LFLM Version 1.0
An S-PLUS / R library for locally weighted fitting of linear models*

Henrik Aalborg Nielsen†

December 17, 1997

Abstract

The conditional parametric model is an extension of the well known linear regression model, obtained by replacing the parameters by smooth functions. Estimation in such models may be accomplished by fitting a, possibly larger, linear model locally to some explanatory variable(s). In this report the conditional parametric model is described together with a method of estimation. An S-PLUS / R implementation is described and an example given. Since the user interface is similar to other S-PLUS / R functions for regression the software is easy to use. Furthermore, to increase speed, the core of the program is written in the ANSI-C programming language. The software allows for experimentation with variable bandwidth selection procedures and evaluation structures.

1 Introduction

Conditional parametric models are considered in (Chambers & Hastie 1991), (Hastie & Tibshirani 1993), and (Anderson, Fang & Olkin 1994). These

*IMM Technical Report number 22-1997 downloaded from `http://www.imm.dtu.dk`.

†Dept. of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark. E-mail: `han@imm.dtu.dk`

models may be viewed as linear regression models in which the parameters are replaced by smooth functions of other explanatory variables. The purpose of the analysis is to estimate and make inference about these functions, without assuming a parametric form. The models considered in (Chambers & Hastie 1991) and (Anderson et al. 1994) are rather restricted, since the originating linear model need to be a strait line or a linear (hyper) surface. Conditional parametric models are a subset of the varying-coefficient models described in (Hastie & Tibshirani 1993), but the general description requires a more complicated estimation procedure.

In Section 2 a general description of the model is given, together with a method for estimation. In Section 3 an S-PLUS / R library which is able to estimate the smooth functions is described. An example on how to use the software is shown in Section 4. In Section 5 details on how to obtain the source code are given. Finally, in Section 6 we conclude on the paper.

2 Theory

2.1 Model

A conditional parametric model is a model of the form

$$Y_i = \boldsymbol{z}_i^T \boldsymbol{\theta}(\boldsymbol{x}_i) + e_i; \quad i = 1, \ldots, N, \tag{1}$$

where the response Y_i is a stochastic variable, $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$ are explanatory variables, e_i is i.i.d. $N(0, \sigma^2)$, $\boldsymbol{\theta}(\cdot)$ is a vector of unknown but smooth functions with values in $\mathcal{R}$, and $i = 1, \ldots, N$ are observation numbers. When $\boldsymbol{x}_i$ is constant across observations the model reduces to a linear model, hereof the name.

2.2 Local constant estimates

Estimation in (1) aims at estimating the functions $\boldsymbol{\theta}(\cdot)$ within the space spanned by the observations of $\boldsymbol{x}_i; i = 1, \ldots, N$. The functions is only estimated for distinct values of the argument $\boldsymbol{x}$. Below $\boldsymbol{x}$ denotes one single of these points and $\hat{\boldsymbol{\theta}}(\boldsymbol{x})$ denotes the estimates of the coefficient functions, when the coefficient functions are evaluated at $\boldsymbol{x}$.

One solution to the estimation problem is to replace $\boldsymbol{\theta}(\boldsymbol{x}_i)$ in (1) with a constant vector $\boldsymbol{\theta}(\boldsymbol{x})$ and fit the resulting model locally to $\boldsymbol{x}$, using weighted least squares. Below two similar methods of allocating weights to the observations are described, for both methods the weight function $W : \mathcal{R}_0 \to \mathcal{R}_0$ is a nowhere increasing function. The weight functions available in LFLM are listed in Table 1 on page 9.

In the case of a spherical kernel the weight on observation i is determined by the Euclidean distance $||\boldsymbol{x}_i - \boldsymbol{x}||$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}$, i.e.

$$w_i(\boldsymbol{x}) = W\left(\frac{||\boldsymbol{x}_i - \boldsymbol{x}||}{h(\boldsymbol{x})}\right). \tag{2}$$

A product kernel is characterized by distances being calculated for one dimension at a time, i.e.

$$w_i(\boldsymbol{x}) = \prod_j W\left(\frac{|x_i(j) - x(j)|}{h(\boldsymbol{x})}\right), \tag{3}$$

where the multiplication is over the dimensions of $\boldsymbol{x}$. The scalar $h(\boldsymbol{x}) > 0$ is called the bandwidth. If $h(\boldsymbol{x})$ is constant for all values of $\boldsymbol{x}$ it is denoted a fixed bandwidth. If $h(\boldsymbol{x})$ is chosen so that a certain fraction (α) of the observations fulfill $||\boldsymbol{x}_i - \boldsymbol{x}|| \leq h(x)$ it is denoted a nearest neighbour bandwidth. If $\boldsymbol{x}$ has dimension of two or larger, scaling of the individual elements of $\boldsymbol{x}_i$ before applying the method should be considered, see e.g. (Cleveland & Devlin 1988). Rotating the coordinate system in which $\boldsymbol{x}_i$ is measured may also be relevant.

Note that if $\boldsymbol{z}_i = 1$ for all i the method of estimation reduces to determining the scalar $\hat{\theta}(\boldsymbol{x})$ such that $\sum_{i=1}^{n} w_i(\boldsymbol{x})(y_i - \hat{\theta}(\boldsymbol{x}))^2$ is minimized, i.e. the method reduces to kernel estimation (Härdle 1990, p. 30). For this reason the described method of estimation of $\boldsymbol{\theta}(\boldsymbol{x})$ in (1) is called kernel or local constant estimation.

2.3 Local polynomial estimates

If the bandwidth $h(\boldsymbol{x})$ is sufficiently small the approximation of $\boldsymbol{\theta}(\cdot)$ as a constant vector near $\boldsymbol{x}$ is good. This implies that a relatively low number of observations is used to estimate $\boldsymbol{\theta}(\boldsymbol{x})$, resulting in a noisy estimate or large bias if the bandwidth is increased. See also the comments on kernel estimates in (Anderson et al. 1994)

It is, however, well known that locally to $\boldsymbol{x}$ the elements of $\boldsymbol{\theta}(\cdot)$ may be approximated by polynomials, and in many cases these will be good approximations for larger bandwidths than those corresponding to local constants. Local polynomial approximations are easily included in the method described. Let $\theta_j(\cdot)$ be the j'th element of $\boldsymbol{\theta}(\cdot)$ and let $\boldsymbol{P}_d(\boldsymbol{x})$ be a column vector of terms in a d-order polynomial evaluated at $\boldsymbol{x}$, if for instance $\boldsymbol{x} = [x_1 \ x_2]^T$ then $\boldsymbol{P}_2(\boldsymbol{x}) = [1 \ x_1 \ x_2 \ x_1^2 \ x_1x_2 \ x_2^2]^T$. Furthermore, let $\boldsymbol{z}_i = [z_{1i} \ldots z_{pi}]^T$. With

$$\boldsymbol{u}_i^T = \left[z_{1i}\boldsymbol{P}_{d(1)}^T(\boldsymbol{x}_i) \ldots z_{ji}\boldsymbol{P}_{d(j)}^T(\boldsymbol{x}_i) \ldots z_{pi}\boldsymbol{P}_{d(p)}^T(\boldsymbol{x}_i)\right] \tag{4}$$

and

$$\hat{\boldsymbol{\phi}}^T(\boldsymbol{x}) = [\hat{\boldsymbol{\phi}}_1^T(\boldsymbol{x}) \ldots \hat{\boldsymbol{\phi}}_j^T(\boldsymbol{x}) \ldots \hat{\boldsymbol{\phi}}_p^T(\boldsymbol{x})], \tag{5}$$

where $\hat{\boldsymbol{\phi}}_j(\boldsymbol{x})$ is a column vector of local constant estimates at $\boldsymbol{x}$ corresponding to $z_{ji}\boldsymbol{P}_{d(j)}(\boldsymbol{x}_i)$, estimation is handled as described in Section 2.2, but fitting the linear model

$$Y_i = \boldsymbol{u}_i^T\boldsymbol{\phi}(\boldsymbol{x}) + e_i; \quad i = 1, \ldots, N, \tag{6}$$

locally to $\boldsymbol{x}$. Hereafter the elements of $\boldsymbol{\theta}(\boldsymbol{x})$ is estimated by

$$\hat{\theta}_j(\boldsymbol{x}) = \boldsymbol{P}_{d(j)}^T(\boldsymbol{x})\,\hat{\boldsymbol{\phi}}_j(\boldsymbol{x}); \quad j = 1, \ldots p. \tag{7}$$

When $\boldsymbol{z}_j = 1$ for all j this method is identical to the method by Cleveland & Devlin (1988), with the exception that they center the elements of $\boldsymbol{x}_i$ used in $\boldsymbol{P}_d(\boldsymbol{x}_i)$ around $\boldsymbol{x}$ and so $\boldsymbol{P}_d(\boldsymbol{x}_i)$ must be recalculated for each value of $\boldsymbol{x}$ considered.

Example: If the first element of $\boldsymbol{\theta}(\cdot)$ is approximated locally by a 2nd order polynomial and if $\boldsymbol{x}_i = [x_{1i} \ x_{2i}]^T$ then z_{1i} in (1) is replaced by the elements $z_{1i}, \ z_{1i}x_{1i}, \ z_{1i}x_{2i}, \ z_{1i}x_{1i}^2, z_{1i}x_{1i}x_{2i},$ and $z_{1i}x_{2i}^2$. If the corresponding parameters are denoted $\phi_{1,0}, \ \phi_{1,1}, \ \phi_{1,2}, \ \phi_{1,11}, \ \phi_{1,12},$ and $\phi_{1,22}$ the estimate of $\theta_1(\boldsymbol{x})$ is $\hat{\phi}_{1,0} + \hat{\phi}_{1,1}x_1 + \hat{\phi}_{1,2}x_2 + \hat{\phi}_{1,11}x_1^2 + \hat{\phi}_{1,12}x_1x_2 + \hat{\phi}_{1,22}x_2^2$.

□

2.4 Heteroschadasity

For estimation purposes only observations in an neighbour-hood of $\boldsymbol{x}$ is used. Consequently, it is not required that e_i has constant variance. It is sufficient that the variance is approximately constant within a neighbourhood of $\boldsymbol{x}_i$, i.e. we may write e_i is i.i.d. $N(0, \sigma^2(\boldsymbol{x}_i))$.

It is possible to extend this to local polynomial estimation of $\sigma^2(\boldsymbol{x}_i)$. However, it is required that the local approximations used are strictly positive and it would be obvious to approximate $\log(\sigma^2(\boldsymbol{x}_i))$. Note that this may pose problems if the true variance is very small. Furthermore, it is necessary to use a local likelihood method (Hastie & Tibshirani 1990). These methods are not included in LFLM.

2.5 The smoother matrix

As for linear regression the fitted values $\hat{y}_i,\ i = 1, \ldots, N$ is linear combinations of the observations $y_i,\ i = 1, \ldots, N$. If the observations are arranged in a column vector $\boldsymbol{Y}$ this may be expressed as

$$\hat{\boldsymbol{Y}} = \boldsymbol{L}\boldsymbol{Y}, \tag{8}$$

where the $n \times n$ matrix $\boldsymbol{L}$ is called the smoother matrix and is independent of $\boldsymbol{Y}$. For linear regression the matrix is often called the hat-matrix (Jørgensen 1993).

The property (8) is shared with other smoothers (Hastie & Tibshirani 1990) and hence the model, variance, and error degrees of freedom can be calculated

$$\begin{aligned} df_{mod} &= \operatorname{tr}(\boldsymbol{L}) && (9)\\ df_{var} &= \operatorname{tr}(\boldsymbol{L}\boldsymbol{L}^T) && (10)\\ df_{err} &= N - \operatorname{tr}(2\boldsymbol{L} - \boldsymbol{L}\boldsymbol{L}^T) && (11) \end{aligned}$$

For linear regression df_{mod}, df_{var}, and $N - df_{err}$ are identical due to the special properties of $\boldsymbol{L}$ in this case.

To see that (8) is true let $\boldsymbol{U}$ be the design matrix corresponding to local constant estimates, i.e. row i is $\boldsymbol{u}_i$ from (4). Let $\boldsymbol{W}_i$ be a diagonal matrix, where element (k, k) is the weight on observation k when $\boldsymbol{x}_i$ is used as fitting point, i.e. $\boldsymbol{x} = \boldsymbol{x}_i$. The local constant (intermediate) estimates can then be expressed as

$$\hat{\boldsymbol{\phi}}(\boldsymbol{x}_i) = \left[\boldsymbol{U}^T\boldsymbol{W}_i\boldsymbol{U}\right]^{-1}\boldsymbol{U}^T\boldsymbol{W}_i\boldsymbol{Y}. \tag{12}$$

Hence the fitted value at observation i equals

$$\hat{y}_i = \boldsymbol{u}_i^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}_i) = \boldsymbol{u}_i^T \left[\boldsymbol{U}^T \boldsymbol{W}_i \boldsymbol{U}\right]^{-1} \boldsymbol{U}^T \boldsymbol{W}_i \boldsymbol{Y}. \tag{13}$$

Consequently, the vector of fitted values can be written as (8). If the weights are not chosen based on the values of $\boldsymbol{Y}$ or $\hat{\boldsymbol{Y}}$ then $\boldsymbol{L}$ only depend on $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$, $i = 1, \ldots, N$.

3 Software

The weighted least squares problem, described in Section 2, is solved by the algorithm described in (Miller 1992). The algorithm was originally written in Fortran but here a port to C by A. Shah is used. This was obtained as `pub/C-numanal/as274_fc.tar.z` by anonymous ftp from `usc.edu`.

The local constant estimation is implemented as a function written in ANSI-C. This is also true for most of the calculations related to the smoother matrix. The remaining part of the program is written in S-PLUS / R.

The S-PLUS / R front end constitutes a user friendly interface which is described in this section. Some experience with S-PLUS or R is assumed, including the notion of classes and methods (Statistical Sciences 1993). An example on how to use the software is given in Section 4.

Below a fixed width font indicates computer terms, e.g. files, S-PLUS / R objects, and arguments to S-PLUS / R functions. If the string is ended with "`()`" this indicates that reference to a S-PLUS / R function is made.

3.1 The main function

The main function of LFLM is `lflm()`. The arguments of this function are described in this section. Below the term "fitting points" is used to denote the points in which the estimates are to be calculated ($\boldsymbol{x}$ in Section 2.2). The handling of weight functions and bandwidth are inspired by the LOCFIT program by Clive Loader, Lucent Technologies, Bell Laboratories, see `http://cm.bell-labs.com/stat/project/locfit/index.html`.

3.1.1 Required arguments:

The arguments listed below must be supplied to `lflm()`.

- `formula`: Model specification, see Section 3.1.3.
- `alpha`: The bandwidth to use, by default this is interpreted as a fixed bandwidth. If a nearest neighbour bandwidth is used `alpha` specifies the fraction of observations to be covered for each fitting point. In this case `alpha` may have length two, the second element is then taken as a lower bound on the bandwidth found by the nearest neighbour method. To allow experimentation with variable-bandwidth selection procedures, it is also possible to specify an individual bandwidth for each fitting point. In this case the length of `alpha` must equal the number of fitting points, see also the description of argument `bt` in Section 3.1.2.
- `data`: Data frame containing the data.

3.1.2 Optional arguments:

Default values are supplied for the following arguments. These are by no means guaranteed to be appropriate, especially the bandwidth type `bt`, the degree of the local polynomial approximations `degree`, and the number / placement of fitting points `n.points` and `x.points` should be considered.

- `degree`: Degree of the local polynomial approximations. If the length is one this is used for all elements of $\boldsymbol{\theta}(\cdot)$. If the length is different from one it must equal the number of functions to estimate and the order must correspond to the global part of `formula`. Default: `2`.
- `CP`: If `TRUE` crossproduct terms are included in the local polynomial approximations. As for `degree` the length must either be one or equal the number of functions to estimate. Default: `TRUE`.
- `scale`: Vector, the length of which correspond to the dimension of $\boldsymbol{x}$. For each dimension of the fitting points and the corresponding observations the values are divided by the corresponding element of `scale` before distances, and consequently weights, are calculated. Default: No scaling.

- `bt:` Type of bandwidth; fixed (`"fix"`), nearest neighbour (`"nn"`), or user specified (`"user"`), i.e. an individual bandwidth for each fitting point. If user specified bandwidths are used the argument `x.points`, described below, must be supplied. Default: `"fix"`.
- `kt:` Type of kernel; spherical (`"sph"`) or product (`"prod"`), see Section 2.2. Default: `"sph"`.
- `kern:` Type of kernel or weight function; box (`"box"`), triangle (`"tangl"`), tricube (`"tcub"`), or Gaussian (`"gauss"`), see Table 1. Default: `"tcub"`.
- `n.points:` Number of fitting points per dimension of the data corresponding to the local formula, c.f. Sections 3.1.3 and 3.3. This argument is disregarded if `x.points` is specified. Default: `10`.
- `x.points:` Data frame containing the fitting points. The names and order must correspond to the local formula. This argument allows experimentation with evaluation structures. Default: Constructed using `n.points`.
- `na.action:` Function specifying the action to take when missing data (`NA`) is found in `data`, often `na.omit` is desirable. Default: `na.fail`.
- `weight:` Vector, which length is the number of observations with no missing values, specifying the weight on the observations. Default: unweighted.
- `calc.df:` Should the degrees of freedom of the model be calculated? In version 1.0 this is possible only when the estimates are calculated for all observations. Default: `FALSE`.
- `save.smoother.matrix:` Should the smoother matrix be calculated and saved? In version 1.0 this is possible only when the estimates are calculated for all observations. Default: `FALSE`.
- `circ:` Should the distances be calculated between points on the unit circle? This argument may only be `TRUE` for local constant estimates and for one-dimensional fitting points. Default: `FALSE`.
- `dump:` Should `lflm()` dump if an error from the C-code is trapped. Default: `TRUE`.

Name	kern argument	Weight function
Box	`"box"`	$W(u) = \begin{cases} 1, & u \in [0;1) \\ 0, & u \in [1;\infty) \end{cases}$
Triangle	`"tangl"`	$W(u) = \begin{cases} 1-u, & u \in [0;1) \\ 0, & u \in [1;\infty) \end{cases}$
Tribube	`"tcub"`	$W(u) = \begin{cases} (1-u^3)^3, & u \in [0;1) \\ 0, & u \in [1;\infty) \end{cases}$
Gauss	`"gauss"`	$W(u) = \exp(-u^2/2)$

Table 1: Weight functions available in LFLM.

3.1.3 Model specification:

Model specification (`formula`) follows the usual S-PLUS / R formula language

```
<response> ~ (<global formula>) | (<local formula>)
```

where the global formula is a S-PLUS / R model specification corresponding to $\boldsymbol{z}$ in (1), and the local formula corresponds to $\boldsymbol{x}$ in (1). The elements in the local formula must be separated by asterisks.

Example: If `x1`, `x2`, `z1`, `z2`, and `y` are numeric vectors (*not factors*) then

```
y ~ (z1 + z2) | (x1 * x2),
```

specifies the model $y_i = \theta_0(x_{1i}, x_{2i}) + \theta_1(x_{1i}, x_{2i})z_{1i} + \theta_0(x_{1i}, x_{2i})z_{2i} + e_i$. As usual the intercept term can be dropped from the model by replacing `z1 + z2` with `-1 + z1 + z2`.

Note: The function does not handle factor variables and functions of numeric variables included in the global formula correctly. Factor variables must be replaced by a number of coding variables before using the program. Similar steps must be used to include functions of numeric variables.

3.1.4 Output:

After successful completion `lflm()` returns a list of class `lflm` with the following components:

- `call`: An image of the call that produced the list.
- `data.name`: Under S-PLUS: The name of the data frame used for estimation. Under R: the string `"unknown"`; assign it manually after the call to `lflm()`.
- `run.time`: The date and time at which `lflm()` were called, as returned by `date()`.
- `formula`, `degree`, `CP`, `circ`, `kt`, `kern`, `bt`, `alpha`, and `scale`: Copies from the call to `lflm()`.
- `x.points`: Data frame in which the rows are the fitting points used.
- `est`: Data frame in which the rows correspond to the rows in `x.points` and the columns correspond to the function estimates.
- `df`: If requested; the degrees of freedom of the model, otherwise: `NA`.
- `S`: If requested; the smoother matrix, otherwise: `NA`.
- `bandwidth`: Vector containing the actual bandwidth used for each fitting point, i.e. the positions correspond to the rows of `x.points`.
- `rank.defic`: Vector containing non-negative integers indicating the rank deficiency for each fitting point, as reported by the WLS algorithm. A warning will be issued by `lflm()` if these are not all zero.
- `loc.nparam`: The number of local constants used, i.e. the number of parameters estimated by the WLS algorithm.
- `err.func`: If positive one of the C-functions in the WLS algorithm has returned an error condition. The value can be used to locate the function in the C code. By default a positive value will cause `lflm()` to stop without returning a result.
- `err.val`: If `err.func` is positive; the error value returned by the function, see (Miller 1992).

3.1.5 R notes

As mentioned above the name of the data frame used are not returned when the software runs under R. To be able to calculate the fitted values or the residuals, see Section 3.2, the correct name must be assigned to the list returned by `lflm()`. For instance if the result is saved in the list `fit` and the data frame containing the data are called `mydata`, the command `fit$data.name <- "mydata"` must be issued.

Furthermore, when using R (version 0.50), a call to `lflm()` will cause the warning `"some row names are duplicated; argument ignored"`. This can safely be disregarded.

3.2 Methods

Methods, corresponding to objects of class `lflm`, are supplied for the functions `coef()`, `fitted()`, `lines()`, `plot()`, `points()`, `predict()`, `print()`, `residuals()`, and `summary()`. These functions are briefly described below. Often these functions will be appropriate only for preliminary analyses and it will often be necessary to write application specific functions. Graphics are mainly handled for the case of $\boldsymbol{x}_i$ in (1) having dimension two or less, the plot method allows a call to `coplot()` for higher dimensions.

When $\boldsymbol{x}_i$ in (1) has dimension two or more methods using interpolation (`fitted()`, `predict()`, and `residuals()`) requires the fitting points to be placed in a grid similar to the grid generated when the argument `n.points` is used. However, the grid need not to be rectangular, it is sufficient that for all $i = 1, \dots, N$ a (hyper) cube of fitting points containing $\boldsymbol{x}_i$ can be identified.

Below the methods are described.

`print()` only print some of the components of the object. Since `print()` is called when an object is returned to the top level, or if the name of an object is just typed at the prompt, `unclass()` must be used to see the full content of the object.

`summary()` returns a list containing the formula, the degrees of freedom (if calculated), the number of local parameters (length of $\boldsymbol{\phi}$ in (5)), the `degree` and `CP` arguments from the call to `lflm()`, information of how weights were constructed, summary information of the estimates (incl. fitting points and bandwidth), and information on whether rank deficiencies were detected.

`coef()` returns a data frame, in which the first column(s) are the fitting points, followed by a column containing the bandwidth used (possibly on the scaled coordinates), the remaining columns contain the estimates of the functions at the fitting points.

`fitted()` returns the fitted values of the response. These are calculated using interpolation between fitting points.

`residuals()` returns the residuals using `fitted()`.

`predict()` returns predictions based on the fitted model. Three types of predictions are available through the `type` argument; (i) `"response"` (default) predictions of the dependent variable, (ii) `"terms"` each element of $\boldsymbol{z}^T\boldsymbol{\theta}(\boldsymbol{x})$ in (1) separately, and (iii) `"coefficients"` each element of $\boldsymbol{\theta}(\boldsymbol{x})$ separately. The calculations are performed using interpolation. By default missing values are returned if interpolation is not possible, it is possible to parse arguments to the function carrying out the interpolation.

`plot()`: By default, when $\boldsymbol{x}_i$ in (1) is one dimensional this function plots all the estimated functions; the user should set up the graphics devise to contain more than one plot before calling this function. Arguments can be parsed to the builtin plot function of S-PLUS / R. When $\boldsymbol{x}_i$ has dimension two, and S-PLUS is used, surface plotting using Trellis Graphics are available, also `coplot()` may be called. In other cases only `coplot()` may be called.

`points()` / `lines()` adds points / lines to the current plot, the argument `what` is used to specify the estimate to add and therefore should be among `names(coef(obj))`, where `obj` is a list of class `lflm`. These methods are only implemented for the case where $\boldsymbol{x}_i$ in (1) is one dimensional.

In the case that the data frame (`data`) contain missing values and `lflm()` is called with argument `na.action=na.omit` the functions `fitted()` and `residuals()` will return vectors of the same length as the number of rows in the data frame, but with missing values inserted as appropriate. This is not consistent with other regression functions in S-PLUS, but we find it more convenient for general use.

3.3 Surface Estimation

When $\boldsymbol{x}$ in (1) has dimension two or more, using the argument `n.points` to `lflm()` will result in a rectangular grid of fitting points being spanned parallel to the coordinate axes. Often, no observations will be present in the corners of this grid. In this situation we suggest that the fitting points are supplied directly through the `x.points` argument to `lflm()`.

To facilitate this process it is possible to use `lflm.data.grid()` to generate the rectangular grid and `in.chull()` to delete the fitting points not inside the convex hull spanned by the observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$. However, `in.chull()` works only when $\boldsymbol{x}$ has dimension two.

Note: The function `in.chull()`, supplied together with the software, is not used by other functions. Therefore, it can safely be deleted or renamed.

4 Example

The ethanol example of (Chambers & Hastie 1991, Section 8.2.2) is used since this clearly illustrates some of the differences between the standard S-PLUS function used of locally weighted regression, `loess()`, and `lflm()`. However, the strength of `lflm()` is mainly when, conditioning on one or two variables, there are more explanatory variables in the linear model than here. The example uses data stored in the data frame `ethanol` included in S-PLUS. Below ">" indicates the S-PLUS prompt and "+" indicates the secondary prompt.

The data are from an experiment with a single-cylinder automobile test engine using ethanol as fuel (Brinkman 1981). The dependent variable `NOx` is the amount of nitric oxide and nitrogen dioxide in the exhaust, normalized by the work done by the engine, and the unit is μg per joule. There are two predictors (i) the compression ratio `C` and (ii) the equivalence ratio `E`, a measure of the air to fuel ratio. There were 88 runs of the experiment.

The purpose of the analysis is to estimate the dependence of the expected value of `NOx` on `C` and `E`, without an a priory assumption of a specific parametric form. Since the convex hull spanned by the observations of the predictors is almost rectangular we will use fitting points spanning a rectangular grid of the predictors. In (Chambers & Hastie 1991, pp. 331-

335) it is shown that substantial curvature exists in the direction of the equivalence ratio E. For this reason a local quadratic approximation seems appropriate. Using a 50% nearest neighbour bandwidth the surface can be estimated using the command

```
> loess(NOx ~ C*E, span=0.5, degree=2, data=ethanol)
```

this will scale the predictors, by dividing them by their 10% trimmed sample standard deviation (Chambers & Hastie 1991, p. 315). With the exception of the fitting points used this can also be reproduced by use of `lflm()`. The command

```
> eth.surf <- lflm(NOx ~ (1)|(C*E), bt="nn", alpha=0.5,
+ degree=2, scale=sqrt(c(var(ethanol$C),var(ethanol$E))),
+ data=ethanol, n.points=15)
```

will scale by the untrimmed sample standard deviations and store the result as `eth.surf`. The estimated surface can be viewed by issuing the command

```
> plot(eth.surf, pt="wireframe", what="Intercept", zlab="NOx")
```

The plot is displayed in Figure 1. It seems that the "hill" runs parallel to the direction of the compression ratio C. As a first step we could then drop the cross-products between C and E by adding the argument `CP=F` in the call to `lflm`. However we will go directly to a conditional parametric model, in which the surface is linear in C, i.e. the expected value of NOx is modelled as $\theta_0(\mathtt{E}) + \theta_1(\mathtt{E})\mathtt{C}$, where $\theta_0(\cdot)$ and $\theta_1(\cdot)$ are smooth functions. Such a model is fitted by the command

```
> eth.cpm1 <- lflm(NOx ~ (1+C)|(E), bt="nn", alpha=0.5,
+ degree=2, data=ethanol, n.points=50)
```

Since an intercept term is included by default the formula `NOx ~ (C)|(E)` will be equivalent. As usual, a model without an intercept can be requested by replacing `1` with `-1`. The model just fitted includes third order terms in the local design matrix used. Hence, to mimic

```
> loess(NOx ~ C*E, span=0.5, degree=2, parametric="C",
+ drop.square="C", data=ethanol)
```

which is an example of how a conditional parametric model is specified in `loess()` (Chambers & Hastie 1991, p. 344), the command

```
> eth.cpm2 <- lflm(NOx ~ (C)|(E), bt="nn", alpha=0.5,
+ degree=c(2,1), data=ethanol, n.points=50)
```

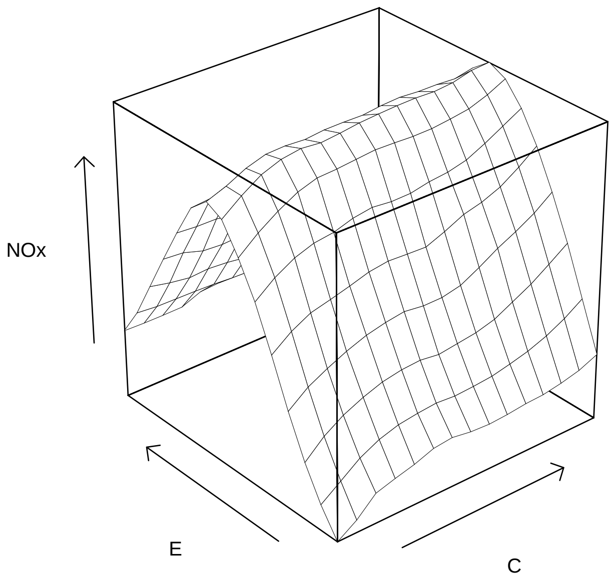

Figure 1: Wire-frame plot of intercept in `eth.surf`.

should be used. The fit `eth.cpm1` and the coefficient corresponding to `C` in `eth.cpm2` may be plotted by the commands

```
> par(mfrow=c(1,2))
> plot(eth.cpm1, type="l")
> lines(eth.cmp2, what="C", lty=2)
```

The resulting plots are shown in Figure 2 and as expected `eth.cpm2` results in more smooth estimates. Printing the bandwidth component of `eth.cpm1` or `eth.cpm2` reveals that at the leftmost point the bandwidth spans 56% of the axis and at the rightmost point the corresponding number is 43%.

Comparing with (Chambers & Hastie 1991, Section 8.2.2) it is seen that the results are presented quite differently from when `loess()` are used; `loess()` focus on the surface, while `lflm()` focus on the coefficient functions.

A page containing some simple diagnostics is obtained by the commands

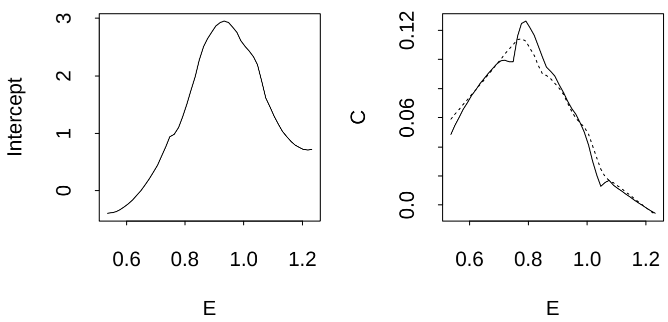

Figure 2: Estimated coefficient functions from `eth.cpm1` (solid) and the coefficient function corresponding to `C` from `eth.cpm2` (dotted).

```
> par(mfrow=c(2,2))
> plot(fitted(eth.cpm2),residuals(eth.cpm2))
> qqnorm(residuals(eth.cpm2))
> qqline(residuals(eth.cpm2))
> plot(ethanol$C,residuals(eth.cpm2))
> plot(ethanol$E,residuals(eth.cpm2))
```

The plots are shown in Figure 3 on page 17. The lower left plot indicates that the dependence on `C` is not strictly linear given `E`. Actually, in this case an additive model fitted by use of `gam()` is probably more appropriate, see (Chambers & Hastie 1991, Section 7.2.5).

To gain some understanding of the uncertainty associated with the estimates, bootstrapping of the residuals can be applied (Efron & Tibshirani 1993). In Appendix A.1 a program which will generate 200 bootstrap replicates of `eth.cpm2` calculated at 30 points of equal distance along the `E` axis is shown. The program also calculates pointwise estimates of the mean and the standard deviation.

The actual bootstrapping (the `for`-loop) took 141 seconds on a HP 9000/800. Figure 4 on page 18 shows the 95% standard normal intervals based on the bootstrap replicates. To interpret these as confidence intervals we need to assume that the model is correct and estimated without bias. As argued above this is somewhat doubtful. The plot was generated by the program shown in Appendix A.2.

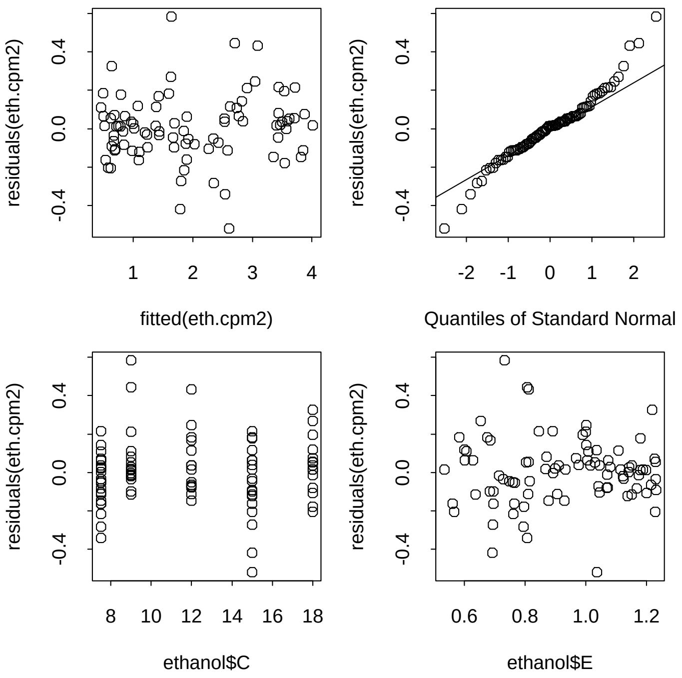

Figure 3: Simple diagnostics for the fit corresponding to `eth.cpm2`.

5 Obtaining the code and installation

The source code is found at `http://www.imm.dtu.dk/~han/lflm.tgz`

On UNIX systems: Place `lflm.tgz` in a temporary directory. Uncompress the file by executing `gunzip lflm.tgz` and unpack the resulting tape archive file by executing `tar -xvf lflm.tar`. Hereafter; follow the instructions in the file called `README`.

The program is known to compile and run under HP-UX 9 and 10, with S-PLUS 3.4 installed, and under RedHat Linux 4.0 (kernel version 2.0.18) with R 0.50 installed.

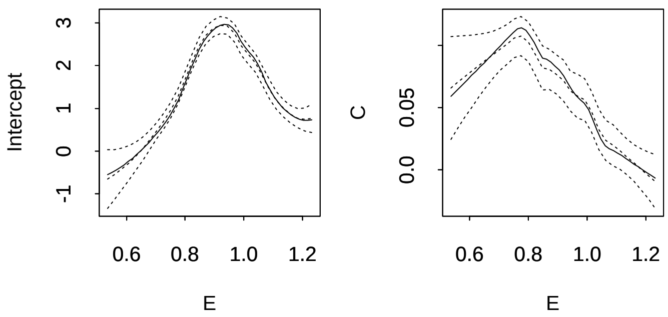

Figure 4: Mean and 95% standard normal intervals based on 200 bootstrap replicates of `eth.cpm2` (dotted), together with the original estimates (solid).

6 Conclusion

The conditional parametric model is reviewed, together with estimation using locally weighted fitting of a linear model derived from the original model. Furthermore, a software package for S-PLUS / R is described. The software can also be used for well known methods like kernel regression and locally weighted polynomial regression.

Since the user interface is similar to other S-PLUS / R functions for regression the software is easy to use. Furthermore, to increase speed the core of the program is written in the ANSI-C programming language. The software is flexible enough to allow experimentation with variable bandwidth selection procedures and evaluation structures. Also, the size of the regression problem which can be handled is solely determined by the hardware / operation system configuration.

A Sample S-PLUS programs

A.1 Bootstrapping

The following S-PLUS program was used to generate 200 bootstrap replicates of `eth.cpm2` produced by `lflm()` as shown on page 14. The program also calculates pointwise estimates of the mean and standard deviation.

```
eth.cpm2.resid <- residuals(eth.cpm2)
eth.cpm2.fitted <- fitted(eth.cpm2)
eth.cpm2.boot.1 <- matrix(nrow=30, ncol=200)
eth.cpm2.boot.C <- matrix(nrow=30, ncol=200)
eth.cpm2.boot.E <-
  seq(min(ethanol$E),max(ethanol$E), length=30)

for(b in 1:200) {
  tmp <-
    lflm(NOx ~ (C)|(E), degree=c(2,1), bt="nn", alpha=0.5,
         x.points=data.frame(E=eth.cpm2.boot.E),
         data=data.frame(C=ethanol$C, E=ethanol$E,
           NOx=eth.cpm2.fitted +
           sample(eth.cpm2.resid, length(eth.cpm2.resid), T)))
  eth.cpm2.boot.1[,b] <- coef(tmp)[, "Intercept"]
  eth.cpm2.boot.C[,b] <- coef(tmp)[, "C"]
}

eth.cpm2.summ <- vector("list", 0)
eth.cpm2.summ$Intercept <-
  data.frame(mean=apply(eth.cpm2.boot.1, 1, "mean"),
             sd=sqrt(apply(eth.cpm2.boot.1, 1, "var")))
eth.cpm2.summ$C <-
  data.frame(mean=apply(eth.cpm2.boot.C, 1, "mean"),
             sd=sqrt(apply(eth.cpm2.boot.C, 1, "var")))
```

A.2 Plotting

To produce the plot of the pointwise 95% confidence intervals shown in Figure 4 on page 18 the following S-PLUS program was used:

```
par(mfrow=c(1, 2))
for(what in c("Intercept", "C")) {
  matplot(eth.cpm2.boot.E,
          cbind(qnorm(p=0.025,
                      mean=get(what, eth.cpm2.summ)$mean,
                      sd=get(what, eth.cpm2.summ)$sd),
                get(what, eth.cpm2.summ)$mean,
                qnorm(p=0.975,
```

```
                         mean=get(what, eth.cpm2.summ)$mean,
                         sd=get(what, eth.cpm2.summ)$sd)),
           type="l", lty=2, col=1, xlab="E", ylab=what)
  axis(1)
  axis(2)
  box()
  lines(eth.cpm2, what=what)
}
```

References

Anderson, T. W., Fang, K. T. & Olkin, I., eds (1994), *Multivariate Analysis and Its Applications*, Institute of Mathematical Statistics, Hayward, chapter Coplots, Nonparametric Regression, and conditionally Parametric Fits, pp. 21–36.

Brinkman, N. D. (1981), 'Ethanol fuel—a single-cylinder engine study of efficiency and exhaust emissions', *SAE transactions* **90**(810345), 1410–1424.

Chambers, J. M. & Hastie, T. J., eds (1991), *Statistical Models in S*, Wadsworth, Belmont, CA.

Cleveland, W. S. & Devlin, S. J. (1988), 'Locally weighted regression: An approach to regression analysis by local fitting', *Journal of the American Statistical Association* **83**, 596–610.

Efron, B. & Tibshirani, R. J. (1993), *An Introduction to the Bootstrap*, Chapman & Hall, London/New York.

Härdle, W. (1990), *Applied Nonparametric Regression*, Cambridge University Press, Cambridge, UK.

Hastie, T. & Tibshirani, R. (1990), *Generalized Additive Models*, Chapman & Hall, London/New York.

Hastie, T. & Tibshirani, R. (1993), 'Varying-coefficient models', *Journal of the Royal Statistical Society, Series B, Methodological* **55**, 757–796.

Jørgensen, B. (1993), *The Theory of Linear Models*, Chapman & Hall, London/New York.

Miller, A. J. (1992), '[Algorithm AS 274] Least squares routines to supplement those of Gentleman', *Applied Statistics* **41**, 458–478. (Correction: 94V43 p678).

Statistical Sciences (1993), *S-PLUS Programmer's Manual, Version 3.2*, StatSci, a division of MathSoft, Inc., Seattle.