one point to another means to suppress some data from the computation and to add some other data, the rest of the data need not be involved in recomputation. This is absolutely true when the kernel in use is a uniform one, and can be adapted when the kernel in use is of some polynomial type, see Fan and Marron[94]. Concretely, the kernel in use must be of the beta family, namely, referring to the figure 3, the uniform, Bartlett-Epanechnikov, Biweight or Triweight. The updating method has a big advantadge: it is the only fast method that can still be used in both way of variable bandwidth, but has also some stability problems, specially when the bandwidth becomes small relatively to the bin width. Updating ideas can be applied also to binned data and KDE's updating method work actually in this way.

Fan and Marron[93] show that, in a typical univariate setting, Warping and Updating methods are very fast and argue that use of FFT method is no worth the extra trouble. This is true, but there are some important questions to remark.

Warping and Updating are posibly faster for computing one estimation for a given bandwidth, but when we are in an interactive setting, with a user changing rapidly the bandwidth value, FFT is much more convenient because it allows computing the FFT of the array of counts $\{c_i\}_{i=1}^{G}$ for a very wide range of bandwidths. Moreover, for warping be efficient, it is important to have the proper bin width for the given bandwidth, this aspect being not so critical when convolving with FFT. So, when the user is changing the bandwidth in an exploratory task and WARP is in use, it will be often the case that data must be rebinned, and this is a hard task for big n. Finally, it must be noted that when using interpreted languages such as LISP-STAT (other popular choices, GAUSS, S, Matlab, are also interpreted) one must analyze the part of the code that will be interpreted and the part that is executed directly by inner (compiled) routines. This can change from one package to another, but in general, the FFT routine is internal and very fast, while direct vector convolution can be slow in some languages, as is in LISP-STAT. In GAUSS and Matlab, discrete convolution is also available as internal compiled routine.

In summary, we have seen that implementing a fast kernel estimation method implies three steps 1) rounding the data into grid points, 2) computing some weights and 3) doing a finite convolution. The first step is done usually by a linear method, and the third can be done directly (WARP) or by Fourier Transform (FFT). Alternatively, 2 and 3 can be substituted by an updating mechanism. In the next section we will come back to this issue from a non-expert user point of view.

2.3.1 Fast Computing automatic bandwidth selectors

Another involved issue is the computation of automatic bandwidth selectors. The simplest, the normal based rule-of-thumb is very fast to compute but his performance is very poor except for estimating densities near to the normal, in which case it can rather be better to use a parametric method. In recent simulation studies (see for example Cao, Cuevas and Gonzalez-Manteiga[94]) selectors as the PM, SJ or HSJM mentioned in section 2.2.1 are shown to be the best. Actual calculation of these selectors involve the computation of a double sum to estimate the integral of the second derivative of the unknown density. This pose a very high computational cost unless some binned method is used. Some work in progress by Matt Wand and the Santiago people discuss accuracy questions that arise in this problem. We have implemented and tested some of these methods in a binned version, and the provisional conclusion is that with a high grid count (greater than 300, say) they don't have big accuracy problems.

3 Using KDE objects interactively

This section is intended to give the reader an idea about how to use the software described in the previous section. In the appendix we describe the hardware and software requirements for running KDE software and the ways to get it all. In this section we describe how to use KDE in two main ways: as a non-lisp user or as a introduced lisp user that writes his own programs. If you are not a lisp user you will need to type only a few (or none at all) lines of lisp to get KDE running, and from then on the mouse will be the tool for comunicating with the graphic objects. We will try to do also in this section some technical advice for helping users non expert in density estimation to take the main decissions while using the software. The second part of the section, is for XLISP-STAT users and provides them with hints for using KDE object from the lisp interpreter or from his/her own lisp programs.

KDE software is built over XLISP-STAT, see the appendix for a brief description. Let's assume that you have all the requirements detailed there and so you be able to load the file that get all together running, `runkde.lsp`. So, once you have the xlisp prompt, usually a `'>'`, you can type

```
(load "runkde")
```

and you will get the main file loaded. Other files will then be loaded as needed without further user operation.

If you have some data already in a lisp list form, you can then create a KDE object to analize your data by typing

```
(make-kde :data my-data :title "Window for my data")
```

If your data reside in a file, you can use a call similar to

```
(make-kde-from-file "myfile")
```

provided that your data are in the file in the usual way, separated only by spaces or newline characters.

Once you have your data installed in a KDE object, you will get a window on the screen, and have access to the KDE menu. See the next section for a description of the items it provides. By choosing among them with the mouse you will be able to fix the kind of kernel to use, or to interactively choose the bandwidth, for example.

But if a user wants to work with the object from the keyboard or from a program file, (s)he will *send messages* to it. In the next very simple example, we create a KDE object named testkde with 500 normal random numbers, we decide to use a bisquare kernel and ask the object for the optimal bandwidth according the Rule of thumb method (see below for details). Lines beginning with '>' are typed by the user, each one is followed by a response line from the lisp interpreter.

```
> (def testkde (make-kde :data (normal-rand 500)))
TESTKDE
> (send testkde :kernel-type 'b)
B
> (send testkde :rule-of-thumb)
(0.80787 NIL)
>
```

The file `runkde.lsp` also implements other functions to create specific KDE objects useful for testing purposes. For example, after `(load "runkde")`, calling `(mkn 500)` will create and save in a variable named `nkde` a KDE object with 500 normal random data points. In MARRON and WAND[92] there is a very interesting set of normal mixture densities. You can play araound with them by calling the lisp function `make-mw-kde` with a number between 1 and 15 as the density number and an optional second argument for the sample size. For example, figure 6 has been obtained by typing `(make-mw-kde 8 800)`. A user knowing something about the LISP-STAT language can easily build his own functions to explore particular distributions and/or sample sizes. The file `runkde.lsp` contains other functions as well for using it or for serve as further examples.

3.1 The graphical interface to a KDE object

Each KDE object is visible in a window. A KDE window has three regions (see figure 1 or 6): The button region, the main display region and the info region. The button region, whose appearance depends on the windowing system in use, contains a button for closing the window and possibly another for opening the KDE menu (see below for a description of the items available on the menu). The KDE menu can appear in some menu bar when the KDE window is active. In the info region, some methods can display textual information about the progress or result of their calculations. In the main region of the window, the estimated density is displayed framed by labeled axis. If the object has been assigned a

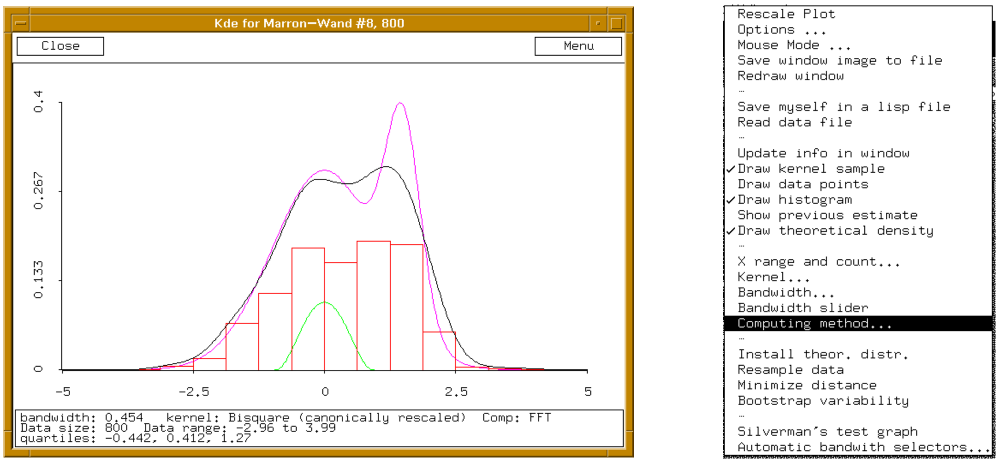

Figure 6: A KDE window with added graphical information, and the KDE menu.

theoretical distribution, its density function can be displayed using a different pattern or color. Using the mouse, the user can do different actions (see Mouse modes below) in this main region of the window. One can also decide to display, together with the density function(s), a histogram of the data and/or other useful complementary graphical information.

3.1.1 KDE Menu

When a KDE window is active, the user has access to the KDE menu. It offers a series of items in five groups. The first group offers items for graphic control of the window, the second group includes basic file access, the third one allows the user to control what information is to be displayed in the main area of the window. Through the items in the fourth group the user can control the basic parameters of the kernel estimate and of the computed approximation to it, and finally, the last items provide some statistical techniques to explore the kernel density estimate.

The items of this first group control basic aspects of the window and its interaction with the user. They are provided essentially by XLISP-STAT as standard capabilities of graphical windows.

Rescale plot Redraw the whole window, rescaling in order to include all the lines internally computed. It can be necessary to use this item after changing some parameter in the current estimation.

Options... This is the standard options dialog from XLISP-STAT. It allows the user to choose some display options like the background color. (S)he can also decide whether the window has scroll-bars or must draw all its contents stretched to fit in the actual window.

Mouse mode... The mouse mode determines what the user can do with the mouse by clicking (and dragging) in the main region of the window. Currently, selecting this item will pop-up a dialog window for selecting one of three mouse modes. (see figure 7 and the description of mouse modes above).

Show coordinates... The default mouse mode. When the user clicks within the graph, coordinates of the click point are shown and lines locating the point are drawn. One can move the mouse with the button pressed to read the numbers in a clean place.

Zoom in allows the user to click-and-drag the mouse to define a rectangle in the graph. The graph will be rescaled to focus on this rectangle. To restore the original view rectangle, user can select `Rescale plot` from the menu or can do a shift-click in the graph (i.e. press the mouse button while holding down the shift key).

Probability from..to.. allows the user to push down the mouse button in a "from" value x_0 and release it in a "to" value x_1 and have an approximate value of the estimated probability $P(x_0 < x < x_1)$ according the current kernel density estimation. It can be useful when using variable bandwidth depending 'on x-position', because the estimates obtained in this that case posibly does not integrate to one.

The file `kde-add-point.lsp` implements an additional mouse mode for adding points and seeing how the estimate is changing. Typing

```
(load "kdeaddpt")
```

you will obtain a kde window with a few data points, for each click you do in this window you will add a new data point in the mouse position. All the other features of KDE objects will still be working.

Save to file This item prompts for a filename and then saves an image of the main region of the window to this filename in postscript format. This item (XLISP-STAT native) is not available in some versions of XLISP-STAT. In Macintosh or MS-Windows versions of XLISP-STAT, you can use the standard facilities from the Edit menu to Copy/Paste the graphs in KDE windows to some graphics program, and then save or print your windows from it.

Redraw window Does it.

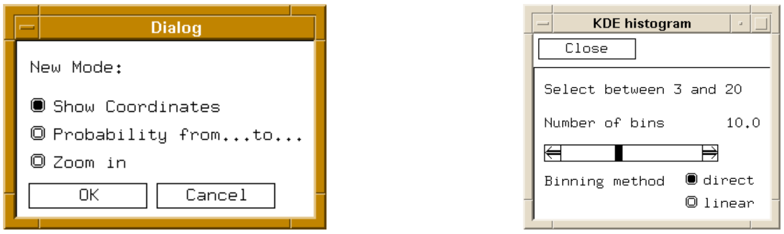

Figure 7: Some dialogs of a KDE window

The items in this second group provide access to files for loading data into the KDE object or for saving the current configuration of the object in a lisp file.

Save myself in a lisp file This item prompts for a filename to use, and save in the file all the lisp instructions that will bring to life the object in a future lisp session. For example, if you save the object in a file called `aFile.kde`, typing `(load "aFile.kde")` at the lisp prompt, you will get the object again. This is a very convinient way to store and modify a KDE object. Once you have saved the object, you can open the file eith your favourite editor and, with a little of LISP knowledge (see section B.3 in the appendix) you can modify the lisp call that will recreate the object.

Graph to gnuplot file Will display a dialog to choice some options and then, a gnuplot version of the graph window will be created.

Read data from a file Instruct the KDE object to read his data from a file in some standard formats. For the moment, one must provide a file containing raw univariant data, separated by spaces, tabs or newline characters.

Items in this third group allow the user to decide what information will be displayed in the window. Most of them are mark/unmark items. This means that selecting them will make a little mark appear at the left of the item in the menu, showing that it is already active. Selecting the item again will unmark it, and its function will be deactivated.

Update info in window Some basic info about the current data set and the current kernel type and width will appear in the bottom of the window. Some commands or posible programs can change the information being displayed in this area without updating it. Using this item the user can force redisplaying the correct current basic information.

Draw kernel sample When this item is marked, a little kernel function is drawn over the graph, to give a graphical idea of the bandwidth and kernel

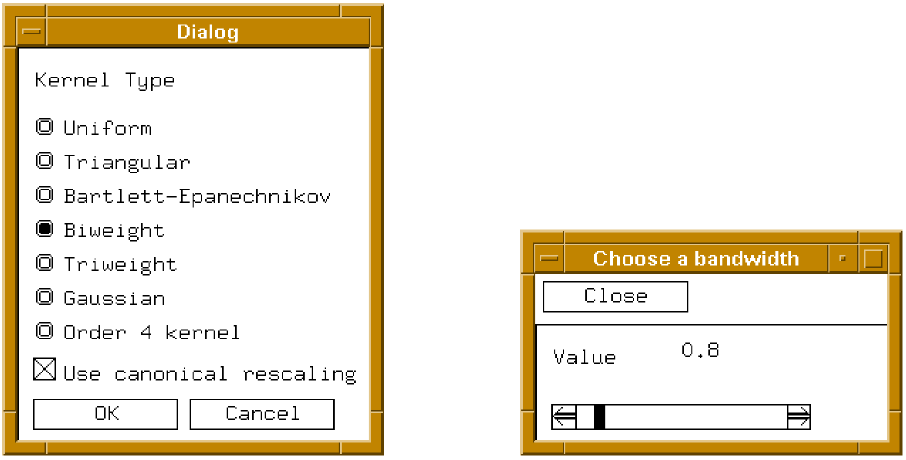

Figure 8: Dialogs for changing the kernel and tuning the bandwidth

currently in use. If variable bandwidth technology is currently in use, three copies of the kernel function will be drawn (see figure 4) showing a sample of the different bandwidths applied.

Draw data points If you mark this menu item, in the upper part of the KDE window some cross marks will be drawn showing the position where some data exist. No difference is shown about how many data point there are in each position, this can be investigate with the histogram or the denstity estimation itself. Please note that this item should not be marked when working with a big data set: the information you get is not really useful and drawing the window can be frustatingly slow.

Draw histogram While this item is marked, a scroll-bar dialog is available to determine the number of bins the superimposed histogram of your data will have. Close the slider dialog or select this item again to suppress the histogram. Default number of bins is computed by Sturge's rule, see for example SCOTT[92].

Show previous estimate If this item is marked, and the user makes some change to the estimation (the kernel or the bandwidth is changed, for example) a second curve will be drawn, showing the estimation previous to the last change. It is shown in a different colour, usually blue.

Draw theoretical density Obviously, this item will be accessible only when a theoretical density function has been given to the object. In this case, and if the item is marked, a shadow of the theoretical density function will appear together with the estimated density.

Items in next group are to determine the basic choices about the kernel density estimation: the kernel type and the bandwidth to use. But remember: what you get when using a computer is not the kernel density estimation, it is only a numerical approximation, so you must also be able to determine how this computation is done. Items in this group provide you the means to decide it.

X range and count Allows the user to give new minimum and maximum values for the x-axis. Once given, this range will be used in all the displaying and computing estimates. The user can also type in a third number, the number of values to use for display and calculations. Thus, one can enter in the x-range dialog either "`-1  1`" or "`-1 1 200`". The resulting numbers will be used as grid values. When using some fast computation method involving binning (see section 2.3.1), these numbers become the bin centers.

Choose kernel... A dialog window will appear showing the kernel functions available (see figure 8, see also figure 3 for formulas describing the kernel functions listed in this dialog.). This dialog will let you also decide whether to use canonical rescaled kernels or not. See section above for a discussion of this question. In brief, remember that if you do pretend to compare the estimation given by different kernel functions you must check this option. If not, checking it can no harm you anyway.

Bandwidth... The user can type the bandwidth to use in the density estimate. The user can also type the bandwidth range (two numbers with a space between them) to use in several methods like the one invoked by the following item or when automatically selecting the bandwidth with a optimizing criterion. For using varaible bandwidth technology, go to the item Computing method.

Bandwidth slider/Local bandwidth controller The bandwidth slider allows the user to interactively choose the bandwidth by moving with the mouse the slider in a scroll-bar (see figure 8). Following the suggestions of, for example, MARRON[93], the slider has been tuned with a logarithmic scale. See below for some considerations about the bandwidth range in use. When variable bandwidth is in use, instead a slider dialog you will get a *transformation window* to decide what bandwith function is to be applied. By moving the knots with the mouse, you can decide what bandwidth value must apply to each part of the X axis. Note that the transformation window has logarithmic scale in the vertical axis. See the figura 4 for a typical bandwidth controller. In the appendix of this paper a full description of the transformation interface can be found, here we will give only the main hints to its use. For some insight about variable bandwidh, see section 2.2.2 The transformation window has a menu, a main graphing area and a Mouse Mode area in the lower part of the window. In the menu you can find a Shift Transformation item for a fast way of modifying the current transformation by adding to all the values a fixed constant, let's say. You can find also a Knots Options item that allow you to move horizontally the knots to an equally spaced position, among other things. By clicking with the mouse in the little square at lower part of the window, you can change the mouse mode. The mouse mode determines the effect of clicking and dragging with the mouse in the main area of the window. For exemple, you can click first in the little squared box placed at left of the 'Select' label to choose the 'Select' mouse mode. Then you can click-and-drag the mouse to select some knots by closing them in the selection rectangle. Once some knots have been selected, you can choose the 'Move knots' mouse mode and move all the selected knots at once.

Computing method Allows the user to interactively choose the computing method to apply to the kde object. Currently, it will give four possibilities: Direct, Binned with WARP, Binned with FFT and Updating. See section 2.3.1 for a description of the methods. From a non-expert user point of view, we recomend using Binning/FFT method usually, that's the default in KDE. Updating methods are the right choice for variable bandwith. Only if the data set is of small size (less than 200, say) the direct method can be faster.
The lower part of this dialog window allows the user to choose between fixed bandwidth estimation or one of the two variable bandwidth techniques available (see section 2.2.2).

The last group of items in the menu consist in statistical methods to explore the kernel estimation. The first three of them are useful to explore and visualize the estimation capabilities of kernel methods to fit a particular distribution with a given sample or sample size.

Install theoretical distribution A dialog will be displayed offering some theoretical distributions, see figure 9. If one is chosen, its density function will be taken as reference in the kde object and the corresponding random generator will be installed also. They will be used through the menu items that follow. Choosing 'Normal Distribution Adjusted to data' the distribution will be a Normal one with mean and deviance estimated from the current data set. If you choose 'Normal Mixture', a dialog will be presented for desinging you own mixture using the mouse over a set of sliders.

Resample data Using the random generator attached to the object (if there is one), a new sample will be generated with the same size as the current data. If there is no random generator for the object, this item will be not available.

Minimize distance A golden-search minimization loop will start to determine the bandwidth that minimizes the distance between estimated density and theoretical density. Distance to use can be fixed between those derived from the $\mathcal{L}_0$, $\mathcal{L}_1$ or $\mathcal{L}_2$ norms. This process can be rather slow on small computers specially if the number of grid points is big. Be warned: in some rather strange cases, it is possible to have more than one local minima in the mentioned distance, so the golden search procedure can fail the target.

Bootstrap variability One can give the number of bootstrap samples to use. Once the samples are generated, and the density estimation for each one is drawn (using the current bandwidth and kernel function), a series of lines will appear. They outline the quartile values (0, 0.25, 0.5, 0.75 and 1 quantiles) for the values obtained for each probability density in each of the current x-values. (This item is not available right now, it must be improved).

Automatic bandwith selectors... Choosing this item will bring up a dialog box (see figure 3.1.2) providing some methods to automatically select the bandwidth. To *automatically select* means to base the calculations only on the data, although differents methods apply different assumptions on the target density function and different criteria to measure the discrepancy between the target and the estimate. There is absolutely no way to decide what is the *best* method and thus the best bandwidth. Some of them are very fast to compute, some are proven to perform good in simulations, the user must try a few of them and decide by examining the density estimates obtained.

In KDE we provide some automatic selectors. We have selected them looking at good performance, easy of computation and/or spread of use in the literature. Here we give only some short hints directed to a non-expert user, some more technichal description and references can be found in section 2.2.1.

Normal based rule of thumb This is a really simple and very fast to compute rule to select the bandwidth. It assumes that the target distribution is a normal one, so it can be very unreliable. This is the default value KDE takes when first reading a data set.

Least squares cross-validation The idea of this selector is a *leave-one-out* one. It selects the bandwidth that better predicts each data point based on the remaining data when removed. Though it has some assymptotic optimal properties, is known to have some tendency to oversmoothing and to have a high variability, so it has no great reputation.

Park-Marron Plug-In method

Sheather-Jones Plug-In method Based on the same principles as the rule of thumb, these highly improved selectors are much more costly to compute. They use similar technology and simulation evidence favoures the second over the first but with no great difference. Our implementation of these methods is done via simple iteration of a function ϕ to solve an equation of the form $x = \phi(x)$, so the initial value matter. You can try the methods with small or big initial values, usually you will obtain equivalent results in the sense that density estimates will be about the same in both cases. When in doubt, fix the starting value as given by the rule of thumb above.

Devroye's Double Kernel method This method try to minimize the $\mathcal{L}_1$ distance between the estimate and the unknown density by minimizing the distance between two estimates. After some time of computing, a graph of the distance to minimize will be shown in its own window. The user can then evaluate if the minimum obtained in the approximate computation is to be trust or there is another local minimum preferable. In the graph window, a zoom-in mouse-mode is provided.

Hall-Sheather-Jones-Marron method This selector is faster to compute than most of the others and has a very good asymptotical performance.

Silverman's test graph A window with the graph of the second derivative of the estimated density will appear to easy application of Silverman's test graph method (see SILVERMAN[86]). The user can use the bandwidth slider dialog to adjust the bandwidth and see how the second derivative changes. In Silverman's own words (see p. 56, text enclosed in brackets is ours)

> ...for good estimation of the density itself, the magnitude of the noise in $\hat{f}''$ [*$\hat{f}$ is the estimated density, while f is the unknown density function to estimate*] will be about half of the maximum value of the trend of this curve. For reasonably large sample sizes, the noise will appear as rapid fluctuations of the curve $\hat{f}''$.
> The proposed method for choosing the smoothing parameter is as follows. Draw 'test graphs' of the second derivative of $\hat{f}$ for various values of h [*h is the bandwidth, this step can be done interactively*

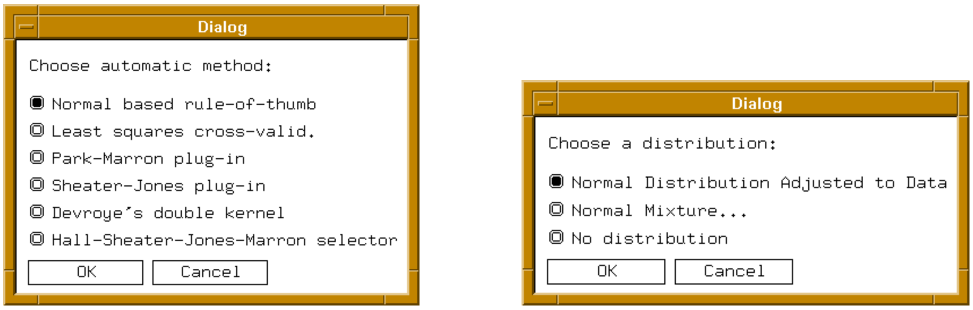

Figure 9: The dialog for choosing bandwidth selectors and the dialog for installing a reference distribution.

> *with the bandwidth slider*]. In the light of the discussion above, the ideal test graph should have rapid fluctuations which are quite marked but do not obscure the systematic variation completely. Choose the window width that yields a test graph conforming to this principle, and use this window width for estimating the density itself.

Although this method has no great reputation in the current nonparametric comunity, we agree with Silverman in recommending some trial with it in a interactive context like KDE.

3.1.2 Some keyboard commands

Some interface commands can be typed in when a KDE window is active.

key	command
`+`	multiply bandwidth by 1.5
`-`	divide bandwidth by 1.5
`a`	rescale plot
`d`	toggle show data points
`b`	toggle boxplot
`k`	toggle show kernel sample
`h`	toggle show histogram
`p`	toggle histogram/frequency polygon
`r`	shift histogram bin edges 10% to right
`l`	shift histogram bin edges 10% to left

3.2 Using KDE objects from xlisp-stat

Besides using KDE with the mouse, XLISP-STAT users with some experience can also include KDE objects in his programs or modify some of its methods to suit special purposes, or simply to improve it. This section gives some clues for facilitate it.

KDE objects are actually implemented in two main branch of objects, the `kde-proto` descendants and `wkde-proto` descendants. `wkde` objects contain the window and other interface widgets for allowing `kde` objects display and communication with the user. `kde` objects actually hold the statistical and computational machinery.

As all other LISP-STAT objects, kde objects have *slots* and *methods*. Slots are like variables owned by the object, and accesible only through it. Methods are functions or procedures owned by the object, and posibly different from other methods with the same name but belonging to other objects.

The main slots of a kde object are accessible through *Accessor methods*. Conventionally, an accessor method is a method that is used to get or set the value of a slot and usually have the same name as the slot. You can find below a commented list of the slots of a `kde` object that have an accessor method. You will find also a couple of examples of using this kind of methods. If you want to know all the slots of a `kde` object, you must look at the file `kde.lsp` for the definition of `kde-proto`.

3.2.1 Methods implemented in KDE objects

Reading the description of the KDE menu items available gives an idea of the main methods currently programmed for a KDE object. Nevertheless, we will explain here with more detail some of the more interesting methods. But before, we must explain some technical terminology of the object programming paradigm as used in XLISP-STAT.

As said before, methods are like "local functions". Each object *knows* how to apply its methods, i.e., it knows which piece of code must be executed. Using the object-oriented programming paradigm means that the method will use the most of the information it needs from the slots of the object who is running the method (the local values) instead requiring all the information via parameter passing. So, slots and methods provide a very convenient way of *encapsulating* all the information in one unique piece of software.

To invoke a method, we *send* to the object a *message*, like in:

```
> (send ofd :describe-data)
((107 3.45991 1.0403) (1.67 2.3 3.8 4.25 4.93))
```

and we get a response, in this case some statistics describing the data set. In this example, an object called `ofd` is sent a message `:describe-data` that produces the execution of the method bound to this message.

In the rest of this section we will describe some of the methods a KDE object knows about, mainly those with more statistical interest.

Accessor methods

Many methods are of the *accessor type*: they are designed for accessing the slots (these sort of object's local or private variables) to get or to set his values. So, if we send to a `kde` object the message `:data`, we get the statistical data stored within the object. If we send the same message together with a list of values, we are setting this list as the new data for the object. In the table of figure 10 there are the main methods defined for KDE objects, with a brief description of the meaning of the information they manage. Complementary information can be found in section 2. In the third column of the table the default value for each slot is shown. Those with a star as default value are slot which default value is computed from the data following criteria explained in section 2.

Examples of use of accessor methods

Here are two examples of legal lisp lines using some of the accessor methods described above for a KDE object named `akde`:

```
(send akde :bandwidth)
(send akde :bandwidth 0.8)
```

The first will return the current bandwidth in use for `akde`, the second line will install 0.8 as the bandwidth for the object `akde`.

If you have an object `mykde` created, say, by the function `make-kde` (see section 3), you can customize it by sending the following series of messages:

```
(send mykde :title "My first kde object")
(send mykde :kernel-type 'h)
(send mykde :use-canonical nil)
(send mykde :calc-method 'warp)
(send mykde :bandwidth 1.5)
(send mykde :distr-dens #'normal-dens)
...
```

Message	Information accessed	default
`:data`	*the statistical data to work with*	
`:x-values`	*the list of values in the x axis to use in computations*	$(\star)$
`:estimates-y`	*The list of y values of the current estimation, to be used jointly with the previous*	$(\star)$
`:bw-ends`	*the ends of the interval of interest for the bandwidth*	$(\star)$
`:kernel-type`	*The code for the kernel function, see figure 3*	`'b`
`:bandwidth`	*the bandwidth or window width*	
`:distr-dens`	*the theoretical density function attached to the object*	`nil`
`:distr-rand`	*the random generator attached to the object*	`nil`
`:title`	*A string to be shown as window name*	"KDE"
`:variable-bandwidth`	*A flag* (`t/nil`) *meaning use variable/fixed bandwidth*	`nil`
`:use-canonical`	*A flag* (`t/nil`) *meaning use or not canonical rescaled kernels*	`t`
`:calc-method`	*The computing method, must be one of* `direct update warp fft`	`fft`
`:histogram-num-bins`	*The number of bins the histogram will have next time it appear*	$(\star)$

Figure 10: The main accessor methods in a KDE object

References

Bowman, A. W. (1984) "An alternative method of cross-validation for the smoothing of density estimates", *Biometrika*, **71**, 353–360.

Devroye, L. (1989) "The double kernel method in density estimation" *Ann. Inst. H. Poincaré Probab. Statist.* **25**, 533–580.

Fan, J., Marron, J. S. (1994) "Fast implementations of nonparametric curve estimators" *Journal of Computational and Graphical Statistics*, 3:35-56.

Hall, P., Sheather, S. J., Jones, M. C., Marron, J. S. (1991) "An optimal data-based bandwidth selection in kernel density estimation" *Biometrika* **78**, 263-269.

Hall, P., Wand M. P. (1993) "On the accuracy of binned kernel density estimators" AGSM Working Paper 93-003, University of New South Wales.

Marron, J. S. (1993) "Discussion of 'Practical Performance of Several Data driven bandwidth selectors' by Park and Turlach", *Comp. Stat.*, **8**, 17–19.

Marron, J. S., Nolan, D. (1988) "Canonical kernels for density estimation", *Statistics & Probability letters* **7**, (1989) 195–199.

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

Park, B. U., Marron, J. S. (1990) "Comparison of data-driven bandwith selectors", *JASA*, **85**, 66-72.

Park, B. U., Turlach, B. A. (1992) "Practical Performance of Several Data driven bandwidth selectors", *Computational Statistics*, **7**, 251–270.

Parzen, E. (1962) "On estimation of a probability density function and mode", *Ann. Math. Statist.*, **33**, 1065–1076.

Rudemo, M. (1982) "Empirical choice of histograms and kernel density estimators", *Scand. J. Statistics*, **9**, 65–78.

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

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

Scott, D. W. and Factor, L. E. (1981) "Monte Carlo study of three data based nonparametric density estimators", *JASA.*, **76**, 9-15.

Scott, D. W., Tapia, R. A., Thompson, J. R. (1977) "Kernel Density Estimation Revisited", *Nonlinear Anal. Theory, Methods Appl.*, **1**, 339-372.

Sheather, S. J (1992) "The performance of six popular bandwidth selection methods on some real data sets", *Comp. Stat.*, **7**, 225–250.

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

Wand, M. P. (1993) "Fast computation of Multivariate Kernel Estimators" AGSM Working Paper 93-007, University of New South Wales.

A Transformation objects in XLISP-STAT

Transformation of data or parameters is a technique used very often in practical statistics. In kernel density estimation it appears at least in two contexts in the last years research: WAND, MARRON and RUPPERT[91] apply some transformation to data before doing density estimation and they apply the inverse transformation to the density obtained for getting the density estimation of the original data. In a very different context, and following a suggestion of Steve Marron, we have investigated the effect of applying a user defined transformation to the bandwidth to be used in a kernel density estimation.

In this appendix we will describe an implementation of an interactively user-defined transformation. You can try and use it by selecting one of the Variable Bandwidth options in the Computing method item in the KDE menu, and then asking for a Local bandwidth controler in the same menu. In the rest of this appendix, we will explain the use of transformations in our LISP-STAT implementation, both from the point of view of a non-expert user and of a lisp programmer.

In our context, a transformation is a map between two intervals of the real line, called domain and image respectively. Though parametric families of transformations are very interesting, we focus here our attention in non-parametric transformations: the user can decide the aspect of the graph of the transformation using the mouse.

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 the curve (see figure 4 or 11) that we will call knots. They are for controling 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. 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.

A.1 Controling a transformation with the mouse

When a transformation window is on the front of your screen, you can interact with it 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.

A.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 toghether with a pair of lines showing the coordinate position. The number and the lines will be erased when the mouse button will be released.

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). If you then choose mouse mode Move knots, you will be able to move all the selected knots in a single move.

Move knots If you click in one knot and grag the mouse, you will move the knot to a new position. The final position can be affected by some restriction, 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). Also, it is posible that you (or some programmer) have decided that your knots have fixed vertical positions, see Knots options in the Transform menu for more on this. 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 selected 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.

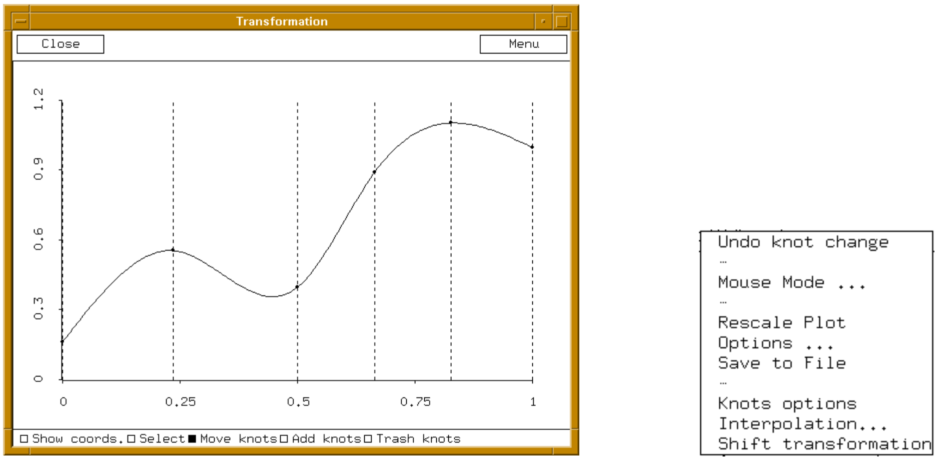

Figure 11: A transformation window with vertically restricted knots, and his menu.

A.1.2 The Transform menu

When a transformation window is in use in your screen, you will be able to access to his menu, see figure 11 (the menu can appear in the main menu bar or in the window itself, this depend on the window system in use). We describe here the items you will find in the menu, and some related questions.

Undo knot change Selecting this item will revert your transformation to the previous state, as before you have made the last knot change. Only the very last change 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 item for a LISP-STAT graph window. It is for choosing background colours and other graphic features.

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 11). 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... In the transformation window, the user decide about the position of the knots, and then the transformation is built by some interpolation method. 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 joinction the two polynomials have in common both the function value and the two first derivatives. The last method, Bezier b-splines, is not properly an interpolation method, the resulting curve not passing through all the knots. In this case, the knots become handles to control the curve shape. The transformation is built as a cubic spline such 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.

A.2 Using transformations from LISP-STAT programs

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 comunicate only with the transformation object, a `transf-proto` descendant, while the user will communicate only with the window transformation, a `transf-window-proto` descendant. The communication between the two objects will be transparent to you (and to the user, of course).

```
(load "transfor")

; create the transformation and store it in variable mytransf
; here we don't specify all the parameters for the call
(def mytransf
     (make-transformation ...
          :notice-to-owner (list originator :transf-has-changed)
          ...))

; this method will be called when 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 12: A typical use of transformation objects.

Using transformation objects in a lisp program involves three steps: creating the transformation object, creating the window in which the transformation is shown and modified by the user, and asking the transformation object to transform some values according the current state of the transformation. This last step needs to be executed again whenever the user makes any change to the transformation, so some mechanism is needed to automate it. The transformation is created using the `make-transformation` function. The help documentation of this function is very comprehensive, covering all the key parameters you can use to taylor the transformation to suit your needs. The transformation window is automatically created by default. You must provide an important key parameter: `notice-to-owner`, a list of an object and the message that you want to be sent to that object whenever some change occurs to the transformation.

A.2.1 Some examples

In KDE objects, transformations are implemented in such a way that the user has complete control over it. Through a menu item (only accessible when variable bandwidth is in use) the user pops up the transformation window. Internally, the message `:create-bw-transformation` is sent to the kde object. As `:notice-to-owner`, a list consisting of the kde object itself and the message

```
(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 13: A more refined call to create a transformation object.

`:redraw-window :force t` is passed. This mean that every time that the user makes some change to the transformation window, the KDE object will receive this message, and the kernel density window will be redrawn, forcing recomputation of it all.

For another 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 will have some method, say `recompute-model` for doing all 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 12. A more complete call to `make-transformation` can include more parameters in the form of key/value specification as shown in figure 13.

B Getting and running the software

B.1 XLISP-STAT and XLISP

Our KDE objects are build on XLISP-STAT, a statistical language developed by Luke Tierney. It is an extension of XLISP, a lisp dialect developed by David Betz. To XLISP, Luke Tierney has added some important features for statistical programming. First, most of the numerical functions are vectorized: the same functions apply to numbers as well as to vectors or arrays. Second, an object-oriented paradigm has been included to support statistical modeling. And third, XLISP-STAT includes graphical objects for easy building of user interfaces.

Following the terminology of the author, XLISP-STAT is just an implementation of a generic specification, LISP-STAT. A full description of LISP-STAT can be found in TIERNEY[90], where the author gives a more complete description of the software than those included in the documentation distributed with the software.

XLISP-STAT is free software. It is currently available for UNIX/X11, Macintosh and MS-Windows environments. It is possible to obtain both the source code and some compiled versions by anonymous ftp to *umnstat.stat.umn.edu* in the directory `pub/xlispstat`.

B.2 Getting KDE software

Once you have XLISP-STAT running, you will need some files to be able to use KDE objects. KDE files can be fetch from
`http://halley.upf.es/stat/kde`. There you will find a README file with more detailed and upgraded directions to install and run the software. There is also a FILES file with the current contents of the directory. Once the files are installed, you can follow the directions in section 3. Following is a brief description of the files included in the current version of KDE.

`runkde.lsp` Loading this file is the normal way to access kde objects, it contains some functions for creating kde objects.

`kde.lsp` This is the main file, contains definition of the central prototype object, `kde-proto`, and his methods.

kde_conf.lsp Global var definitions and some utility functions.

`calckde.lsp` Contains numerical routines for computing kernel estimates and several utility functions.

`binning.lsp` Contains routines for binning data following several methods of binning and several criteria for outside values.

`wkde.lsp` Contains the code to implement `wkde-proto` object, the prototype for windows that display KDE objects.

`transfor.lsp` Contains code to implement `transf-proto` and other transformation reated objects.

`funnorms.lsp` Contains some utilities for computing integrals and other functionals.

`plotline.lsp` Contains a modified verison of the XLISP-STAT function plot-lines.

`goldsear.lsp` In some XLISP-STAT versions, the function golden-search is documented but not defined. This file contains a lisp version of it.

`distrobj.lsp` contains code for `distrobj-proto`, a primitive distribution object prototype that allows easy building of mixtures.

`kdeaddpt.lsp` A modification of a `kde` object to show all the copies of the kernel function involved in the computation. Useful for better understanding of kernel technology, and for fun.

There are also other files containing some kde objects with real data sets, some of them quite popular: `buf-snow.kde, chrondite.lsp, income75.lsp, notes.kde, old-faithful.kde, Stamps.kde, Log-Suicide.kde.`

B.3 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 things 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) nedeed 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)
```

for calling a mathematical function,

```
(sin 0.36)
```

to create a KDE object,

```
(make-kde-from-file "myfile.dat")
```

or to load a file

```
(load "afile").
```

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 surronded 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 13 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 to the non-given arguments.