On Smoothing under Bounds and Geometric Constraints

Martin Löwendick, Essen

Dissertation zur Erlangung des Doktorgrades
vorgelegt dem Fachbereich 6
—Mathematik und Informatik—
der Universität GH Essen
im August 1999

Diese Dissertation lag dem Promotionsausschuß des Fachbereichs 6 – Mathematik und Informatik der Universität Gesamthochschule Essen vor. Als Gutachter haben mitgewirkt Prof. Dr. P. Laurie Davies (Essen) und Prof. Dr. Holger Dette (Bochum). Die Disputation fand am 4.11. 1999 statt.

Meiner Frau Beate und meinen Eltern.

Contents

Chapter 1

Introduction

quo quo scelesti ruitis
aptantur enses conditi?

Horace

The construction of functional models for pairs of measurements is a fundamental technique in many areas of science. Often there is a variability in measurement data which corrupts the assumed underlying signal. Thus it seems plausible to assume a model

$$\text{Data} = \text{Signal} \oplus \text{Noise}.$$

In terms of nonparametric regression, we arrive at the model

$$y = m(x) + \varepsilon(x) \tag{1.1}$$

It is now the main concern to separate the noise ε from the signal function m which is assumed to underly the data.

Usually, we deal with a data set of length $n + 1$ which reduces (1.1) to

$$y_i = m(x_i) + \varepsilon_i$$

for some $(x_i, y_i),\ x_i \in \mathbb{R}^k, y_i \in \mathbb{R}, 0 \leq i \leq n$. We will restrict ourselves to the case of $k = 1$. The ε_i are commonly assumed to be independent random variables. Dependencies in the measurements are treated with the techniques of time series analysis which will not be considered in this thesis.

The underlying model m is completely unknown. In contrast to nonparametric regression, parametric regression assumes the model belongs to some parametric family of functions, e.g. the affine linear family

$$F_\theta = \{\theta_1 x + \theta_2 | \theta \in \mathbb{R}^2\}.$$

A criterion is required which allows us to pick out one candidate from the continuum of hypothetical models F_θ. It is very common to choose a L^2 criterion because of its mathematical properties, e.g. the residual sum of squares

$$d(\theta) = \sum_{i=0}^{n} \left(y_i - (\theta_1 x_i - \theta_2)\right)^2$$

which defines a distance between the model θ and the data. The model minimizing d over all affine linear models is easy calculable and has been well studied in the statistical literature.

But what if the data do not have some linear structure? In this case even the optimal model candidate from F_θ cannot be adequate for the data (optimality defined in the L^2 sense from above). On the other hand, the choice of the criterion also influences the model: suppose the data to be roughly linear but some few observations are distanced from the rest of the data. Then it is well known that the L^2 criterion is badly behaved and θ^{ls} can be made to break down by distancing just one observation. We see that both the choice of parametric family and the criterion are crucial.

One way to remove the limitations of the parametric family is nonparametric regression. There is a variety of nonparametric techniques including local polynomials, wavelets, and smoothing splines. We demonstrate the approach by the kernel estimator and the penalized likelihood approach. Kernel estimators are defined by

$$f_h(x) = \frac{\sum\limits_{i=0}^{n} y_i K\left(\frac{x-x_i}{h}\right)}{\sum\limits_{i=0}^{n} K\left(\frac{x-x_i}{h}\right)}$$

where K is a kernel function K, i.e. K is a unimodal symmetrical probability density and h a bandwidth parameter. It turns out that the specific choice of K does not have major impact on the estimation but the choice of the bandwidth h is crucial. Again, one has to fix a criterion (and usually it will be a L^2 criterion) and h is then chosen to be optimal with this criterion. If we use the integrated mean square error we have the asymptotic optimal bandwidth of [Härdle (1991)]

$$h_{\mathrm{opt}} = \left(\frac{m(x) \int K^2(s)ds}{(n+1)\left(m^{(2)}(x) \int s^2 K(s)ds\right)^2}\right)^{1/5}$$

which heavily depends on the unknown signal m. There are several other proposals for automatic bandwidth selection including resampling techniques such as bootstrap and cross validation. But sometimes it seems impossible to find any appropiate bandwidth parameter at all, see Figure 1.1.

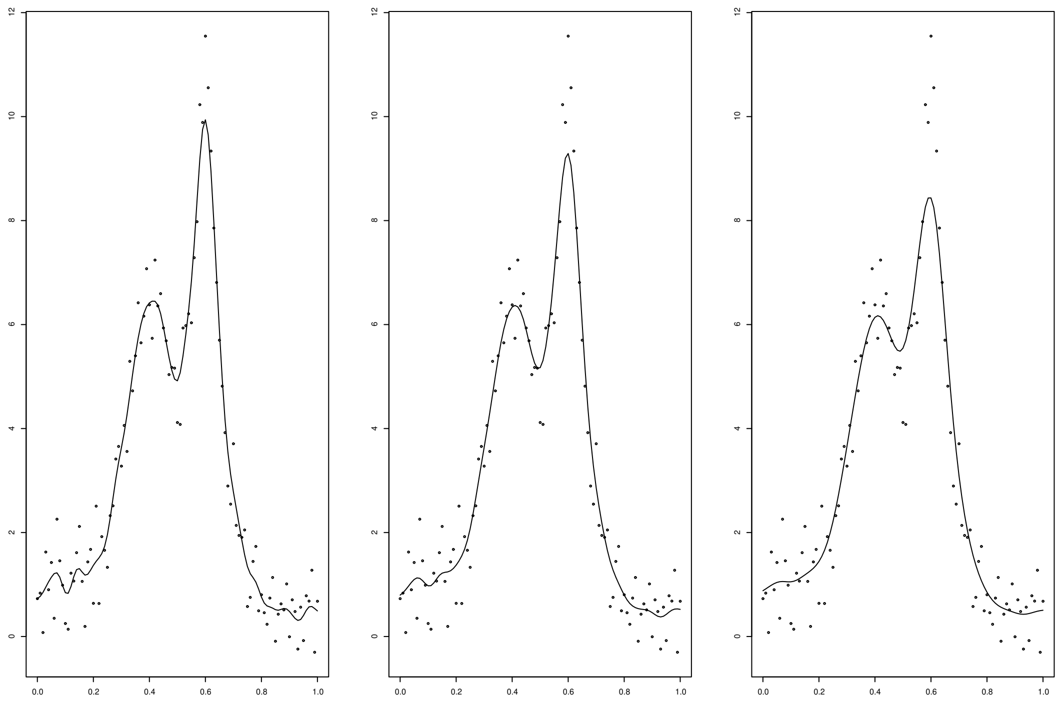

Figure 1.1: Data Set B.1 and three different choices of bandwidth for the gaussian kernel: A small bandwidth where spurios extrema occur, a compromise bandwidth with fewer spurios extrema but the extrema are underestimated, a large bandwidth where no spurious extrema occur but the extrema are strongly underestimated. No choice is satisfying!

Whatever bandwidth we select by whatever technique, the result is not fully satisfying because the graphical shape of the assumed underlying signal is not resembled in the model. If we use a small bandwidth the regression model yields many smaller local extrema which are artefacts, i.e. which come from the regression technique but not from the data. If we use a large bandwidth the peaks are underestimated and the minimum between the peaks is too high which is not desirable. Even the smallest bandwidth which resembles the right number of local extrema (which is known a priori here) is not yet appealing as shown in Figure 1.1.

One might use some sort local bandwidth selection which is more computer intensive. This is a lot better but does not fully solve the problem.

Another method of nonparametric regression is the penalized likelihood approach of smoothing splines which are defined to be the solution of the minimum problem

$$\sum_{i=0}^{n}(y_i - f(x_i))^2 + \lambda \int_{x_0}^{x_n} (f^{(2)}(t))^2 dt = \min .$$

Here a roughness penalty is introduced to avoid the maximum likelihood estimator degenerating. The introduction of smoothness in the context of nonparametric regression does not only happen for technical reasons but it has a justification of its own. The observer's eye will prefer the smoother model if forced to pick out one of two equally satisfying models. Again, the choice of the smoothing parameter λ is crucial and usually follows some L^2 criterion and resampling techniques.

We might assume now that these deficiencies come from the L^2 kind of criterion we used. This is right in parts, and there are ideas of more graphical or visual error criteria to be used in the regression context, e.g. by taking the local orthogonal projection to the curve as distance measure as done in [Marron and Tsybakov (1995)]. But full control of modality is not possible in this approach either.

We conclude that

- the number of extrema is a very important feature of the underlying signal which cannot be controlled by most existing nonparametric regression techniques and

- that smoothness is not only a user demand but a visual simplicity criterion for regression models.

It is the goal of this thesis to construct smooth regression models under control of the modality.

In Chapter 2, we give a short literature review of regression under constraints on modality. Then we explain two more recent approaches to nonparametric regression which *fully* control the modality and explore the modality structure from the data. These techniques based on runs in the signs of the residuals and on taut strings can be shown to have desirable properties. We also shortly comment on the complexity of the calculations which should not exceed order $n \log n$ to be practicable. Unfortunately, the results of these techniques are piecewise constant functions which do not satisfy the demand for smooth models. Thus we construct smooth models from them whilst retaining the modality of the underlying regression model. Because there must be some space for smoothing, we construct bounds that an appropiate smooth regression model should not leave. We present our computer code and show the performance on some data examples.

Section 3 gives theoretical and practical results of the attempts to find the smoothest regression candidate under the constraints we constructed in Chapter 2. The problem turns out to be theoretically solveable by means of convex analysis, but direct calculation of the solution is a complicated matter mainly because of the non–locality of the cubic spline solution and the discrete nature of the problem. We have made some improvement on an iterative procedure of [Metzner (1997)] called QSOR but the computation time in many situations is still not satisfactory.

Thus, if the smoothest regression function is too hard to find, why not give an easy calculable and sufficiently smooth function in the spirit of the modality constraints? This is exactly what we will do in Chapter 4 where a local parametric smooth function is proposed which is very fast calculable and yields appealing estimates for all simulation experiments we did.

Acknowledgements

I am gratefully indebted to my teacher and supervisor Professor P. Laurie Davies whose attitude to statistics has had a strong impact on my scientific interest. His scepticism in conventional foundations of statistics such as maximum likelihood and optimality is enlighting, his vast output of valuable ideas on ′how we can make things better′ is outrageous. He is an outstanding researcher, a charismatic teacher and tolerant and liberal as a superior. It was him who interested me in nonparametric regression because ″one can show some nice figures on this topic" and probably these figures at last lead to this thesis.

The warmest thanks go to my dear colleague ARNE KOVAC who accompanied my work from the very beginning and who helped me understanding many problems in uncountable discussions and by just adressing the right question to me. Many thanks for always delivering the latest source code of the string method, for proof reading, and for being a friend all the time.

I gratefully thank MARTIN SLOWIK for proof reading of the functional analytical part 3.1. He together with UWE WORTMANN and THOMAS BOGENSCHÜTZ sometimes was the support in the dark moments of demotivation and deliberation which every research worker knows very well.

I want to thank TANJA WICHMANN for delivering interpolation examples from her diploma dissertion [Wichmann (1998)].

For accepting to examine my thesis, I want to thank Professor HOLGER DETTE.

I acknowledge the financial support which I received from the DEUTSCHE FORSCHUNGSGEMEINSCHAFT, SFB 475 in the first year of the preparation of this thesis.

Last but not least I want to thank JOHANN SEBASTIAN BACH for writing *The Art of Fugue (BWV 1080)* and GLENN GOULD for playing it. This timeless piece of music and its marvellous interpretation inspired many parts of my work.

Essen, in August 1999
Martin Löwendick

Chapter 2

Extracting Shape From Data

alia bella geruntur, tu felix austria nube!

Habsburger philosophy

2.1 Nonparametric Regression

Much work has been done on the problem of nonparametric regression. We have already mentioned some of the different approaches in the Introduction such as kernel estimation (see [Jones and Wand (1995)] for a survey), penalized likelihood and smoothing splines ([Silverman (1985)] , [DeBoor (1978)]), wavelet thresholding ([Donoho et al. (1995)]), and local polynomials ([Fan and Gijbels (1996)]). None of these methods is directly concerned with modality although there is a strong need for nonparametric models with certain shape properties. Well known is the problem of estimating a distribution function where the nonparametric function must be isotonic. The first approach to isotonic estimation was made by [Barlow et al. (1972)]. A least squares isotonic function is calculated by the Pool Adjacent Violator algorithm which interestingly has some strong links to the very recent taut string method that we will explain in Section 2.3. The Pool Adjacent Violator calculates the greatest convex minorant of the cumulative sums of the data. This is a piecewise linear function whose piecewise derivative is the desired least squares isotonic estimator of the data. Clearly, it is piecewise constant and the height of those constance intervals is a local average of the data points which fall in the distinct interval. One can think of robustifying this approach by substituting the averages by local medians which can be done in at most order $n\log(n)$ complexity if some tricky, tree structured piecewise sorting routine is used, [Davies and Dümbgen (1999)]. Similar techniques can be applied to estimate production functions and Engel curves which have to be convex resp. concave, see [Hildreth (1954)] where the problem was first adressed. A more recent convex smoothing proposal can be found in [Li et al. (1996)]. We do not want to go in more detail here because the basic approach of inequality constrained smoothing applies in the same way to monotone and convex resp. concave smoothing. Thus we focus on monotonicity constraints and keep the possible generalization to higher order derivatives in mind.

In [Wright and Wegman (1978)] we find the first deeper theoretical treatment of constrained spline smoothing. Some results concerning the existence of the estimator are stated. The possibility of an infinite number of aditional knots in the spline solution gives rise to the introduction of the terminology of generalized splines, but the problem of calculation is not solved. To our knowledge this deficiency of the spline approach still exists. Even in the most recent work in this area we are aware of, [Mammen and Thomas-Agnan (1998)], where a new theoretical impetus is given by a two step procedure of first smooth then project to the set of constraints (which is mathematically a projection with respect to some Sobolev seminorm), the algorithmic proposal remains unprecise. The discretization step which is necessary in order to make the problem computer tractable at all makes the problem very complex in computation order. But despite this, the new presentation of the constrained spline as a projection of the unconstrained one projected to some constrained set is esthetically pleasing and allows for a better understanding of the estimator which e.g. leads to a generalization of the convergence rate results of [Utreras (1983)]. In [Mammen et al. (1998)], these projection approach is extended to a general framework for constrained smoothing. New algorithmic proposals are presented, e.g. for constrained local polynomial smoothing. The framework allows for a general treatment of different smoothing approaches.

The idea of a two step procedure goes back to [Friedman and Tibshirani (1984)] where a smoothing step (by kernel estimates) is combined with an isotonization step which is done by the Pool Adjacent Violator algorithm. To our knowledge this is one of the first papers which gives smooth isotonic regression models which are calculable in an acceptable computation complexity of order $n \log n$. The other possible approach of first isotonizing and then smoothing was first considered by [Mukerjee (1988)]. Clearly, if we smooth isotonized data with a positive kernel function we will achieve a smooth isotonic regression function. On the other hand, if we isotonize smoothed data, we have an at least contiuous isotonic model. Asymptotic properties of estimates of both types are studied in [Mammen (1991a)]. It is stated that both possible combinations of the smoothing and the isotonization step yield asymptotically equivalent estimators. For a smooth kernel function it seems to be better not to isotonize the observations before smoothing, for the choice of discontinuous kernel the result is just opposite. These results are based on the calculation of asymptotic bias and variance for the different estimates. A more recent approach to monotone smoothing is given in [Ramsay (1998)] where a representation of monotone twice differentiable functions as solution of a differential equation with an unconstrained coefficient function is given. This coefficient function is now approximated using a special basis expansion and a penalized likelihood approach. The roughness penalty used in this paper is the relative curvature which can be expressed easily by the coefficient function. Generalizations for constraints on higher order derivatives can easily be incorporated.

These techniques can be applied on monotonic intervals and merged to an estimator of given modality and positions of the local extrema. We refer to [Mammen (1991b)] where an inequality constrained least squares estimator is defined. The convergence rate of this regression estimator is shown to be the same as in the classical unconstrained case.

All the above techniques assume that the underlying signal is known to be monotone or that the number and locations of local extremes is given a priori. This is reasonable in some situations, e.g. if we are interested in a transformation which preserves ordering and thus is forced to be monotone. But in the general context of nonparametric regression, the assumed underlying signal is usually unknown. Thus we need data exploration techniques which estimate the modality structure of the data. Work in this direction has been done by [Dümbgen (1998a)] and [Dümbgen (1998b)] where linear rank tests are applied to local extrema. By inversion of the tests one gets bounds for regression functions, the minimum number of extrema consistent with the data, and location intervals for these extrema. The run method ([Davies (1995)], [Metzner (1997)]) which we will discuss in some detail further below may also be seen as inversion of a test, namely the run test for independence in a sequence of random variables. The tests inverted by Dümbgen in [Dümbgen (1998a)] and [Dümbgen (1998b)] can be shown to have better convergence rates on standard test beds but at the cost of a greater computational complexity. Another approach has been suggested by [Hengartner and Stark (1995)] where the Kolmogoroff ball centered at the empirical distribution is used to obtain nonparametric confidence bounds for shape–restricted densities. In [Chaudhuri and Marron (1999)], the authors assess the significance of zero crossings of derivatives and use their results to provide a data explorative, graphical device for displaying the significance of the local extrema. The proposal is the application of an approach well known in the genre of computer vision. A more popular approach is that of mode testing. We refer to [Good and Gaskins (1980)], [Hartigan and Hartigan (1985)], [Silverman (1986)], and [Fisher et al. (1994)]. In [Mächler (1995)], the number of inflection points instead of the number of local extrema is used as a roughness penalty in a penalized likelihood approach. A semiparametric estimator is given which only contains significant inflection points.

A very promising approach to estimate a regression function under full control of modality is [Davies and Kovac (1999)] which makes use of taut strings. Taut strings are well understood in the context of fitting isotone functions. We already mentioned the slope–of–the–greatest–convex–minorant type estimator of [Barlow et al. (1972)] which was asymptotically analysed by [Leurgans (1982)]. The idea of taut strings in Kolmogoroff tubes for density estimation goes back to [Hartigan and Hartigan (1985)] where a test for unimodality of a density is derived. In [Davies (1995)] it was explicitly used to calculate density estimators. The reason for the usefulness of the taut string for providing densities under control of modality is that the taut string is a function of lowest modality in the Kolmogoroff tube. The first time the idea of taut string was used in the context of nonparametric regression was in [Mammen and van de Geer (1997)] where it is shown that the taut string is a special case of a penalized likelihood estimator. The penalty term is based on the norm of total deviation. A short description of the taut string as well as an order n algorithm is presented. But since functions of bounded variation are used and thus, the width of the tubes shrinks with order $n^{-2/3}$, one has to accept additional extrema — asymptotically and in practice. In [Davies and Kovac (1999)], the size of the supremum tube shrinks in order $n^{-1/2}$ which can be proven to estimate the modality consistently. This property is shared by the run method of [Davies (1995)].

Since the run method of [Davies (1995)] and the taut string method of [Davies and Kovac (1999)] are to our knowledge the only methods of regression under control of modality which achieve asymptotically the correct modality, we are now going to discuss them in some detail.

2.2 Runs in the Signs of Residuals

As already mentioned the run method (see [Davies (1995)] and [Metzner (1997)]) may be seen as the inversion of the run test for testing independence of a sequence of observations. This idea was first used in [Davies (1995)] to obtain bounds for adequate regression models. Adequacy or approximation is here measured by runs in signs of residuals of regression candidates. We only give a short description of the method: for a more detailed description and for proofs of the claims see [Metzner (1997)].

Let $s_i,\ 0 \le i \le n$ be a sequence of independently distributed random variables with
$$\mathbb{P}(s_i = 1) = \mathbb{P}(s_i = -1) = \frac{1}{2}.$$
We denote the length of the longest run, i.e. a subsequence of the s_i which consists entirely of 1 or -1, by r_n . r_n is the length of the longest run in the s_i. For any given $0 < \alpha < 1$, we can calculate the α–quantile of r_n,

$$\mathrm{qu}(n, \alpha, r_n) = \min\{m | \mathbb{P}(r_n \le m) \ge \alpha\}.$$

There is a simple asymptotic approximation for large n,

$$\mathrm{qu}(n, \alpha, r_n) \approx \lceil \log_2(n+1) - \log_2\left(-\log\left(\frac{1+\alpha}{2}\right)\right)\rceil - 1, \tag{2.1}$$

where $\lceil x \rceil$ denotes the smallest integer number $k \ge x$.

The idea is now that we want a regression model such that the signs of the residuals look like a sample of a sequence of independently distributed random variables with the same same distribution as the s_i. The expression 'look like' is made precise in terms of the length of the longest run in the signs of the residuals.

Fix an adequacy level α, say $\alpha = 0.95$ or $\alpha = 0.5$. Calculate the length of the longest run $\mathrm{qu}(n, \alpha, r_n)$ which occurs (with probability α) in a sequence of s_i. Now we obtain a regression model having minimal modality under the constraint that the signs of residuals of the model do not yield a longer run than $\mathrm{qu}(n, \alpha, r_n)$.

Stretching to the right: Start with an initially non–increasing function f. An upper bound for f at the design point x_i is given by

$$u_i(n, \alpha) = \min\{u_{i-1}(n, \alpha), \max\{y_j | i - \mathrm{qu}(n, \alpha, r_n) \le j \le i\}\} \tag{2.2}$$

and a lower bound is given by

$$l_i(n, \alpha) = \max\{l_{i+1}(n, \alpha), \min\{y_j | i \le j \le i + \mathrm{qu}(n, \alpha, r_n)\}\}. \tag{2.3}$$

We set $u_i(n, \alpha) = \infty$ and $l_i(n, \alpha) = -\infty$ if the maximum in (2.2) and the minimum in (2.3) are not taken over the whole range of $\mathrm{qu}(n, \alpha, r_n) + 1$ observations which gives infinite bounds at local extrema and at the beginning and end of the data. If the upper and lower bound intersect at some point then a non–increasing regression model can be no longer adequate and we switch to a non–decreasing function for which the upper and lower bounds can obviously constructed in an analogous manner. This process is continued until the end of the sample is reached. Repeat the procedure this time starting with a initially non–decreasing function and choose those bounds which minimize the required number of local extrema. The result of 'stretching to the right' as it is denoted in [Davies (1995)] is a lower bound for the number of extrema required for an adequate regression model as well as upper bounds for the location of those extrema.

The reverse procedure 'stretching to the left' refers to starting at the end and processing to the beginning of the sample, yielding lower location bounds and again a lower bound for the number for local extreme points. We cite the following theorem from [Metzner (1997)].

Theorem 2.1 *Let* (x_i, y_i) *be an arbitrary data set with* $x_i \neq x_j$ *for* $i \neq j$. *Then for any fixed maximum run length, the lower bound for required local extrema evaluated by stretching to the left and stretching to the right is the same.*

Thus the run bounds deliver consistent information about the number of required extrema. Simultaneously, we have existence intervals for those extreme points because we get right interval bounds by stretching to the right and left bounds by stretching to the left.

One can ask what happens if we increase the adequacy level under the additional constraint that the required number of local extrema is preserved. This coincides with a smaller maximum run length $k \leq \mathrm{qu}(n, \alpha, r_n)$. We cite [Metzner (1997)] again.

Theorem 2.2 *Let* (x_i, y_i) *be an arbitrary data set with* $x_i \neq x_j$ *for* $i \neq j$ *and* $k \leq \mathrm{qu}(n, \alpha, r_n)$ *a maximum run length which resembles the number of required local extrema* l *evaluated by stretching to the left/right with* $\mathrm{qu}(n, \alpha, r_n)$. *Let* $K_j,\ 1 \leq j \leq l$ *be the location intervals derived from run length* k *and* $Q_j,\ 1 \leq j \leq l$ *those derived from run length* $\mathrm{qu}(n, \alpha, r_n)$ *. Then*

$$K_j \subset Q_j$$

holds for every $1 \leq j \leq l$.

Thus, once the lower bound for the number of local extrema required for an adequate regression model is known, we can consistently squeeze the bounds by reducing the run length until the number of extrema increases.

We make some computional remarks.

Remark 2.1

- *The decision whether we should start isotonic or antitonic can be made without calculation of two complete versions of the bounds. One determines which of both versions is forced to build in an extremum first. If we have to include an extremum before the design point x_{iso} starting isotone and at least before x_{anti} starting antitone, then we should start isotone for $x_{\mathrm{iso}} \leq x_{\mathrm{anti}}$ and antitone otherwise. Thus, we can reduce the computional complexity.*
- *The speed of calculation can be optimized by an updating algorithm for the running maxima and minima in (2.2) and (2.3) which then delivers an overall calculation complexity of order n if the data set is ordered in the design points.*
- *The above remarks are coded in FORTRAN, the source code is printed out in the appendix, A.1. The program is a slight modification of a routine provided by Davies.*

Our idea in the preceeding was to find the simplest adequate model. The following theorem says that we succeeded in the construction of necessary bounds for such a model.

Theorem 2.3 *There is a function f between the bounds which has the following properties:*

- *f has exactly the required number of extrema and*
- *f satisfies the run condition.*

Proof of Theorem 2.3: By a simple construction, see [Metzner (1997)]. □

One can argue that once we have the run bounds, it is not too important that the regression model fulfills the run condition as long as it exhibits the right monotonic behaviour and lies between the bounds. A slight modification of the construction of the run bounds introduced by Metzner in [Metzner (1997)] called Fast Bounds delivers sufficient bounds for the run criterion, i.e. *every* function between those bounds automatically fulfills the run condition. Our experience is that Fast Bounds and real run bounds differ only slightly. We omit the details and call all functions between the bounds with the correct monotonic behaviour adequate.

Next, we want to investigate the asymptotic behaviour of the function f_n^{Run} of midpoints of the run bounds (where both are finite) based on the α–quantile of the length of the longest run. Let k_n^α the number of local extreme points of f_n^{Run}, I_j^{ex}, $1 \le j \le k_n^\alpha$ the intervals where f_n^{Run} is infinite and $x_j^{\mathrm{ex}}(n,\alpha)$ the midpoints of these intervals. For the following theoretical considerations, we specify a test bed on which we can compare our reconstruction f_n^{Run} with some true function f. Thus let for the moment

$$y_i = f(x_i) + \varepsilon(x_i) \tag{2.4}$$

with

- f having bounded continuous first derivative $f^{(1)}$ and exactly k local extreme values at the points $0 < x_1^{\mathrm{ex}} < \ldots < x_k^{\mathrm{ex}} < 1$
- $f^{(1)} = 0$ only for $x \in \{x_1^{\mathrm{ex}}, \ldots, x_k^{\mathrm{ex}}\}$
- the $\varepsilon(x_i)$ being i.i.d. random variables, having median zero and a bounded continuous density function in the neighbourhood of zero.

The following theorem holds.

Theorem 2.4 *Under the above assumptions for the test bed (2.4), the following statements hold:*

1. *There exists a constant $A > 0$ such that for all $\delta > 0$*

$$\lim_{\alpha\to 1} \liminf_{n\to\infty} \mathbb{P}\Big(\{k_n^\alpha = k\} \cap \{\max_{1\le j\le k} |I_j^{\mathrm{ex}}| \le \delta\} \cap \{\max_{1\le j\le k} |x_j^{\mathrm{ex}}(n,\alpha) - x_j^{\mathrm{ex}}| \le \delta\}\Big) = 1.$$

2. *For all $\delta > 0$*

$$\lim_{\alpha\to 1} \liminf_{n\to\infty} \mathbb{P}\left(\sup_{\{x \mid f^{(1)}(x)\ge\delta\}} |f(x) - f_n^{\mathrm{Run}}(x)| \le A\left(\frac{\log\log n}{\log n}\right)\right) = 1.$$

Proof of Theorem 2.4: This is Theorem 2.1 of [Davies and Kovac (1999)]. □

The theorem states that the modality is consistently estimated, the locations of the extrema are reproduced and that the function f is approximated with a slow rate. It is surprising that the run method yields good results (which we will demonstrate later on) even though the rate of convergence is very poor

$$\frac{\log\log n}{\log n},$$

compared with the optimum order of $n^{-1/3}$, see [Stone (1982)]. This indicates that speed of convergence is possibly not the only and the best property of a method. The run method for example has a poor rate but a natural robustness property in the following sense: consider a sequence of k observations which are much larger than their neighbours. The situation is almost philosophical: are these observations outliers or is there a structure in the data which should be reflected in the model? Clearly, the greater k is the more we are inclined to believe in a structure of the data. The run method incorporates a natural method of tuning for this problem. If the maximum run length is k or greater then the sequence of k distanced observations does not influence the construction of the bounds.

We show some examples to demonstrate the power of the method which is more convincing than an optimal convergence rate could ever be.

Example 2.1 *The Donoho–Johnstone test signals first introduced in [Donoho and Johnstone (1995)] are well known in the statistical literature. We rescaled these four signals (Doppler, Heavisine, Blocks, and Bumps) to have equal power (see [Kovac (1998)]), evaluated them at the regular design mesh of length 2048 and finally corrupted the results with i.i.d. $\mathcal{N}(0, 0.4)$ white noise. The recalibration allows for comparison between the signals possible because of the identical signal–to–noise ratio. For more details, see the Appendix, B.3. We derive the maximum run length*

$$k = \mathrm{qu}(2047, 0.5, r_{2047}) = 10$$

according to (2.1). After having calculated the number of extrema that have to be build in at this run length we squeeze the bounds further by reducing the runlength (i.e. increasing the level of adequacy) until no further squeezing is possible without taking another extremum into account. The results are displayed in Figure 2.1. For the Doppler signal, the run method with asymptotic runlength does not have a good resolution at the left hand side of the data which coincides with the natural robustness feature we described above. The discontinuities of the Heavisine signal are detected properly. The sharp peaks of the spectroscope–like Bumps signal are resolved while there is no systematical base line shift.

The intervals of infinite bounds are somewhat unpleasing. We can think of setting the values of the extrema and then monotonizing the bounds again. The same procedure can be applied to the boundary values. Sometimes it may be useful to the user to —within certain constraints— specify the values freely due to additional a priori information. On the other hand, we can postulate that the distance between the bounds and the data should not be too large, say 3 times the MAD (potentially after some outlier resection). This will improve the bounds especially for the Blocks data. We deal with tools for those purposes in the next subsections.

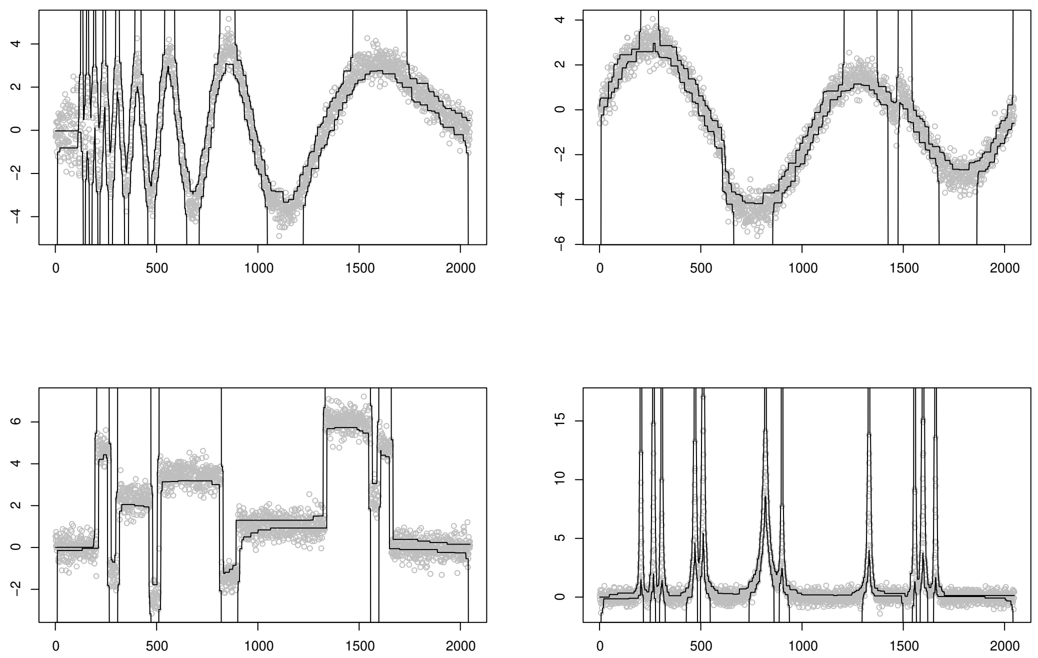

Figure 2.1: The four Donoho–Johnstone test signals and the run bounds (maximum run length 10, then decreased until another extremum has to be inserted).

2.2.1 Positioning of Extrema

Clearly it is sufficient to solve the problem of determining the height and location of a maximum. A simple proposal is to fit a parabola to the maximum data and choose the maximum of the parabola as an approximation.

Least Squares Quadratic Regression: Let $(x_1, y_1), \dots, (x_k, y_k)$ be a set of k data points and let without loss of generality the means of the x_i and y_i vanish. Then some tedious calculations show that the parameters a_{ls}, b_{ls}, c_{ls} minimizing the residual sum of squares

$$\sum_{i=1}^{k} \left(y_i - (ax_i^2 + bx_i + c)\right)^2$$

are

$$\begin{aligned}
a_{ls} &= \frac{k\left(\sum_1^k x_i^2\right)\left(\sum_1^k x_i^2 y_i\right) - k\left(\sum_1^k x_i^3\right)\left(\sum_1^k x_i y_i\right)}{-n\left(\sum_1^k x_i^3\right)^2 + n\left(\sum_1^k x_i^4\right)\left(\sum_1^k x_i^2\right) - \left(\sum_1^k x_i^2\right)^3} \\
b_{ls} &= \frac{\left(\sum_1^k x_i y_i\right) - \left(\sum_1^k x_i^3\right) a_{ls}}{\left(\sum_1^k x_i^2\right)} \\
c_{ls} &= \frac{\left(\sum_1^k x_i^2 y_i\right) - \left(\sum_1^k x_i^4\right) a_{ls} - \left(\sum_1^k x_i^3\right) b_{ls}}{\left(\sum_1^k x_i^2\right)}.
\end{aligned}$$

We are only interested in the extremum of the parabola which lies in

$$\left(\frac{-b_{ls}}{2a_{ls}}, c_{ls} - \frac{b_{ls}^2}{a_{ls}}\right).$$

The above formulae may easily be transformed for data in general position. The calculations can be done in order k. To test the performance of maximum detection methods we designed four test beds. The performance of the above method (which is implemented as a policy called `quadratic` in our code package) will be exemplified in Figure 2.2. We have not been too optimistic anyway, and we see that our expectations are met by the real performance of the least squares quadratic function approach. In particular the symmetry of

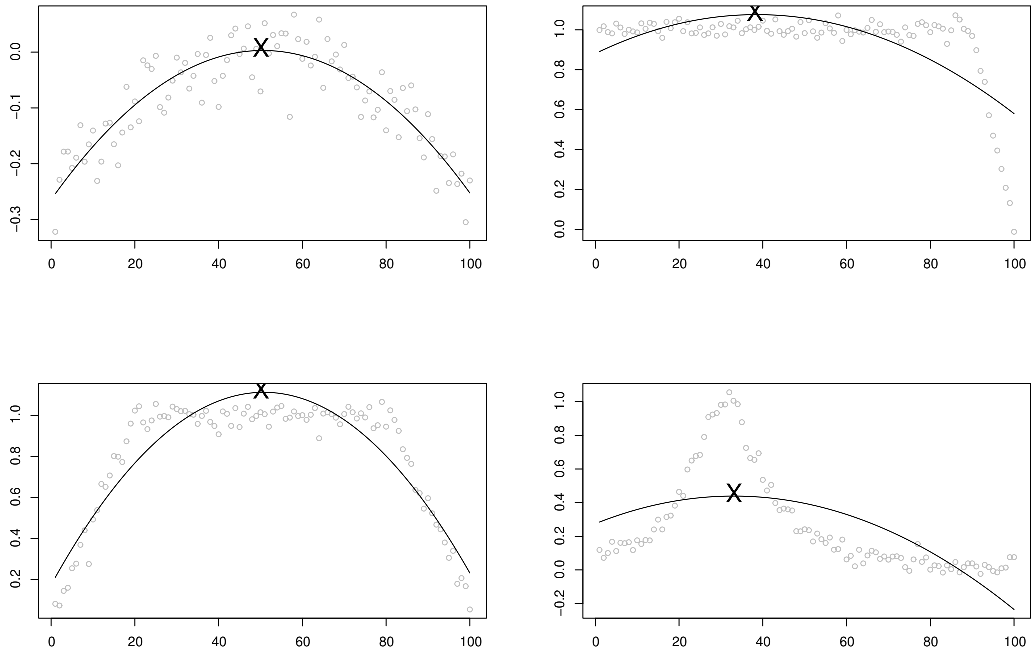

Figure 2.2: The `quadratic` policy for extremum fixing: the 'X' denotes the chosen extremum position, the line indicates the parametric fit.

the parametric model forces the fit not to follow the data which can be seen most clearly in the last of the four panels. Even though the maximum is well shaped and has an easy structure, the parametric fit completely fails to assign a suitable maximum height. We conclude that this parametric model is too restrictive.

Roof–type extremum fitting: We introduce another parametric fit for finding a suitable extremum position which is less restrictive than the parabola model. We propose to divide the maximum interval into two intervals. Then, in each interval, a least squares linear regression is calculated. The residual sum of squares in each interval is evaluated and added to a score

$$\sum_{1}^{s}(y_i - (m_1 x_i + b_1))^2 + \sum_{s+1}^{k}(y_i - (m_2 x_i + b_2))^2.$$

This must be done for all possible split points $1 < s < k$. Now we minimize the above score in s which provides the location of the extremum. The height of the extremum is not yet fixed exactly since there is obviously no need that the two regression lines must intersect in s. It is plausible to choose the height of the maximum as an arbitrary value between $m_1 s + b_1$ and $m_2 s + b_2$. An easy adhoc possibility is the mean, but we conjecture that we obtain sharper maxima and minima if we take

$$\max(m_1 s + b_1, m_2 s + b_2)$$

in the case of a maximum and

$$\min(m_1 s + b_1, m_2 s + b_2)$$

in the case of a minimum. We have analysed the method in the four artificial maximum situations. The parametric fit sometimes does not approximate the data very well but this is not our goal in the present situation. The chosen extremum positions are better, especially for the last test bed, compared to the quadratic regression case. The Cauchy–like maximum of the last panel is now treated well. But there is a deficiency in catching the symmetry of the third situation. Thus the less restrictive parametric approach not only takes a lot more computing time (because of its order k^2 complexity), but has a symmetry deficiency in its parametric structure as well. We may conclude that large variability of possible maximum situations is not captured by simple parametric models.

Least Squares Unimodal Regression: Using the Pool Adjacent Violator algorithm first introduced by [Barlow et al. (1972)] for isotonic least squares regression, we can give a method for exact least squares unimodal regression

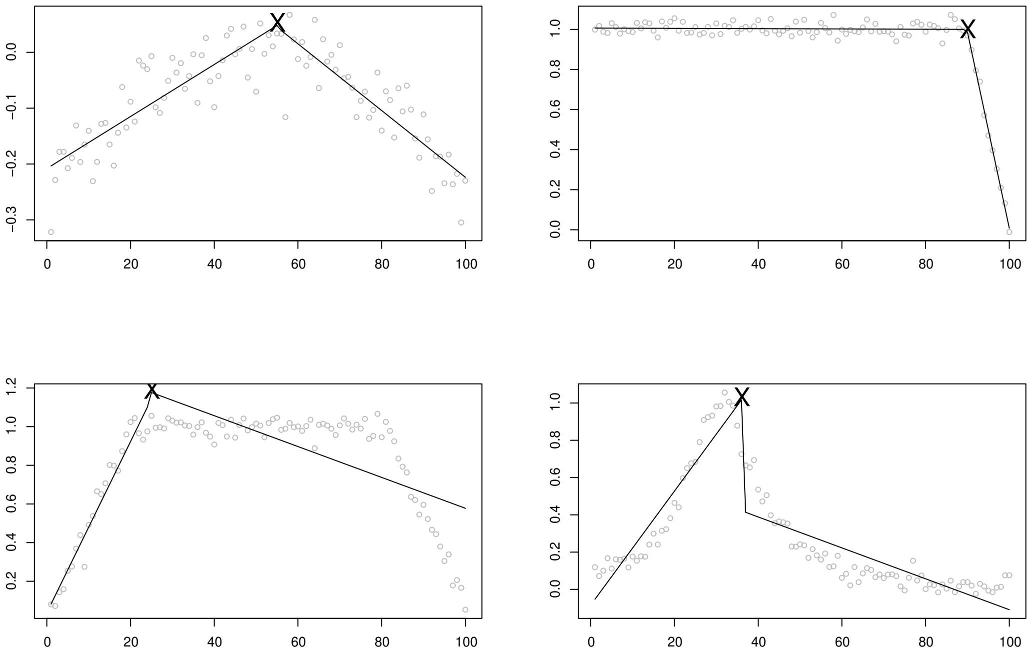

Figure 2.3: The `roof` policy for extremum fixing: the 'X' denotes the chosen extremum position, the line indicates the parametric fit.

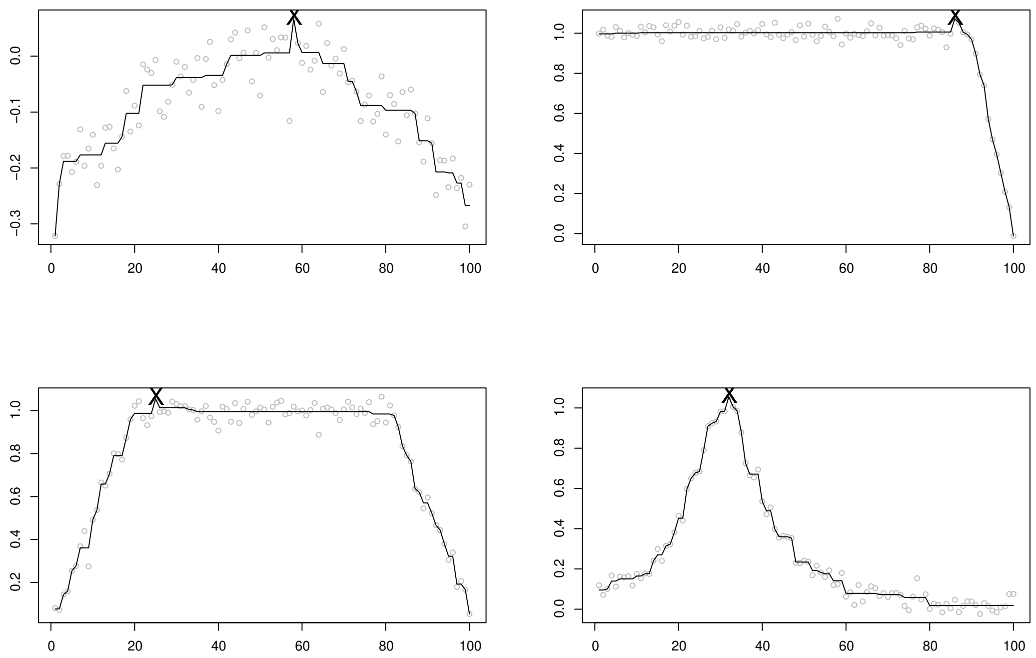

Figure 2.4: The `unimodal` policy for extremum fixing: the 'X' denotes the chosen extremum position, the line indicates the parametric fit.

with a computional complexity of order k^2. We divide the maximum interval again into two pieces at s and calculate the isotonic least squares solution on the left of s and the antitonic least squares solution to the right (algorithms for monotone least squares smoothing based on [Barlow et al. (1972)] are available on the internet, e.g. in the STATLIB archive). Then the scores are added again and minimized in the split point s which serves as an estimate for the location of the extremum. The height of the extremum is assigned similiar to the `roof` policy: maximum of what is left and right in a maximum, minimum of what is left and right in a minimum. There are no no parametric limitations and we expect a well–behaved fit. Due to the flexibility it can happen that the chosen extremum coincides with the largest observation which is not always desirable. We comment on that later on. Note that we can reduce the computional complexity to linear order in k if we calculate taut strings described below. Since asymptotically and in all our simulation experience extremum intervals are rather small compared to the length of the data set, this computational effort is perhaps of minor interest. Figure 2.4 shows that the nonparametric fit is best so far. The implementation of this policy is called `unimodal` and is default because in many regression situations we are interested in sharp extrema, e.g. in context of spectroscopy.

User–defined extrema: It is of some interest to consider user defined positions of the extrema and their values. For this purpose we provide an interactive program modus called `interactive`. The user fixes a position for each extremum by a mouse click in graphic display. The area of permissible choices is marked by a green box, no clicks outside are accepted. The results of the other procedures `unimodal`, `roof`, and `quadratic` are displayed so that the user can click them if one of the standard policies appears sensible for this extremum situation. We demonstrate this interactive procedure by using the Härdle data B.2 in Figure 2.5. The figure indicates that the `roof` policy is usually a compromise between the sensitivity of the least squares unimodal and the least squares parabola approach.

All of the above extremum procedures and in particular the least squares unimodal approach tends to be influenced by outlying observations as follows: if there is a very large observation in the maximum interval then the used procedure to locate the extremum will tend to interpolate this one observation. This is not desirable in every situation and is a consequence of the well–known senisitivity of L^2 criteria against outliers. Even if the noise is Gaussian the interpolation effect can have somewhat unpleasant consequences. Consider the Heavisine data where the smooth extrema should be well modelled by the least squares quadratic approach. We have constructed the run bounds with 6 i.e. the correct number of modes (which is known a priori in this artificial situation) and applied all three automatical extremum locators to it. In Figure 2.6, the original test signal and the estimated positions of extrema are shown. Clearly, the extrema near the discontinuity at 0.7 are more difficult to assign. As we expected, the least squares parabola method behaves well in the other, more smooth extremum situations due to the fact that $\sin$ has very parabola–like extrema. It can also be seen that both the other techniques tend to overestimate the extrema of the signal since large observations have strong influence in the estimation. In the present case the effect is not very dramatic but in the case of Cauchy noise outliers might easily force the regression model to distance itself widely from the signal. We already mentioned that the run method has some natural robustness which should not be removed by the method of choosing the distinct position of extreme values. It is natural to propose a standard robustifying method to downweight the influence of distanced observations.

'Winsorizing' the data: Let us assume that we are in the situation of a maximum interval $[x_{i_0}, x_{i_0+k}]$ so the upper run bound is infinite in this interval. We

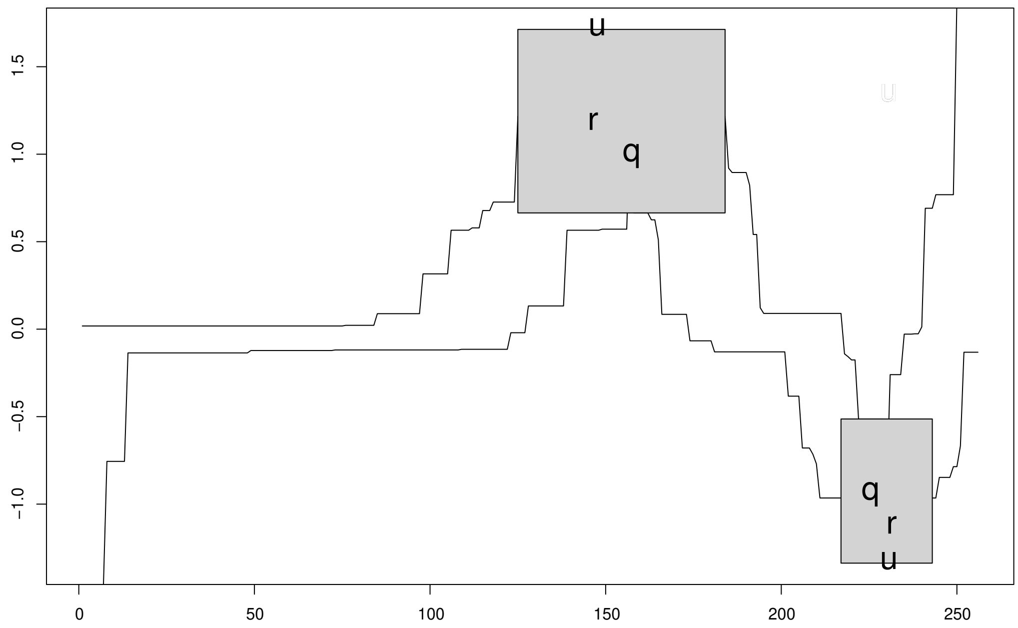

Figure 2.5: Interactive extremum positioning: a screenshot of `interactive` for the Härdle data B.2. The permissible extremum areas are grey shaded, the location of the other policies are denoted by their first letter. The least squares unimodal picks out the greatest observation (data points are left out for clarity)

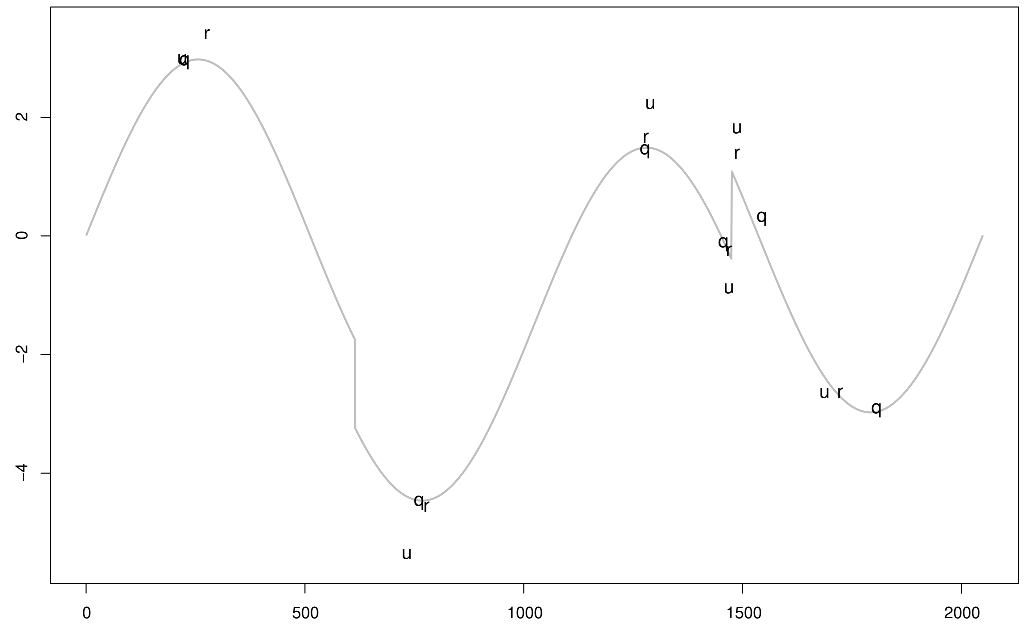

Figure 2.6: Comparing the three different extremum policies: the original Heavisine signal and the extrema positions based on the run bounds with 6 extrema. The policies are denominated by their first letter.

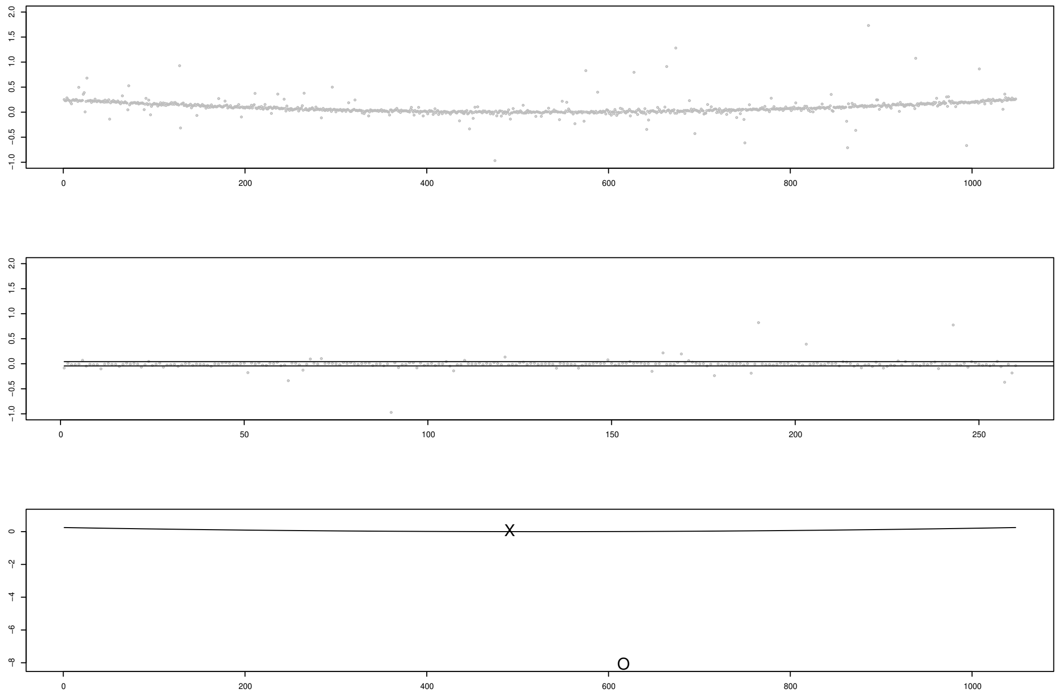

Figure 2.7: Winsorizing the extremum intervals: the upper panel shows a parabola corrupted with Cauchy Noise, the middle shows the shifted data and the $3 \cdot \mathrm{MAD}$ bounds, the lower panel shows the original curve with the estimated position of the extremum with ('X') and without winsorizing ('O').

shift the lower bound until it has the same median as the data in the maximum interval,

$$\tilde{l}_i = l_i - \mathrm{MED}(y_{i_0}, \dots, y_{i_0+k}) \quad i_0 \leq i \leq k,$$

and subtract this shifted lower bound from the data,

$$\tilde{y}_i = y_i - \tilde{l}_i.$$

Now we term every observation $\tilde{y}_i$ which does not lie in

$$(-3\mathrm{MAD}(\tilde{y}_{i_0}, \dots, \tilde{y}_{i_0+k}), 3\mathrm{MAD}(\tilde{y}_{i_0}, \dots, \tilde{y}_{i_0+k})$$

an outlying observation. Outlying observations are reset to the boundary of the above interval to which they are closest. Finally we invert the transformation the winsorized data by adding the shifted lower bound. For a practical example look at Figure 2.7. Other robustifying procedures will of course do perform equally well, but we do not want to make robustness a concern of this thesis. We omit further details.

2.2.2 Boundary Treatment, Tightening the Bounds, and other Refinements

Note that the run bounds we depicted up to now allow the specification of the left and right boundary value within a certain permissible interval. This is somewhat related to the problem of choosing extremum positions. Again, there is no theoretical reason to prefer a distinct choice of the boundary value since every permissible choice is adequate for the data at hand. Sometimes it may be useful to supply a priori information about the boundary values. This might be the case for the motorcycle crash data of [Härdle (1991)] where acceleration of the head of a motorcyclist dummy is measured against time: is seems reasonable to set the initial and final acceleration to 0. User–defined boundary treatment is then a way of dealing with this. Other data sets may not deliver such a priori information thus we need suitable defaults. We give a sketch of some. For simplicity, we restrict attention to the left end of the data set.

Median of first k: If we have run bounds with a maximum permissible run length of $k+1$ in the residuals, and say, the bounds start antitonic, then the upper bound for the first k points is infinite, i.e. the lower bound is the only restriction which must be hold by the left boundary value. By construction it is quite clear that

$$\mathrm{MED}(y_1, \ldots, y_k)$$

is greater than the lower bound at the left end of the data. Thus the median of the first k is permissible and gives a natural choice of the boundary values. This strategy is called `median` in our implementation, see Section 2.4. The only problem with the median is that it does not respect trends in the data because the particular sequence of the first k points is disregarded. Trends may be incorporated by the use of some regression method.

Linear Regression of first k For the reason described above we may think of linear regression methods to devise suitable boundary values. Since we are talking about few data points, we can apply some robust regression technique like least median of squares ([Hampel (1974)], [Rousseeuw (1984)]) or regression depth based estimator ([Rousseeuw (1999)]) which are somewhat expensive to calculate. Once we have drawn the linear estimator from the first k data points we evaluate it at the first data point. Truncate the value to be sure to have it between the bounds.

User–defined boundary values: Like for the choice of the distinct location of an extremum the user should have the possibility to pick out boundary values interactively. We designed an interface for that. The automatic choices are displayed for orientation and for the possibility of choosing different strategies at the left and right end of the data.

See Section 2.4 for coding details and on how to choose the different strategies for boundary values.

Tightening the bounds: We experimented with a method for globally shrinking the bounds until a further extremum has to be taken into account. This is lead by the idea that the distance between the model and the data should not be too large. Let us start e.g. with a scale of

$$s_n = \sqrt{2\log(n)}\mathrm{MAD}(y-m)$$

where

$$m = \frac{l+u}{2}.$$

is the mean of the run bounds. In extrema and at the beginning and end of the data, the mean is infinite, but this hopefully does not matter because MAD is a high breakdown scale estimator. Otherwise we restrict to those points where both bounds are finite. Then we take each observation and intersect the permissible values defined by the run bounds with $y_i \pm 3s_n$.This leads to new bounds

$$\tilde{l}_i = \max(l_i, y_i - 3s_n)$$

and

$$\tilde{u}_i = \min(u_i, y_i + 3s_n).$$

If the whole interval $y_i \pm 3s_n$ is outside the run bounds, i.e. the intersection is empty, we disregard the point y_i because it is distanced from the majority of the points. Then we isotonize the bounds again and check whether we have to build in a new extremum. If there is an additional extremum, we set $s_n \leftarrow 2s_n$ and try again. Then, by a bisection method, we can calculate the smallest scale s which can be applied to tighten the bounds under retaining the number of extrema. At least for continuous signals, this tightening procedure will usually not alter the bounds at all. The Blocks signal is effected by this procedure. As we can see in Figure 2.8, the strategy was succesful because the jumps are resolved more precisely now. We do not want to stress this point too much and leave out details because we do not think this is too important for the following.

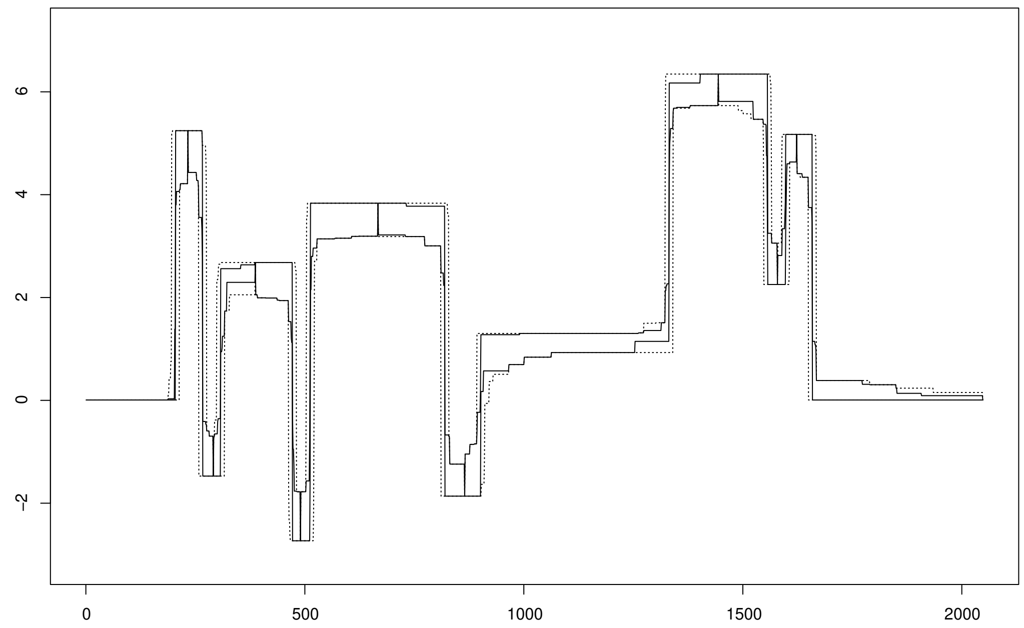

Figure 2.8: The Blocks signal with run bounds (dotted) and tightened run bounds. The discontinuities are very sharp after the tightening. The tightened bounds for the other Donoho and Johnstone signals differ only slightly from the original bounds.

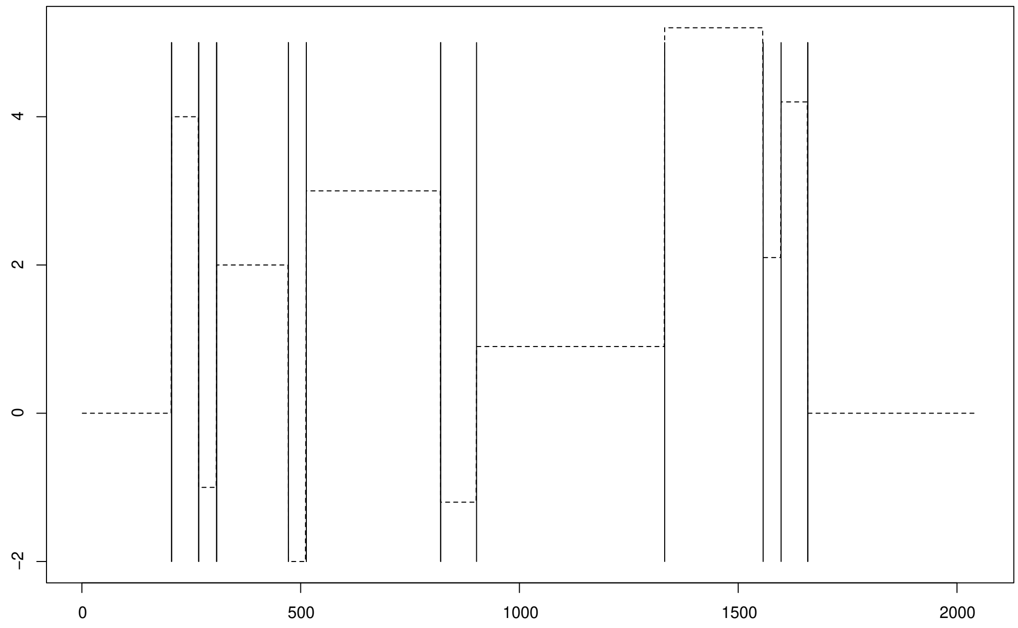

Figure 2.9: The original Blocks signal (dashed) and detected jumps after tightening. For the other signals, no discontinuities were detected.

Jumps or discontinuities: There is the possibility to detect possible jumps in the data. Clearly we have several different choices for the definition of a jump, and we are aware that the detection of jumps is a research field of its own. But it seems natural to talk about discontinuities if subsequent bounds have empty intersection, i.e.

$$[l_i, u_i] \cap [l_{i+1}, u_{i+1}] = \emptyset.$$

In order to allow rapidly increasing signals, we could go one step further and demand that the empty subsequent bounds must destroy monotonicity to be counted as jumps, i.e. for an isotonic region,

$$[l_i, u_i] \cap [l_{i+1}, u_{i+1}] = \emptyset \text{ and } u_i > l_{i+1}.$$

It is recommended to tighten the bounds before trying to detect jumps for sensitivity. We again do not want to stress this issue to much and simply point to Figure 2.9 for an application of the method.

2.3 Taut Strings

In the last section we termed adequacy of a model by runs in the signs of residuals which lead to bounds for an adequate function and to an adequate modality which was consistently estimated. A somewhat different concept of adequacy is introduced now: multiresolution analysis. Let us assume for the moment that the length $n+1$ of the data is a power of 2 and we are confronted with equidistant design,

$$x_i = \frac{i}{n}, \ 0 \leq i \leq n.$$

The more general case of irregular design of arbitrary length may be treated with sampling on a regular grid via linear interpolation or local linear regression, see [Kovac and Silverman (1999)].

Definition 2.1 *We consider partial sums over dyadic intervals,*

$$w_{jk} = \frac{1}{\sqrt{2^j}} \sum_{i=k2^j+1}^{(k+1)2^j} r_i \tag{2.5}$$

and call the w_{jk} multiresolution coefficients of the vector r. The multiresolution coefficients are local averages of the r_i up to the root factor. The residuals

$$r_i = y_i - f(x_i)$$

of a regression model are said to look like white noise if all their multiresolution coefficients are smaller than the threshold

$$\sigma\sqrt{\tau \log(n+1)}$$

where σ is some robust scale functional and $\tau > 0$ can be specified by the user.

Remark 2.2

- *The idea behind the threshold $\sigma\sqrt{\tau\log(n+1)}$ of Definition 2.1 is the following: From the tail approximation*

$$\mathbb{P}(|Z| \geq x) \leq c \exp\left(-\frac{x^2}{\tau}\right) \tag{2.6}$$

  for subgaussian random variables Z and some $\tau > 0$, we have

$$\mathbb{P}\left(|Z| \leq \sqrt{\tau \log(n+1)}\right) \geq 1 - \frac{c}{n+1} \to 1 \quad (n \to \infty).$$

For the normal distribution we have $\tau = 2$. *Since this still holds for sums of independent random variables*

$$\frac{1}{\sqrt{n+1}} \sum_{i=0}^{n} Z_i,$$

and the multiresolution coefficients are of this form, it is reasonable to choose a threshold like

$$\sigma \sqrt{\tau \log(n+1)}. \tag{2.7}$$

A wide variety of commonly assumed distributions is covered by this approach, e.g. the normal distribution. Even when the distribution is not known, the threshold is often a reasonable choice.

- *The restriction of* $n+1$ *being a power of* 2 *comes from the dyadic multiresolution coefficients being examined. This is for reducing computional complexity, of course we could examine all subintervals of the design mesh. Then multiresolution would have a computional complexity of* n^2 *instead of the present* $n \log n$.
- *We chose the robust scale functional mentioned above to be* $1.48 \cdot \mathrm{MAD}$ *of the highest level coefficients of some wavelet decomposition, see [Donoho et al. (1995)].*
- *For i.i.d. Gaussian noise, we would have to choose* $\tau = 2$ *as we mentioned above. This seems a little bit too tight in many situations. Davies and Kovac propose a default of* $\tau = 2.5$*, see [Davies and Kovac (1999)].*

Now we have an approximation or adequacy criterion. But there a two main questions about this are still open:

1. Why did we choose this kind of adequacy measure?
2. How can this criterion be adapted to deal with the purpose of modality controlled regression?

The first question has a simple answer: multiresolution analysis is sensible, fast calculable and gives information not only *if* an approximation is reasonable but also *where not* . For answering the second question, we introduce taut strings and tubes around the integrated data process.

Definition 2.2 *Let* $(x_i, y_i),\ 0 \le i \le n$ *be a data set of* $n+1 = 2^k$ *data points with the design points lying on the unit interval, and let for* $0 \le j \le n-1$

$$Y_j = \frac{1}{n} \sum_{i=0}^{j} y_i \tag{2.8}$$

denote the integrated data process (recall that we only consider equidistant design for simplicity and clarity). For every function $g : [0,1] \to \mathbb{R}$, we denote by

$$T(g,\rho) = \{h| \sup_{0 \le t \le 1} |g(t) - h(t)| \le \rho\} \tag{2.9}$$

the supremum tube around g with radius h. Now consider the taut string through this supremum tube and call it $S_{g,\rho}$. We can differentiate this (at least piecewise), the derivative will be denoted $s_{g,\rho}$. For the calculation of $s_{g,\rho}$ see [Hartigan and Hartigan (1985)] who first introduced taut strings within supremum tubes in the context of the DIP test of unimodality in density estimation. The taut string $S_{g,\rho}$ is a function of minimal modality in the supremum tube. If we calculate the taut string of the integrated data process $S_{Y,\rho}$, it is piecewise linear and thus $s_{Y,\rho}$ is piecewise constant.

The first and simple answer to the second question from above is now:

- Start with a great tube radius ρ
- Calculate the taut string $S_{Y,\rho}$ and $s_{Y,\rho}$.
- Calculate the multiresolution coefficients and check whether they are all smaller than $\sigma\sqrt{\tau \log n + 1}$.
- If they are, stop. If not, decrease ρ, e.g. set $\rho \leftarrow \rho/2$ and iterate.

This procedure is called global squeezing because the bandwidth ρ is squeezed globally. This approach is not fully satisfactory both theoretically and practically.

Example 2.2 *To make the idea of integrated data process more clear, we depict it in Figure 2.10 for the Haerdle data B.2. It is astonishing how smooth the integrated data process is compared to the level of noise we are confronted with at the stage of original data. The extremum intervals are not resolved very fine. A possible reason for this may be the suboptimal convergence rate which is stated in the next theorem.*

For the formulation of the next theorem of [Davies and Kovac (1999)] we need some notation: we denote the modality of the taut string $S(Y, C/\sqrt{n})$ in the supremum tube of the integrated data with radius $C/\sqrt{n}$ by k_n^C. The piecewise derivative of $S(Y, C/\sqrt{n})$ will be called S_n. We denote the intervals where the string attains its extrema by $I_j^{\text{ex}}(n, C)$ and the midpoints of those $x_i^{\text{ex}}(n, C)$. Recall that the extremum locations of the 'true' underlying signal we assume are denoted by x_i^{ex}.

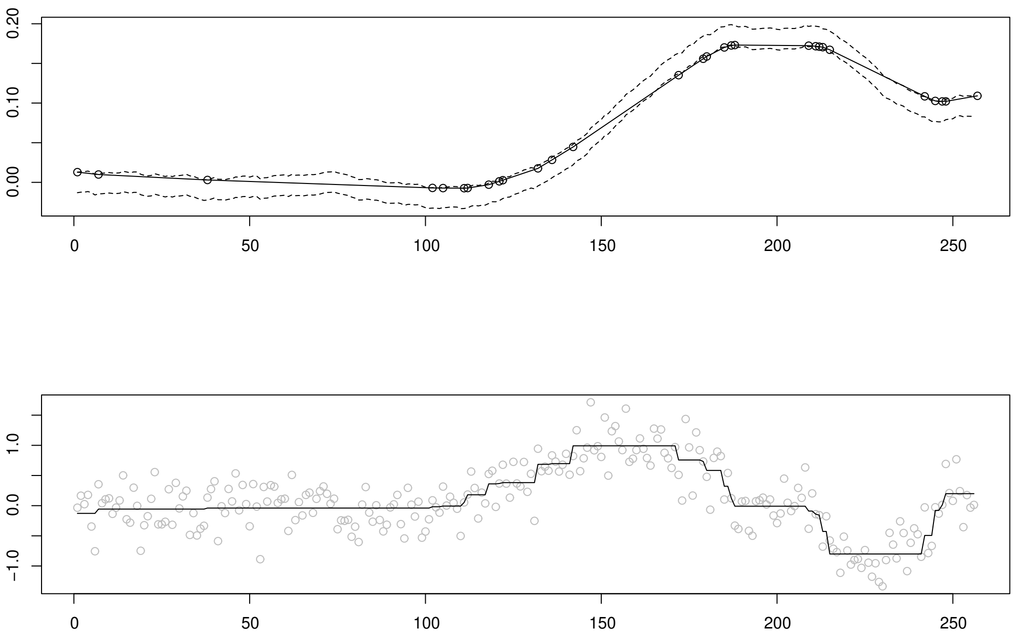

Figure 2.10: Supremum tubes around the integrated data process. In the upper panel the supremum tube with the taut string, in the lower panel the piecewise derivative of the taut string and the data.

Theorem 2.5

1. *Consider the test bed (2.4) but where the error distribution has mean* 0 *and a bounded second moment. Then for all* $\delta > 0$

$$\lim_{C\to\infty} \liminf_{n\to\infty} \mathbb{P}\Big(\{k_n^C = k\} \cap \{\max_{1\le i\le k} |I_i^{\mathrm{ex}}| \le \delta\} \cap \{\max_{1\le i\le k} |x_i^{\mathrm{ex}}(n,C) - x_i^{\mathrm{ex}}| \le \delta\}\Big) = 1. \tag{2.10}$$

2. *Consider the test bed (2.4) with the following additional requirements:*
   - f *has a bounded second derivative* $f^{(2)}$ *which is non–zero at the* k *local extrema.*
   - *the errors* $\varepsilon(x_i)$ *are sub–Gaussian with mean* 0*, i.e.*

   $$\mathbb{E}\left(\exp(\lambda\varepsilon(x_i))\right) < \exp\left(\mu\lambda^2\right)$$

   for some $\mu > 0$ *and for all* $\lambda > 0$.

   Then asymptotically, the location of the extrema is estimated consistently,

   $$\lim_{C\to\infty} \liminf_{n\to\infty} \mathbb{P}\left(x_i^{\mathrm{ex}} \in I_i^{\mathrm{ex}}(n,C), 1 \le i \le k\right) = 1. \tag{2.11}$$

   The asymptotic width of an extremum interval is of order $n^{-1/6}$*, more precisely*

   $$|I_i^{\mathrm{ex}}(n,C)| \sim (6C)^{1/3} |f^{(2)}(x_i^{\mathrm{ex}})|^{-1/3} n^{-1/6}.$$

   If we are away from the extrema and the boundary, i.e. for all

   $$x \in \left[A\left(\frac{\log n}{n}\right)^{1/3}, 1 - A\left(\frac{\log n}{n}\right)^{1/3}\right] \setminus \bigcup_{i=1}^{k_n^C} I_i^{\mathrm{ex}}(n,C),$$

   the convergence rate is $(\log n/n)^{1/3}$*, or more exactly*

   $$|f(x) - S_n(x)| \le O_{\mathbb{P}}\left(|f^{(1)}(x)|^{1/3}\left(\frac{\log n}{n}\right)^{1/3}\right). \tag{2.12}$$

   If we are in extrema intervals of the string derivative, i.e.

   $$x \in \bigcup_{i=1}^{k_n^C} I_i^{\mathrm{ex}}(n,C),$$

   we have

   $$|f(x) S_n(x)| \le (1 + o_{\mathbb{P}}(1)) A C^{2/3} n^{-1/3} \tag{2.13}$$

Equations (2.10) and (2.11) are complete analoga to the corresponding expressions for the run method in Theorem 2.4. In other words the string method estimates the number and location of extrema consistently. For the string method, more precise results are available, e.g. for the asymptotic order of the width of extremum intervals. According to (2.12) the string method attains the optimal rate of $n^{-1/3}$ up to a logarithmic factor in areas not in touch with the boundary or with extremum intervals. In extremum intervals, the optimum rate is $n^{-2/5}$ which is not attained, see (2.13). This is not only of theoretical interest because one can show that multiresolution analysis will detect this suboptimal rate at local extrema, see Theorem 3.3 of [Davies and Kovac (1999)]. Consequently one has to accept spurious local extrema. This effect can be avoided by local squeezing, i.e. squeezing of the supremum tube in shrinking neighbourhoods of the local extrema until all multiresolution coefficients are small enough. In [Davies and Kovac (1999)] the properties of the estimate f_n resulting from local squeezing are gathered in the following theorem. Since it is constructed by local squeezing, f_n per definitionem has the following properties:

- the local extrema of f are contained in the local extremes of f_n,
- f and f_n have the same monotonic behaviour between those extrema,
- the lengths of the intervals where f_n attains its local extrema are of order at least $(\log n/n)^{1/5}$, and
- the multiresolution coefficients of the residuals $y_i - f_n(x_i)$ are all smaller than the threshold (2.7).

Theorem 2.6 *Consider the model (2.4) and let f_n satisfy the above conditions. Then f_n realizes the optimal convergence rates in and outside extremum intervals, i.e. in extremum intervals we have*

$$|f(x) - f_n(x)| = O_{\mathbb{P}}\left(\left(\frac{\log n}{n}\right)^{2/5}\right). \tag{2.14}$$

outside the extremum intervals we have

$$|f(x) - f_n(x)| = O_{\mathbb{P}}\left(|f^{(1)}(x)|^{1/3}\left(\frac{\log n}{n}\right)^{1/3}\right), \tag{2.15}$$

The fact that (2.15) and (2.14) represent the optimal rates up to a logarithmic factor can be found in [Leurgans (1982)].

There is no unique way of calculating local squeezing practically. We adopted the implementation given in [Davies and Kovac (1999)] for the use in our context. In fact we used the author's original code which is contained in the software download avaible at the Statistics Group Home Page, see the appendix.

Thus, whether or not we squeeze locally resp. globally, we have an estimator which consistently estimates the modality structure of the data. Unfortunately, this estimator is piecewise constant which does not meet the demand for smooth models. In order to obtain room for smoothing we have to somehow construct bounds. Recalling that the string estimator consists of local means, i.e. the height of the estimator is the mean of the observations which fall into the particular constance interval, the first idea that comes to our mind is simultaneous confidence intervals for all the local means. If we have k_n constance intervals each of width b_i and local mean $m_i, 1 \leq i \leq k_k$, then an α-confidence region for all means simultaneously can be given by

$$m_i \pm c(\alpha)\hat{\sigma}_i\sqrt{\frac{2\log k_n}{b_i}}. \tag{2.16}$$

Roughly speaking, the denominator of the root term belongs to the i–th confidence interval, the numerator reflects the maximum taken over all confidence intervals for simultaneity, inspired by the asymptotic order of the maximum of k_n replications of normal random variables. The setting (2.16) has the advantage of having the correct asymptotic order for Gaussian white noise. Another advantage which is probably more important is that narrow intervals of constancy have broader confidence intervals, i.e. we have more room for a function between the bounds where its variation is great. This is a desirable property of the bounds.

Since we do not believe that the error distribution is known, the constant $c(\alpha)$ is not calculable as some sort of quantile. We made some experimental effort by setting

$$\hat{\sigma}_i = 1.48\mathrm{MAD}(r)$$

for all $1 \leq i \leq k_n$ and $c(\alpha) = 3$. Other settings might apply as well and can easily incorporated in our computer program requiring only minor changes in the code.

These simultaneous confidence bounds do not necessarily have the correct monotonicity as we can see in Figure 2.11 but this can easily be corrected by some postprocessing. Again, we have some freedom in choosing the boundary values and the position of the extrema. Since we commented on this in some detail in the last section, we briefly sketch the differences of the run bounds and the string bounds and refer to the last section for more details.

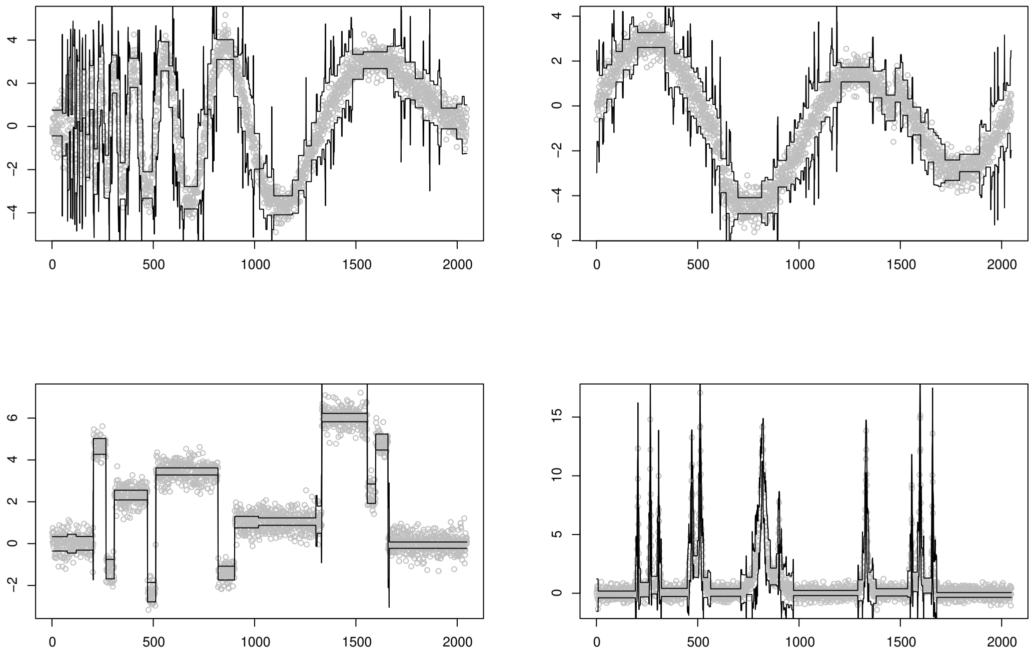

Figure 2.11: The four Donoho and Johnstone data sets and simultaneous confidence bands for the local means of the taut string estimator. The bounds do not necessarily have the correct monotonicity behaviour.

Extrema positioning: Now we are in the situation of having finite upper und lower bounds in minima and maxima which was not the case for the run bounds. An obvious default for the position thus is the mean of the upper resp. lower bounds which is the extremum value of the string estimator itself. All other extremum policies apply as well.

Boundary values: For the string method, we have finite bounds in the left and right boundary which makes it possible to choose the mean of the upper and lower bound as default for a starting value. It is not clear that the MED of the first k observations must lie between the bounds now, it is even not clear how to choose k. We propose to choose k such that the boundary intervals of Theorem 2.5 which were of the asymptotic order

$$A\left(\frac{\log n}{n}\right)^{1/3}$$

are reproduced, i.e.

$$k \sim n^{2/3}(\log n)^{1/3}.$$

Other possibilities might apply as well. Of course the other methods of Section 2.2.2 apply.

Examples of bounds constructed by the string method are shown in the next section.

2.4 Technicalities and Examples

Having explained all different possibilities to get bounds between which an adequate smooth regression model should lie, we give a short introduction to the computer code we designed to practically calculate those. For a deeper of the understanding of the procedures and for future modifications, we printed out the source code in the appendix, A.2. Because of the different strategies, policies, and methods we described so far, our program has to be very flexible. We depict the distinct choices the program leaves open in a flow chart, see Figure 2.12. We decided to make the code work under the GNU statistics package R (and, perhaps, under the commercial Splus from MathSoft) to be able to use easy graphical display and other comfort of a highly developed object oriented statistical programming language. R can be downloaded for free from the STATLIB archive and runs on nearly all platforms including WIN, Linux, and most Unix systems. First we throw away the design points (because the program works on integer design). Then we create a bounds object using the R

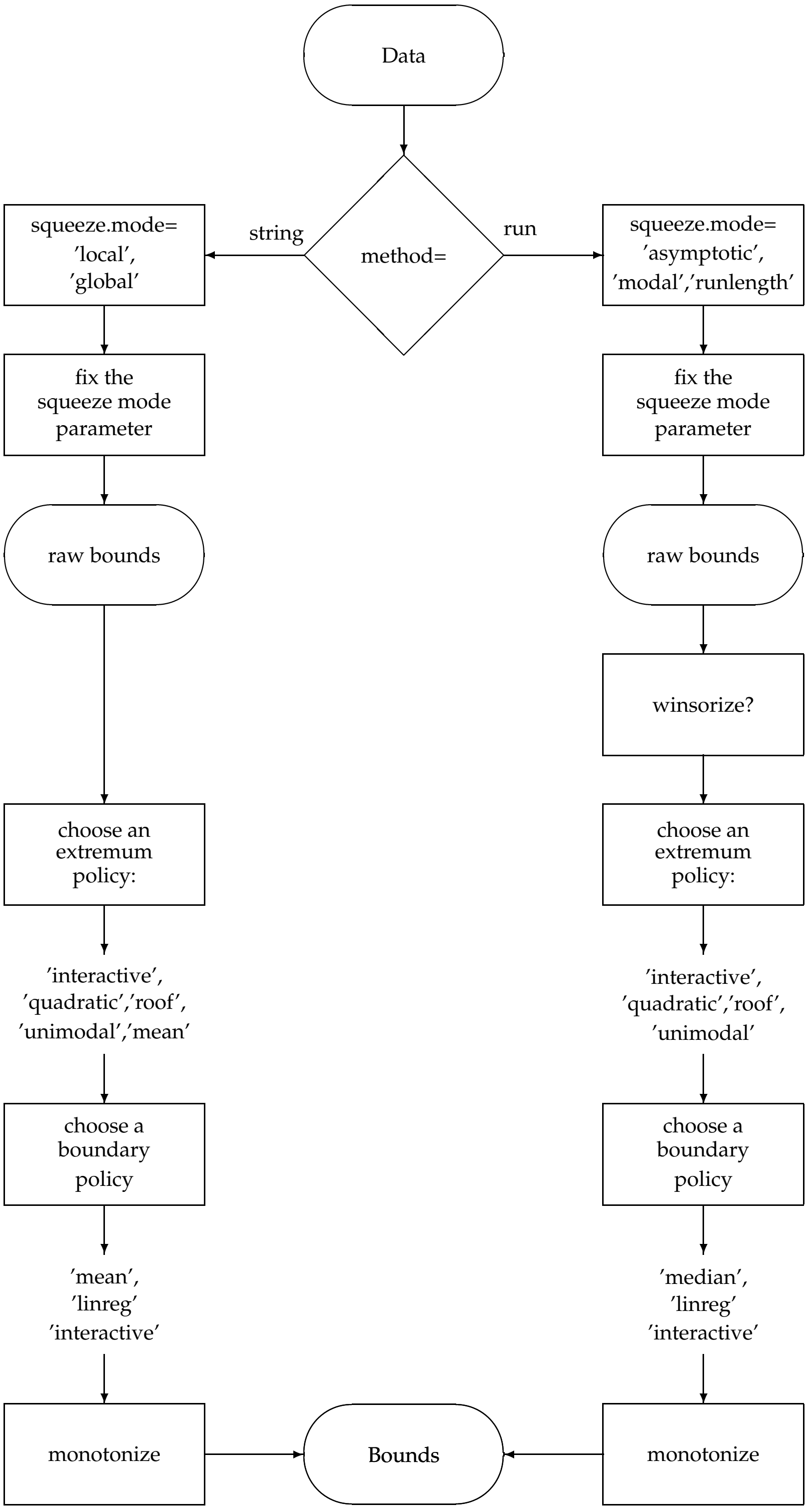

Figure 2.12: A flow chart for creating bounds from data. This is what the `create.bounds` function does, see A.2.

function `create.bounds` which has the following parameters and parameter defaults:

parameter	description	default
`data`	the data object (only y observations)	-
`method`	construction method	'string'
`squeeze.mode`	squeeze method	'local'
`squeeze.mode.par`	parameter for squeezing	2.5
`extpol`	extremum positioning policy	'interactive'
`bdrpol`	boundary value policy	'interactive'
`winsorize`	winsorizing at extrema?	T
`verbose`	graphics while running?	F

Clearly, there is a connection between the meaning of the squeezing parameter and the chosen squeeze mode.

`method`	`squeeze.mode`	`squeeze.mode.par`
'string'	'global'	threshold factor τ
	'local'	
'run'	'asymptotic'	quantile for the max. allowed runlength
	'modal'	number of extrema
	'runlength'	distinct runlength

Example 2.3 *Let us say the data sets we drew out of the Donoho et al. test signals were stored in the R objects* `do1` *to* `do4` *and we want the taut string bounds to be calculated using global squeezing with the default threshold factor. The extremum policy we choose is 'mean' while the boundary policy is chosen to be 'linreg'. Then the following function call calculates the bounds and stores it under the object name* `bds1`

```
bds1<-create.bounds(do1,methode="string",
                        squeeze.mode="global",
                        squeeze.mode.par=2.5,
                        extpol="mean",
                        bdrpol="linreg")
```

If we wanted the default combination of local squeezing string bounds and interactive policies to be calculated, things were less complicated and

```
bds1<-create.bounds(do1)
```

would serve as well.

To finish this Chapter, we depict bounds for the four Donoho et al. data sets with run and string method applied in the most recommended way. They can be reproduced by the following R commands:

```
b1.str <- create.bounds(do1,extpol="mean",
                            bdrpol="mean")

b1.run <- create.bounds(do1,method="run",
                            extpol='roof',
                            bdrpol='median')
```

We experimented with different combinations of strategies, but this seems to be a good compromise and should work well for almost all data sets. In the remaining chapters of this thesis, we use string bounds with the above parameter constellation if not denoted otherwise. The results of the above procedure are shown in Figure 2.13 and Figure 2.14.

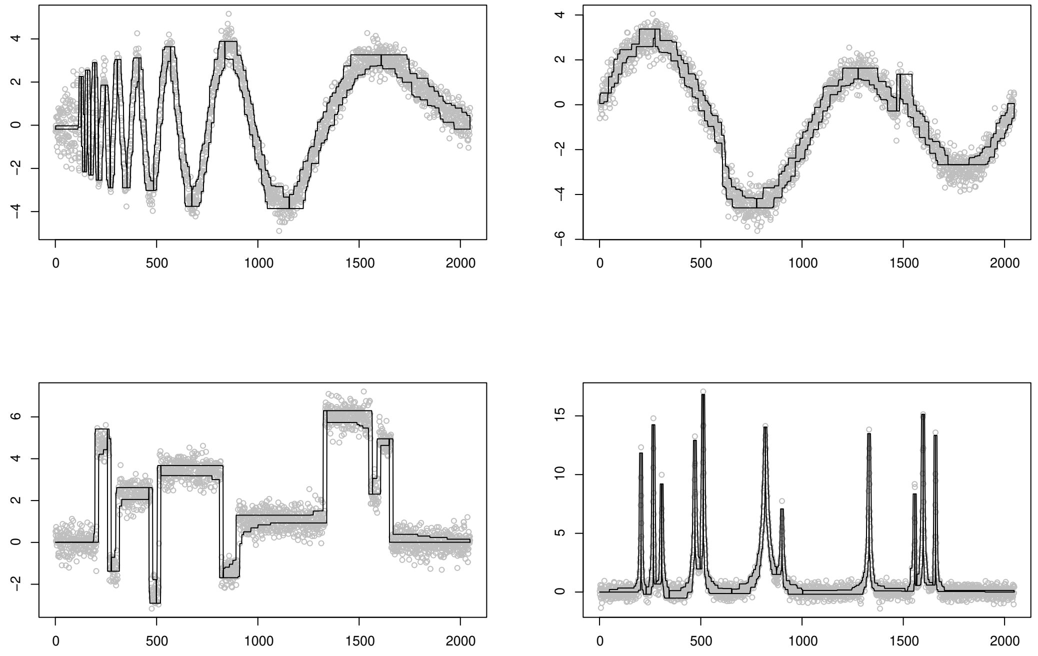

Figure 2.13: The results of the default parameter constellation for the run method applied to the Donoho and Johnstone test data. The `roof` policy for the extremum positions does not work well on the Blocks signal. We should choose another policy, e.g. `quadratic` or `interactive`.

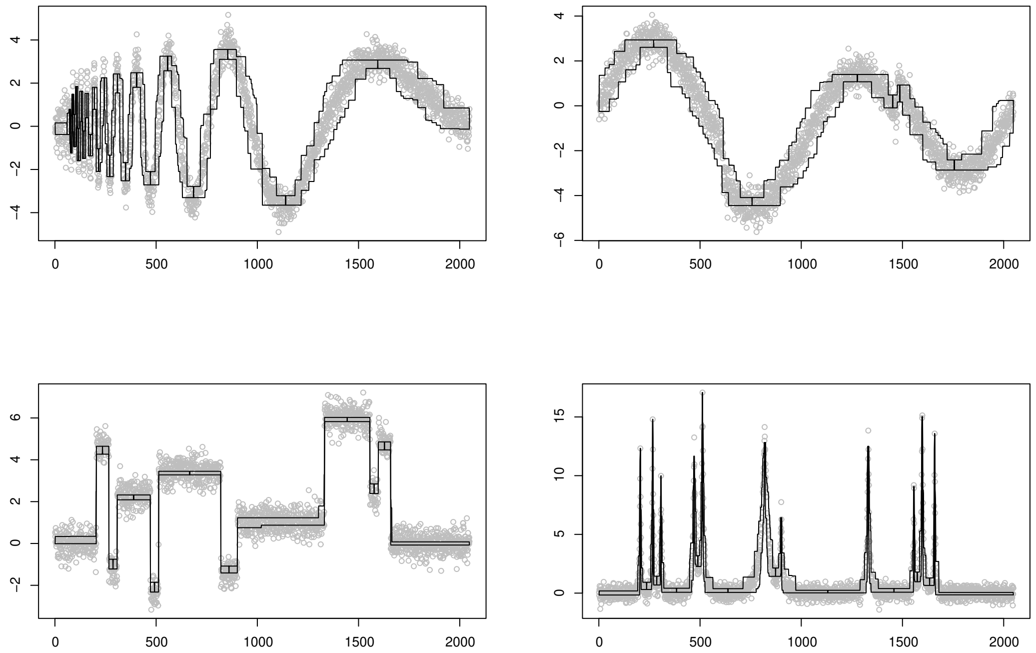

Figure 2.14: The result of the default parameter constellation for the string method applied to the Donoho and Johnstone test data.

Chapter 3

Minimum Roughness Approach

ultra posse nemo obligatur

idiom

First, we consider the problem of finding the smoothest function which satisfies the data driven constraints of the previous chapter. It turns out that the unique solution can be characterized by means of a Kuhn–Tucker–like functional Lagrangian Principle. The solution is a generalized cubic spline. A short survey of the related problem of inequality constrained spline smoothing follows.

The third section deals with discretization at the design points in order to make the problem computer tractable. We present the approach of [Metzner (1997)] and develop it a form of 'best approximating' discretization by a suitable modification of the smoothness integral approximation given there.

We explain the QSOR method which iteratively solves the discrete minimum roughness problem. We introduce a slight modification of [Metzner (1997)] which fills some gaps in the proof of convergence given there. An example demonstrates that the modification is not only necessary for the proof but also for the convergence of the method.

Unfortunately QSOR converges only slowly. We propose a hybrid method which makes use of our knowledge of the solution given in Section 3.1 and perform a benchmark on some very–hard–to–analyse data sets.

3.1 The continuous problem

Having constructed the bounds and monotonicity intervals of the last chapter, we wish now to obtain smooth regression functions which retain the graphical features we have derived from the data. Using the standard roughness measure

$$S(f) = \int_0^1 (f^{(2)}(t))^2 dt$$

we require the smoothest function subject to the constraints. Hence we arrive at the following minimization problem.

The Minimization Problem: Minimize the functional S for suitable f subject to the following constraints:

B For given points $0 = x_1 < \ldots < x_n = 1$ the function f must satisfy the following bounds

$$l_i \leq f(x_i) \leq u_i.$$

∂ Some boundary conditions hold: Either $f(0) = y_0,\ f(1) = y_1,\ f'(0) = y'_0,\ f'(1) = y'_1$ or $f(0) = y_0,\ f(1) = y_1,\ f''(0) = f''(1) = 0$.

M There exist disjoint intervals $I_j, j = 1, \ldots, k_i; A_j, j = 1, \ldots, k_A$ the closure of whose union is $[0, 1]$ and for which f is isotone on the I_j and antitone on the A_j.

Clearly the constraints must be consistent: We assume the existence of a smooth function which satisfies all constraints. If this is not the case, there is no problem. This claim naturally enters the theoretical considerations of the next section (Slater condition (3.3)).

Remark 3.1 *Let $\tilde{z}$ be the unique cubic function satisfying ∂. If z is a solution to the minimization problem, then $u = z - \tilde{z}$ is a solution to a transformed problem: Minimize $\tilde{S}(u) = S(u + \tilde{z})$ over a set of suitably transformed constraints, see [Hornung (1978)] for a discussion. Clearly, u and Du vanish in 0 and 1. Thus, without loss of generality we can restrict our attention to minimization problems having vanishing boundary values for the function and its first derivative in ∂.*

3.2 A characterization result

Definition 3.1 *We will need the following function spaces in the discussion of the minimization problem: Let* $\Omega = (0,1)$ *and denote as usual*

$$C = C(\bar{\Omega}) = \{f | f : [0,1] \to \mathbb{R} \text{ continuous}\}$$

and

$$L^2 = L^2(\Omega) = \left\{ f : \Omega \to \mathbb{R} \Big| \sqrt{\int (f(t))^2 dt} < \infty \right\}.$$

The L^2 *is—endorsed with the usual norm—a normed space. The topological dual space of* L^2 *is* L^2 *and the dual of* C *is the space* B *of signed Borel measures of* C*. Thus,* $\mu \in B$ *has value*

$$\mu(\phi) = \int \phi(x) d\mu(x)$$

for all $\phi \in C$*. We also need the Sobolev space*

$$H^2 = H^2(\Omega) = \{f : \Omega \to \mathbb{R} | D^2 f \in L^2\}$$

equipped with the semi norm

$$\|f\|_{H^2} = \sqrt{\int (D^2 f(t))^2 dt}.$$

This seminorm can be converted to a norm by considering equivalence classes. We define a function space $H_0^2 \subset H^2$ *which consists of all* H^2 *functions vanishing in* 0 *and* 1*, i.e.*

$$H_0^2 = H_0^2(\Omega) = \{f \in H^2 | f(0) = 0, f(1) = 0, Df(0) = 0, Df(1) = 0\}.$$

Obviously,

$$H_0^2 \subset H^2 \subset L^2 \subset C$$

after suitable identification. Note that the topological dual of H_0^2 *is the space* H^{-2} *consisting of functions*

$$f = f_0 + Df_1 + D^2 f_2$$

with $f_i \in L^2$*. All derivatives in this section are taken in the sense of distributions, see for example [Lions and Magenes (1972)] as a standard reference.*

The following theorem is the main result of this section and a generalization of Theorem 3.4 of [Hornung (1978)].

Theorem 3.1 *The minimization problem has a unique solution $z \in H^2$ which is characterized by*

$$D^4 z = \sum_{i=1}^{n} \lambda_i \delta_{x_i} - D\mu \tag{3.1}$$

where δ_x denotes the Dirac measure in x and μ is some Borel measure which satisfies

$$\int_0^1 D^1 z(x) d\mu(x) = 0. \tag{3.2}$$

The substance of the proof is a Kuhn–Tucker like functional Lagrangian principle from [Lempio (1972)] together with some convex analysis. First of all, we show that there is exactly one solution to the minimization problem. Then we pose another minimization problem for which a characterization can be stated using the techniques of [Hornung (1978)] which in turn is an appplication of the functional Lagrangian principle. Finally we are able to show that both solutions are equivalent.

We state the following facts.

Theorem 3.2 *(Prop. II 1.2 of [Ekeland and Temam (1976)]) Let us assume that $S : H_0^2 \to \mathbb{R}$ is strictly convex over a convex set $C_1 \subset H_0^2$, lower semi continuous, proper (i.e. it takes $-\infty$ nowhere and is not identically ∞) and S is coercive over C_1, i.e.*

$$\lim_{\|u\|_{H_0^2} \to \infty} S(u) = \infty, \quad u \in C_1$$

then the minimization problem

$$\inf_{u \in C_1} S(u)$$

has a unique solution.

Note that the minimization is restricted to the space H_0^2, which reflects the application of Remark 3.1. We have to check the properties of S and C_1 to use Theorem 3.2.

Lemma 3.1 *The smoothness functional $S : H_0^2 \to \mathbb{R}$ is strictly convex, proper, coercive, and Gateaux differentiable with*

$$S'_u = 2D^4 u.$$

The set $C_1 \subset H^2$ of functions which have the properties B, ∂, and M is convex and nonempty for a consistent choice of monotonicity intervals in M.

Proof of Lemma 3.1: First of all, we show Gateaux differentiability. For all $u \in H^2$ and $v \in H_0^2$, it is

$$\begin{aligned}\lim_{\lambda\to 0}\frac{S(u+\lambda v)-S(u)}{\lambda} &= \lim_{\lambda\to 0}\frac{1}{\lambda}\int (D^2u(t)+\lambda D^2v(t))^2-(D^2u(t))^2dt\\ &= 2\int D^2u(t)D^2v(t)dt\\ &= 2\langle D^2u, D^2v\rangle\\ &= 2\langle D^4u, v\rangle,\end{aligned}$$

with $\langle,\rangle$ denoting duality in the sense of distributions. Thus (see [Ekeland and Temam (1976)], p. 23),

$$S'(u) = 2D^4u.$$

For strict convexity of Gateaux differentiable functions over convex sets, it is sufficient to show that its differential is strictly monotone, i.e.

$$\langle u_1 - u_2, S'(u_1) - S'(u_2)\rangle > 0$$

for all $u_1 \neq u_2 \in C_1$ (see Proposition I 5.5 in [Ekeland and Temam (1976)]). For $u_1 \neq u_2 \in H_0^2$, we have

$$\begin{aligned}\langle u_1 - u_2, S'(u_1) - S'(u_2)\rangle &= 2\langle u_1-u_2, D^4(u_1-u_2)\rangle\\ &= 2\langle D^2(u_1-u_2), D^2(u_1-u_2)\rangle\\ &= 2\|u_1-u_2\|_{H_0^2}\\ &> 0\end{aligned}$$

Thus, S is strictly convex. Clearly, S is proper. It remains to show that S is coercive. Therefore we use another equivalence of convexity for Gateaux differentiable functions: S is convex if and only if

$$S(v) - S(u) \geq \langle S'(u), v-u\rangle,$$

for all $u, v \in H_0^2$ (see Proposition I 5.4 of [Ekeland and Temam (1976)]). We have convexity of S, thus for all $u \in H_0^2$

$$S(u/2) - S(0) \geq \langle S'(0), u/2\rangle = 0$$

and

$$S(u) - S(u/2) \geq \langle S'(u/2), u/2\rangle = 2\|u/2\|_{H_0^2},$$

and by addition,

$$S(u) - S(0) \geq \frac{1}{2}\|u\|_{H_0^2}.$$

Hence

$$\lim_{\|u\|_{H_0^2}\to\infty} S(u) = \infty.$$

By the consistency claim of the constraints, C_1 is nonempty. Convexity of C_1 is clearly fulfilled, completing the proof. □

Note that Theorem 3.2 and Lemma 3.1 together give existence and uniqueness of the solution of the minimization problem. But no characterization result can be achieved by these techniques. In order to do this we pose another minimization problem which will be referred to as the related problem.

The related problem: Let z be the unique solution of the minimization problem and

$$z(x_i) = z_i.$$

Given the (x_i, z_i), we want to minimize the functional S over the set $C_2 \subset H_0^2$ of all functions interpolating the (x_i, z_i) and possessing the right monotonicity M.

Theorem 3.3 *The related problem has a uniqe solution $\tilde{z}$ which has a characterization*

$$D^4\tilde{z} = \sum_{i=1}^{n} \lambda_i \delta_{x_i} - D\mu$$

for some Borel measure μ satisfying

$$\int_\Omega D\tilde{z}(x)d\mu(x) = 0.$$

The proof of this theorem is the generalization of the proof in [Hornung (1978)] to piecewise monotonicity. Let K be the convex cone of continuous functions which are candidates for $D\tilde{z}$,

$$K = \left\{ f \in C \middle| f(x) \geq 0, x \in I = \bigcup_j I_j,\ f(x) \leq 0, x \in A = \bigcup_j A_j \right\}.$$

Now we want the interpolation restriction to be satisfied: Let

$$\begin{aligned} G : H_0^2 &\to \mathbb{R}^n \\ G(u) &= (u(x_1) - z_1, \ldots, u(x_n) - z_n) = g(u) - z, \end{aligned}$$

with $g : H_0^2 \to \mathbb{R}^n$ the continuous and linear functional of point evaluation. Hence the restrictions of the related problem can be written as

$$u \in Q = \{u | u \in H_0^2,\ G(u) = 0,\ Du \in K\}.$$

Consistency of the montonic constraints in the minimization problem corresponds to the generalized Slater condition, which is assumed to be satisfied and requires the existence of a function $u \in H_0^2$ such that

$$G(u) = 0 \text{ and } Du \in \mathrm{int}(K), \tag{3.3}$$

$\mathrm{int}(K)$ denoting the topological interior of K. Here,

$$\mathrm{int}(K) = K \setminus \{0_{H_0^2}\}.$$

Now we present the Lagrangian principle tailored to our situation.

Lemma 3.2 *The unique solution $\tilde{z}$ of the related problem has a representation*

$$S'(\tilde{z}) = lg + \mu D \textit{ and } \mu(D\tilde{z}) = 0$$

with a linear form l over $\mathbb{R}^n$ and a nonnegative Borel measure μ,

$$\mu \in K^+ = \{m \in B | m(w) \geq 0, w \in K\}.$$

Proof of Lemma 3.2: This is a direct consequence of the much more general Theorem 3.1 of [Lempio (1972)], which is stated in this form for globally monotone functions in [Hornung (1978)], Theorem 2.2. □

We are now able to prove Theorem 3.3.

Proof of Theorem 3.3: We deduce the representation of 3.3 by straightforward calculations: First,

$$\mu H(\tilde{z}) = \mu(D\tilde{z}) = \int D\tilde{z}(t) d\mu(t) = 0.$$

Note that $g = (\delta_{x_1}, \ldots, \delta_{x_n})^t$ with $\delta_{x_i} \in H^{-2}$ denoting the delta distributions which are interpreted as continuous linear forms over H_0^2. With $l = (\lambda_1, \ldots, \lambda_n)$ and $\psi \in H_0^2$,

$$\begin{aligned}
\langle S'(\tilde{z}), \psi \rangle &= \sum_{i=1}^{n} \lambda_i \langle \delta_{x_i}, \psi \rangle + \mu(D\psi) \\
&= \sum_{i=1}^{n} \lambda_i \langle \delta_{x_i}, \psi \rangle - \langle D\mu, \psi \rangle.
\end{aligned}$$

Consequently,

$$\Rightarrow 2D^4\tilde{z} = \sum_{i=1}^{n} \lambda_i \delta_{x_i} - D\mu.$$

Substituting l by $l/2$ and μ by $\mu/2$ completes the proof. $\square$

Now we return to the proof of Theorem 3.1.

Proof of Theorem 3.1: Existence and uniqueness of the solution z of the minimization problem is given by Theorem 3.2. The function z is an appropiate candidate for the related problem by construction, hence $z \in C_2$ and

$$S(z) \geq S(\tilde{z}),$$

with $\tilde{z}$ denoting the solution of the related problem. On the other hand $\tilde{z}$ satisfies all constraints of the minimization problem, $\tilde{z} \in C_1$. Therefore

$$S(\tilde{z}) \geq S(z)$$

which implies

$$S(z) = S(\tilde{z}).$$

Now, by uniqueness of the solution, we have

$$z = \tilde{z}.$$

Thus, z inherits the representation of Theorem 3.3. $\square$

The characterization (3.1) of the spline solution z to the minimum roughness problem can be refined by a closer look at the properties of the Borel measure μ. Let for the moment z be isotone on $(0,1)$, i.e. $Dz \geq 0$. Since

$$\int Dz(t)d\mu(t) = 0,$$

μ may only have mass in points where Dz vanishes:

$$\mathrm{supp}(\mu) \subset \{t | Dz(t) = 0\}.$$

We can generalize this to piecewise monotonicity in the following way: Let

$$Dz(t) = \sum_{j=1}^{k_I} \psi_j(t) + \sum_{i=1}^{k_A} \phi_i(t),$$

with $\psi_j, \phi_i \in K$ and $\mathrm{supp}(\psi_j) \subset I_j, \mathrm{supp}(\phi_i) \subset A_i$. We have just split the solution z in isotonic pieces having positive (and continuous) derivative ψ_j and in antitonic pieces corresponding to negative ϕ_i. Thus

$$Dz = \sum_j \psi_j + \sum_i \phi_i,$$

where for each x there is at most one term which does not vanish. We have

$$\int Dz(t)d\mu(t) = \sum_{j=1}^{k_I} \int \psi_j(t)d\mu(t) + \sum_{i=1}^{k_A} \int \phi_i(t)d\mu(t) = 0. \tag{3.4}$$

Since by construction all ψ_j and ϕ_i are elements of K and $\mu \in K^+$, we have

$$\begin{aligned} \int \psi_j(t)d\mu(t) &\geq 0 \\ \int \phi_i(t)d\mu(t) &\geq 0. \end{aligned}$$

Hence all terms in (3.4) are nonnegative. In order to satisfy (3.4), every single term has to vanish:

$$\begin{aligned} \int \psi_j(t)d\mu(t) &= 0, \quad 1 \leq j \leq k_I \\ \int \phi_i(t)d\mu(t) &= 0, \quad 1 \leq i \leq k_A. \end{aligned}$$

Thus,

$$\operatorname{supp}(\mu) \subset \bigcup_j \{t \in I_j | \psi_j(t) = 0\} \cup \bigcup_i \{t \in A_i | \phi_i(t) = 0\} = \{t | Dz(t) = 0\}$$

as in the isotonic case.

The following corollary holds.

Corollary 3.1 *The unique solution z of Theorem 3.1 is a generalized cubic spline with possible knots in the x_i and possibly countable infinite additional knots in intervals where z is constant.*

It is possible to refine the characterization of the solution somewhat to give a sharp upper bound for the number of possible constancy intervals, see [Hornung (1978)], Theorem 4.2. Since it is of no practical relevance, we do not pursue it. The possibility of countable infinite additional knots cannot be seen directly by use of our techniques, we refer to [Wright and Wegman (1978)], where the following statement can be found.

Theorem 3.4 *If there exists $g \in H_0^2$ satisfying*

$$Dg(t) > 0, \quad t \in [0, 1]$$

and

$$l_i \leq g(x_i) \leq u_i, \quad 1 \leq i \leq n,$$

then the minimization problem has a unique solution which is a cubic spline. Knots are (potentially) located at the data points x_i and possibly at a countable infinite number of points outside.

One can easily think of situations where a countable number of knots is needed to 'sag the spline where it drops' as Wright and Wegman put it in [Wright and Wegman (1978)].

Proof of Theorem 3.4: This is a special case of Theorem 4.1 of [Wright and Wegman (1978)], with $m = 2$ and $F = G$. $\square$

In spite of the strong characterization of the solution given above it remains a very difficult problem to calculate it. One difficulty is to decide which of the x_i represent active constraints. The second difficult task arises afterwards when an interpolation method must be devised which preserves the correct monotonicity behaviour.

3.3 Constrained Spline Smoothing

We mention briefly the related problem of constrained smoothing splines. The constrained smoothing problem,

$$\lambda_n \int_0^1 (f^{(2)}(x))^2 d(x) + \sum_{i=0}^{n} (y_i - f(x_i))^2 = \min$$

over all f which satisfy some monotonicity constraints is easily shown to have a spline type solution. For ease of notation, we only consider the monotonic case which can be translated straightforward to the piecewise monotonic case. There is a vast amount of literature on this topic. We just comment a few assorted papers.

The first steps in monotone spline smoothing were made by [Villalobos and Wahba (1987)] and [Wright and Wegman (1978)]. The existence and uniqueness of the spline solution are given. The paper of [Wright and Wegman (1978)] additionally comes up with some basic statistical properties such as consistency of the estimator. The results are achieved by methods of convex analysis. In [Utreras (1983)], the same methods are used for the calculation of convergence rates.

Theorem 3.5 *Let $f \in H^2$ and*

$$y_i = f(x_i) + \varepsilon_i,$$

with uniform design x_i and ε_i an i.i.d. sequence of random variables with zero mean and variance σ^2. Further, let

$$|f|_k^2 = \int (f^{(k)}(x))^2 dx$$

for $k = 0, 1, 2$, and $z_{n,\lambda,C}$ the constrained smoothing spline with bandwidth λ satisfying the convex set of monotonicity constraints C. The optimal convergence rates for constrained cubic spline smoothing are attained for

$$\lambda_n = O(n^{-4/5}).$$

In this case we have

$$\mathbb{E}(|f - z_{n,\lambda_n,C}|_k^2) = O\left(n^{-\frac{4-2k}{5}}\right).$$

For a proof, see [Utreras (1983)].

Unfortunately, there is no algorithm for calculating this constrained spline. Utreras in [Utreras (1983)] proposes an 'exchange algorithm' for the calculation of the positions of the additional knots, but the idea is not made precise. Furthermore, it is stated that there is no existing technique to calculate the constrained spline. To our knowledge this situation has not changed.

A more recent paper is [Mammen et al. (1998)] and [Mammen and Thomas-Agnan (1998)]. The results of [Utreras (1983)] concerning the convergence rates are generalized to incorporate the case of nonuniform design. An application of the theory of empirical processes leads to some more insight in convergence behaviour. We do not want to go into detail here. The mathematically pleasing representation of the constrained spline as a projection of the unconstrained spline on a set of constraints looks more important. We present a special case of the more general Proposition 4.2 of [Mammen et al. (1998)], tailored to our situation.

Theorem 3.6 *The constrained smoothing spline z_c can be represented as a constrained minimisation over ordinary functions as:*

$$z_c(x) = \operatorname{argmin}_{f \in C \subset H^2} \frac{1}{n} \sum_{i=1}^{n} \|z(x_i) - f(x_i)\|^2,$$

where z is the unconstrained smoothing spline and $\|.\|$ is a Sobolev type norm,

$$\|f(x)\|^2 = \int f^2(x)\nu(dx) + \lambda \int (f^{(2)}(x))^2 dx,$$

and ν the empirical measure of the design points.

In [Mammen and Thomas-Agnan (1998)], the projection based approach is used for the proposal of an algorithm and for the examination of convergence of the estimators. But the calculation of the projection onto the constrained set of functions C (i.e. with respect to the Sobolev type norm) is still not directly computable. One has to discretise which leads to severe problems. In [Mammen et al. (1998)], an algorithm for local polynomial estimators is proposed which cannot help in our situation.

The related problem of monotonic interpolation with cubic splines will be discussed later on.

3.4 Discretization

In Section 3.1, we considered the minimum roughness problem in appropiate function spaces under constraints of the shape and bounds type which arose directly from the underlying data by use of the methods of Chapter 2. Now we want to make the problem computer tractable.

Since in statistical practice, one is mainly interested in point estimators of regression models, it is natural to discretize at the design points x_i. One might think of linear interpolation between these points if interested in continuous time models. For simplicity of notation, we restrict ourselves to a regular design

$$x_i = \frac{i}{n}, \quad 0 \leq i \leq n.$$

Now, the smoothness measure

$$S(f) = \int (f^{(2)}(t))^2 dt$$

must be expressed in terms of the $(x_i, f(x_i))$. A proposal of [Metzner (1997)] is the following:

- Approximate the second order derivative in S by the second order difference

$$f^{(2)}(x_i) \approx n^2 \left(f(x_{i+1}) - 2f(x_i) + f(x_{i-1})\right), \quad 1 \leq i \leq n-1.$$

  At the boundaries we set $f^{(2)}(0) = f^{(2)}(1) = 0$. In 3.1, we have seen a theoretical argument. A practical reason is to extrapolate an infinitesimal linear trend in areas of no data.

- Assume $f^{(2)}$ to be constant between the design points x_i and approximate the integral by a sum:

$$\int (f^{(2)}(t))^2 dt \approx \frac{1}{n} \sum_{i=1}^{n} (f^{(2)}(x_i))^2.$$

Thus, in summary, we have a discrete version of the smoothness functional S,

$$\begin{aligned} S_n^M : \mathbb{R}^{n+1} &\to \mathbb{R} \\ S_n^M(f) &= n^3 \left(f_0^2 + \sum_{i=1}^{n-1} (f_{i+1} - 2f_i + f_{i-1})^2 + f_n^2 \right), \end{aligned} \tag{3.5}$$

denoting

$$f = (f_0, \dots, f_n)^t = (f(x_0), \dots, f(x_n))^t.$$

The context clarifies if we want to deal with a function f or its empirical version $f \in \mathbb{R}^{n+1}$. The terms f_0^2 and f_n^2 in (3.5) are added to assure positive definiteness (which is not done in [Metzner (1997)]). This is not critical because the boundary values are prespecified or fixed by the bounds we use.

However, (3.5) can be written as a quadratic form in the following way:

$$S_n^M(f) = \frac{n^3}{2} f^t Q_{\mathrm{M}} f,$$

and

$$Q_{\mathrm{M}} = \begin{pmatrix} 4 & -4 & 2 & & & & \\ -4 & 10 & -8 & 2 & & & \\ 2 & -8 & 12 & -8 & 2 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 2 & -8 & 12 & -8 & 2 \\ & & & 2 & -8 & 10 & -4 \\ & & & & 2 & -4 & 4 \end{pmatrix}.$$

Note that the value $S_n^M(g)$ of a straight line g is $n^3(f_0^2 + f_n^2)$ due to the positive definiteness correction. This is the smallest value achievable.

Lemma 3.3 *Q_{M} is positive definite and thus S_n^M strictly convex. Hence the constraint minimum problem:* Minimize S_n^M over all vectors f satisfying the shape constraints and bounds *has exactly one solution which can be calculated by quadratic programming.*

Proof of Lemma 3.3: First we note that the set of vectors which satisfy the geometric constraints and the bound constraints is a convex set. Let $y \in \mathbb{R}^{n+1}$, then

$$\frac{1}{2} y^t Q_{\mathrm{M}} y = y_0^2 + \sum (y_{i+1} - 2y_i + y_{i-1})^2 + y_n^2 \geq 0.$$

Let now $y \in \mathbb{R}^{n+1}$ with

$$y^t Q_{\mathrm{M}} y = 0.$$

Since each term is nonnegative, all single terms have to vanish in order to make the whole quadratic form disappear, i.e.

$$\begin{aligned} y_0 &= 0 \\ y_{i+1} - 2y_i + y_{i-1} &= 0, \quad 1 \le i \le n-1 \\ y_n &= 0. \end{aligned} \tag{3.6}$$

The recurrence term in (3.6) can be represented explicitly by standard arguments: Second order linear homogenuous difference equations satisfy

$$y_i = \lambda_1 z_1^n + \lambda_2 z_2^n$$

with z_i the roots of a polynomial coming from (3.6) and the λ_i from the boundary values. In this case, we have

$$y_i = 0 \cdot 1^n + 0 \cdot 0^n = 0$$

for $0 \le i \le n$, thus $y = 0$ and Q_M positive definite. Thus, for $x, y \in \mathbb{R}^{n+1}$ and $t \in (0,1)$,

$$(1-t)S_n^M(x) + tS_n^M(y) - S_n^M((1-t)x + ty) = \frac{t(1-t)}{2} S_n^M(x-y) > 0.$$

Hence S_n^M is strictly convex and has to be minimized over a convex set. This is a sort of problem which is known to have a unique solution. Clearly, this solution can be calculated by quadratic programming since the set of constraints has a suitable linear inequality structure in $\mathbb{R}^{n+1}$, see Example 3.1. $\square$

At first glance the discretization of [Metzner (1997)] is somewhat unsatisfying. We expect the solution of our minimization problem to be a piecewise cubic function because it seems plausible that the discrete solution inherits the properties of the solution of the continuous time minimization problem of Section 3.1. But functions of this type do not belong to the class of functions which are approximated correctly by the Metzner kind of discretization. This systematic approximation error occurs in the integral approximation, whereas the approximation of the second order derivative is exact for cubic functions which have no knots in the approximation interval (x_{i-1}, x_{i+1}).

We propose approximating the integral by

$$\int_{x_i}^{x_{i+1}} (f^{(2)}(t))^2 dt \approx \frac{1}{3n} \left(f^{(2)}(x_i)^2 + f^{(2)}(x_{i+1})^2 + f^{(2)}(x_{i+1}) f^{(2)}(x_i) \right). \tag{3.7}$$

Summing up, we have

$$S_n^L(f) = \frac{1}{3n} \left(f_0^2 + \sum_{i=0}^{n-1} \left(f^{(2)}(x_i)^2 + f^{(2)}(x_{i+1})^2 + f^{(2)}(x_{i+1}) f^{(2)}(x_i) \right) + f_n^2 \right)$$

The boundary values are again set to

$$f^{(2)}(x_0) = f^{(2)}(x_n) = 0$$

and $f(x_0)^2 + f(x_n)^2$ is added to assure positive definiteness. By use of the standard approximation of the second order derivative, we have

$$S_n^L(f) = \frac{n^3}{3} f^t Q_L f, \tag{3.8}$$

where Q_L is a symmetrical band matrix given by

$$Q_L = \begin{pmatrix} 6 & -7 & 2 & 1 & 0 & \cdots & & & \\ -7 & 16 & -10 & 0 & 1 & 0 & \cdots & & \\ 2 & -10 & 16 & -9 & 0 & 1 & 0 & \cdots & \\ 1 & 0 & -9 & 16 & -9 & 0 & 1 & 0 & \cdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \cdots & 0 & 1 & 0 & -9 & 16 & -9 & 0 & 1 \\ & & & 1 & 0 & -9 & 16 & -10 & 2 \\ & & & & 1 & 0 & -10 & 16 & -7 \\ & & & & & 1 & 2 & -7 & 6 \end{pmatrix}$$

As already noted for S_n^M, the smallest attainable value for S_n^L is $n^3/3(f_0^2 + f_n^2)$ which is achieved by the straight line through (x_0, f_0) and (x_n, f_n).

The following lemma says that no further improvement in approximation can be made on the design grid. Some exact approximation properties are given. We can restrict ouselves to piecewise cubic polynomials because the cubic splines are well known to be the smoothest functions interpolating certain data points.

Lemma 3.4

1. *The integral approximation (3.7) is exact for natural cubic splines with possible knots in the* x_i.

2. *There is no integral approximation by discretization at the design points which delivers exact approximation for natural cubics with knots outside the* x_i.

3. *The derivative approximation*

$$f^{(2)}(x_i) = n^2(f(x_{i+1}) - 2f(x_i) + f(x_{i-1})), \quad 1 \le i \le n-1$$

is exact for cubic splines having no knot in (x_{i-1}, x_{i+1}).

4. *There is no approximation based on discretization at the design points* x_i *which replicates exactly the second order derivative of cubics with knots in* (x_{i-1}, x_{i+1}).

Proof of Lemma 3.4:

1. Without loss of generality we can restrict ourselves to just one of the design intervals (x_i, x_{i+1}). We calculate the curvature of a cubic $c(t)$ living on this interval,

$$\int_{x_i}^{x_{i+1}} (c^{(2)}(t))^2 dt.$$

$c^{(2)}$ is a straight line, which can be expressed by any two points, e.g. $(x_i, c^{(2)}(x_i))$ and $(x_{i+1}, c^{(2)}(x_{i+1}))$. Denote

$$m_i = c^{(2)}(x_i), \quad m_{i+1} = c^{(2)}(x_{i+1}),$$

hence

$$\begin{aligned}
\int_{x_i}^{x_{i+1}} (c^{(2)}(t))^2 dt &= \int_{x_i}^{x_{i+1}} (n(m_{i+1} - m_i)(t - x_i) + m_i)^2 dt \\
&= \left[\frac{1}{3(m_{i+1} - m_i)} (n(m_{i+1} - m_i)(t - x_i) + m_i)^3 \right]_{x_i}^{x_{i+1}} \\
&= \frac{1}{3n}(m_{i+1}^2 + m_i^2 + m_i m_{i+1})
\end{aligned}$$

by standard calculations.

2. Clearly, if there is relevant information in the interval which does not influence the approximation, then there is no way of doing it correctly. We formalize this by showing that an arbitrary large amount of roughness can be obtained by adding just one additional knot at $(x_i + x_{i+1})/2$. Without additional knot, we have

$$s_1 = \int_{x_i}^{x_{i+1}} (c^{(2)}(t))^2 dt = \frac{1}{3n}(m_i^2 + m_{i+1}^2 + m_i m_{i+1}),$$

the presence of the additional knot yields another function $\tilde{c}$ and so

$$s_2 = \int_{x_i}^{x_{i+1}} (\tilde{c}^{(2)}(t))^2 dt = \frac{1}{6n}(m_i^2 + m^2 + m_i m + m_{i+1}^2 + m^2 + m m_{i+1}),$$

m denoting the second order derivative of $\tilde{c}$ in $(x_i + x_{i+1})/2$. Since the right hand side of the above equation is a quadratic function in m we only have to consider the discriminant

$$s_2 - c_n \left(\frac{m_{i+1} - m_i}{2}\right)^2,$$

which is positive for sufficiently large s_2. Thus, an arbitrary amount of smoothness can be introduced by additional information. Hence there is no approximation at the design grid allowing exact approximation.

3. It is sufficient to concentrate our interest on the uniquely determined cubic function c defined on (x_{i-1}, x_{i+1}) with

$$\begin{aligned} c(x_{i-1}) &= c_{i-1} \\ c(x_{i+1}) &= c_{i+1} \\ c^{(2)}(x_{i-1}) &= m_{i-1} \\ c^{(2)}(x_{i+1}) &= m_{i+1}. \end{aligned}$$

Since $c^{(2)}$ is a straight line and we have restricted to equidistant design,

$$c^{(2)}(x_i) = \frac{m_{i-1} + m_{i+1}}{2}.$$

We have to show now that

$$n^2(c(x_{i+1}) - 2c(x_i) + c(x_{i-1})) = \frac{m_{i-1} + m_{i+1}}{2}$$

holds. From [Stoer and Burlisch (1990)], p.86, or by standard analytical arguments we deduce the representation for c:

$$\begin{aligned} c(t) &= m_{i-1}\frac{(x_{i+1}-t)^3}{6(x_{i+1}-x_{i-1})} + m_{i+1}\frac{(t-x_{i-1})^3}{6(x_{i+1}-x_{i-1})} \\ &+ \left(\frac{c_{i+1}-c_{i-1}}{x_{i+1}-x_{i-1}} - \frac{x_{i+1}-x_{i-1}}{6}(m_{i+1}-m_{i-1})\right)(t-x_{i-1}) \\ &+ c_{i-1} - m_{i-1}\frac{(x_{i+1}-x_{i-1})^2}{6} \end{aligned}$$

and thus,

$$c(x_i) = \frac{c_{i-1}+c_{i+1}}{2} - \frac{m_{i-1}+m_{i+1}}{4n^2},$$

which is equivalent to

$$n^2(c(x_{i+1}) - 2c(x_i) + c(x_{i-1})) = \frac{m_{i-1}+m_{i+1}}{2}.$$

Note that this technique can be applied to derive non–equidistant–design approximations for integral and second order derivative.

4. By analogy to what we said in 2., any value for the second order derivative is possible by a suitable adjustment of just one knot in (x_{i-1}, x_{i+1}) which does not affect the approximation. So the approximation error can be made arbitrarily large here as well.

□

As for Q_M, we have the following lemma.

Lemma 3.5 *Q_L is positive definite and thus S_n^L strictly convex. Hence the constraint minimum problem:* Minimize S_n over all vectors f satisfying the shape constraints and bounds *has exactly one solution which can be calculated by quadratic programming.*

For an example of quadratic programming with Q_L, see Examples 3.1 and 3.2.

Proof of Lemma 3.5: We only have to show positive definiteness. Let $y \in \mathbb{R}^{n+1}$, then

$$y^t Q_L y = \frac{3}{n^3}\left(y_0^2 + \sum_{i=0}^{n-1}(z_i^2 + z_{i+1}^2 + z_i z_{i+1}) + y_n^2\right) \geq 0$$

since every single summand is nonnegative, denoting

$$z_i = \begin{cases} 0 & i \in \{0, n\} \\ y_{i+1} - 2y_i + y_{i-1} & 1 \leq i \leq n-1 \end{cases}$$

Now, let $y \in \mathbb{R}^{n+1}$ satisfy

$$y^t Q_L y = 0,$$

which is equivalent to

$$\begin{aligned} y_0 &= 0 \\ z_i^2 + z_{i+1}^2 + z_i z_{i+1} &= 0, \quad 0 \leq i \leq n-1 \\ y_n &= 0. \end{aligned}$$

We conclude by induction that $z_i = 0$ for all $0 \leq i \leq n$. Hence

$$y_{i+1} - 2y_i + y_{i-1} = 0, \quad 1 \leq i \leq n-1,$$

which only allows for a constant solution (see proof of Lemma 3.3),

$$y_i = \lambda.$$

Because of $y_0 = y_n = 0$, we have $\lambda = 0$ and thus positive definiteness of Q_L. □

In the next section, it will be of some importance that the row sums of Q_M and Q_L all (but the first and last due to edge effects and to our positive definiteness correction) vanish. This property is not only of technical importance, but does reflect that straight lines have a vanishing curvature (up to an additive constant of $c(f_0^2 + f_n^2)$) in our discretization which is reasonable. We formulate it as a lemma for future reference.

Lemma 3.6 *Let $Q \in \{Q_L, Q_M\}$ and $1 \leq i \leq n-1$. Then,*

$$\sum_{j=0}^{n} Q_{ij} = 0.$$

Proof of Lemma 3.6: By a close look at the matrices. □

We examine now how quadratic programming can be used to solve the problem.

Example 3.1 *We consider a simple example to demonstrate how quadratic programming can be introduced to our situation. There are several quadratic programming tools available on the internet. For the following examples, we used code from Berwin Turlach [Turlach (1998)] which is designed to solve problems of the following kind: Minimize*

$$\frac{1}{2}x^t Q x - d^t x$$

under the constraints

$$A_1^t x = b_1, \quad A_2^t x \geq b_2.$$

The constraints are fed into a matrix

$$A = (A_1 A_2)$$

and a vector

$$b = (b_1^t, b_2^t)$$

which can be generated from bounds by `prepare.constraints`*, an interface which we designed (see the appendix for the code). Result of this interface is the constraints matrix relation*

$$\underbrace{\begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & 1 & \\ \hline -1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & -1 \\ \hline -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix}}_{=A^t} x \geq \underbrace{\begin{pmatrix} l_1 \\ \vdots \\ \vdots \\ l_n \\ -u_1 \\ \vdots \\ \vdots \\ -u_n \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{=b}. \tag{3.9}$$

If we look at examples where isotonicity and antitonicity are present, we have to multiply the rows corresponding to antitonic intervals in the last block of the matrix by -1*. Since we look at isotonic problems here, details are left out. Anyone interested in computional details is referred to the appendix. Our examples consist of two sets of bounds each of eight design points,*

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
l_i^1	-10	-1	-1	-1	-1	-1	1	10
u_i^1	-10	-1	1	1	1	1	1	10
l_i^2	-3	-1	-1	-1	-1	-1	1	2
u_i^2	-3	-1	1	1	1	1	1	2

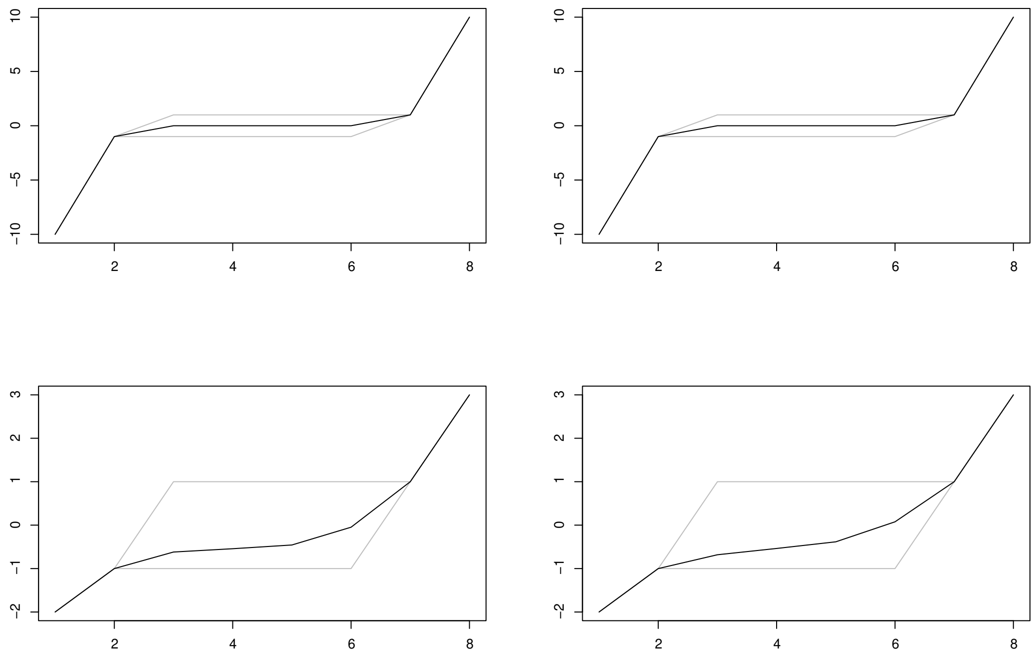

Figure 3.1: The first row shows the first set of bounds from Example 3.1 and the minimum of S_n^M (left) and S_n^L (right). The second row shows the second set of bounds.

The results of these quadratic programs are displayed in Figure 3.1. Note that there is no considerable difference between the two discretization matrices. This meets our experience with most data sets. For this reason and from now on we only apply Q_M which makes our pictures comparable to those of [Metzner (1997)]. The first set of bounds forces a constancy interval at 0 which is reproduced in both panels, whereas the second test bounds are less symmetrical and give strictly increasing curves which meet our optical idea of smoothness very well.

Example 3.2 *We apply our* qp *program to some data. This will be the only time that this is done in this thesis for the following reasons:*

- *Quadratic programming is very memory consuming. In the version we used there is a need of*

$$8 \times \left(2n + \frac{r(r+5)}{2} + 2q + 1\right)$$

bytes for double precision variables, denoting q the number of constraints and $r = \min(n, q)$. Since in our situation,

$$\begin{aligned} q &= 3n - 1 \\ r &= n, \end{aligned}$$

we have to allocate the enormous number of

$$8 \times \left(\frac{n^2 + 21n}{2} - 2 \right)$$

bytes for the calculations (see [Turlach (1998)]). For general quadratic programs it is not possible to improve this substantially, see [Goldfarb (1986)]. Possibly the concise structure of Q and the constraint matrix A which are band matrices may allow for a reduction in the memory demands.

- *Quadratic programming is very time consuming. Each single check of all $3n-1$ constraints takes some order n^2 calculations (if the band structure of A is not used). There are modern variants of quadratic programming which are of complexity $O(n^3L)$, [Goldfarb and Liu (1993)]. The code [Turlach (1998)] we used is actually an implementation of [Goldfarb and Idnani (1982)], where the quadratic program is solved by constructing a sequence of linear programs which are solved by the simplex algorithm. Since it is well known that simplex is at worst of exponential complexity (but of polynomial expected complexity), we have at worst exponential number of calculations to be done, at best some (higher order) polynomial complexity. The more modern approaches of [Goldfarb and Liu (1993)] use worst case polynomial order linear program solvers. In practice, the exponential worst case behaviour of the simplex algorithm never occurs and the older quadratic program solvers are not really outperformed by the newer ones. For our purpose even n^3 is still too much. Probably the complexity can be improved by further optimization due to the special structure of the quadratic program at hand. It seems clear that we cannot avoid the exponential worst case behaviour, but the band structure of the matrices is certainly able to reduce the degree of the polynomial of (expected) complexity by one because the complexity of the matrix product is decreased by one in degree and each iteration has complexity of order n instead of n^2. But even then we are confronted with at least an order n^2 algorithm.*

In our example, we look at the Härdle data B.2 which consist 256 observations, i.e. we produce a 256×767 matrix of constraints (1.5 Mbyte) and will use roundabout 35.000 additional double precision cells for the calculations which is a total amount of more than 1.7 MByte. This is too much for a data set of this size. The calculation time for the preparation of the constraints is about 20 sec. CPU time, the computation of the minimum depends heavily on the constraints. Here, for the Härdle data, we have

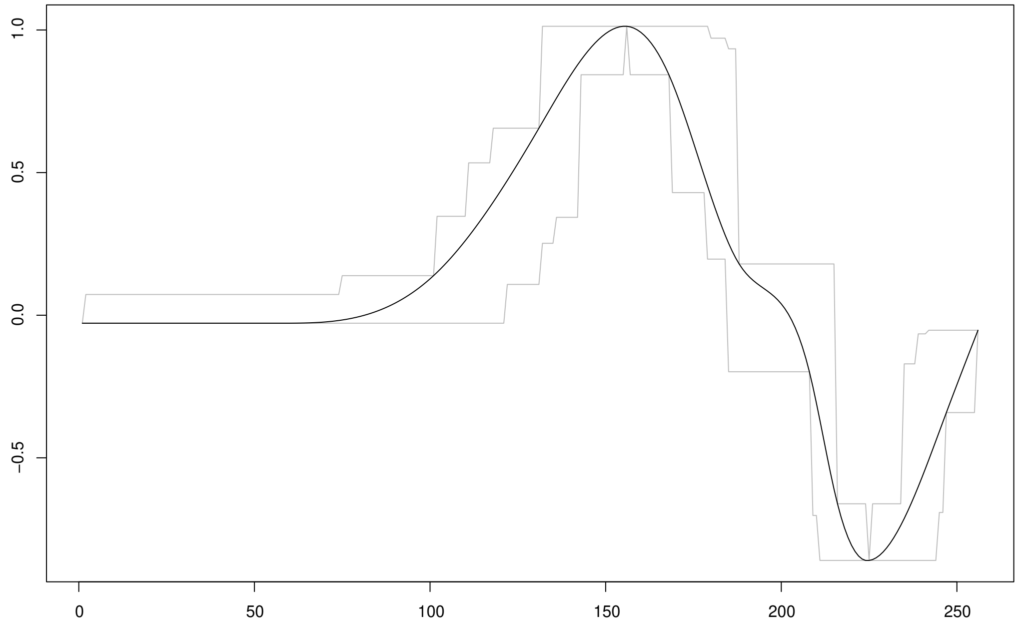

Figure 3.2: The smoothest adequate curve for the Härdle data using the string feature, calculated by quadratic programming

more than 5 sec. CPU time, but we tried other data sets involving more active constraints where the computation time can go up to more than 20 sec. CPU time. Since we expect at least quadratic increase with n (this is quite overoptimistic!), quadratic programming is too expensive both in time and memory usage. If we think of the artificial data sets B.3 drawn from the Donoho and Johnstone test signals ($n = 2048$), the allocated memory would exceed reasonable quantities by far (more than 120 Mbyte), the computation time cannot be expected to be less than half an hour. Our attempts to calculate the minimum for the Donoho et al. data were not successful because we ran short of memory.

Again, there is actually no difference between the minimum of S_n^M and S_n^L, the maximum absolute difference of both is 0.0018 which is neglible. The result displayed in Figure 3.2 is graphically very pleasing, containing no feature which cannot be seen in the data. Even the small dip between the local extremes seems to follow a tendency in the data. Actually, since this data set is artificial, we know that there is such an dip in the true underlying signal. Note that this feature can also be seen using the run method for the construction of the bounds.

We have seen now that the minimum roughness approach retains the features of the data and produces optically satisfactory models of the data. The only problem left is that of the computational complexity of the algorithm: the direct solution by quadratic programming is too computer intensive. We tackle the problem of memory consumption in the next section, whereas the computation speed remains crucial at first.

3.5 QSOR

We have already expressed the desire for fast and sparse algorithms which calculate the solution of the smoothness problem. [Metzner (1997)] proposed an iterative method to calculate the solution subject to the discrete versions of the shape constraints: Metzner's algorithm, which he called the QSOR algorithm, is an adaptation of the method of successive over relaxation (SOR) (see [Stoer and Burlisch (1990)]) but where at each iteration step the shape and bound constraints are imposed.

A slight modification of the algorithm can be shown to converge; some gaps in the proof in [Metzner (1997)] are closed here. We give an example for convergence to a nonoptimal solution without the modification.

First, we give a short summary of the modified QSOR method. We restrict ourselves to the isotonic case, the piecewise monotonic generalization is not so diffcult but would complicate the notation without need.

Definition 3.2 *Let $Q \in \{Q_M, Q_L\}, y \in \mathbb{R}^{n+1}$ and the relaxation parameter $\omega \in (0, 2)$. The following will be called a forward QSOR step with parameter $0 < \omega < 2$:*

- *Start with $i = 1$ (y_0, y_n are fixed by the bounds).*
- *Set*
$$\tilde{y}_i = y_i - \frac{\omega}{Q_{ii}} (Qy)_i \, .$$
Check if $\tilde{y}_i$ holds the constraints: force $\hat{y}_i$ to respect the bounds by setting
$$\hat{y}_i = \min\{\max\{l_i, \tilde{y}_i\}, u_i\},$$
and to satisfy the prescribed left and right monotonicity (i.e. isotonicity here) by
$$\bar{y}_i = \min\{\max\{y_{i-1}, \hat{y}_i\} y_{i+1}\}.$$
If the isotonicity constraint is active, mark the index i. Overwrite
$$y_i \leftarrow \bar{y}_i.$$

- *Set $i = i + 1$, iterate.*
- *If $i = n - 1$, stop.*
- *Correct the height of all constancy intervals, i.e. intervals of active isotonicity constraints. Let thus $[x_\nu, x_{\nu+k}]$ be a constancy interval. Then we set*

$$y'_j = y_j - \frac{\omega}{\sum\limits_{i,j=\nu}^{\nu+k} Q_{ij}} \left(\sum_{i=\nu}^{\nu+k} (Qy)_i \right) \tag{3.10}$$

for all $\nu \leq j \leq \nu + k$, but this time simultaneously without updating y_j before y_{j+1} is calculated. One can use a value other than ω in this situation, but for the sake of simplicity we restrict ourselves to just one relaxation parameter. We have to check the constraints again, obviously it is sufficient to check whether the new height y'_ν lies between $y_{\nu-1}$ and $y_{\nu+k+1}$,

$$y_j = \min\{\max\{y_{\nu-1}, y'_i\} y_{\nu+k+1}\}.$$

A backward QSOR step is defined similarly. A forward and a backward QSOR step together is called a QSOR step.

We refer to SSOR (stepwise successive over– relaxation) method which is the same procedure of combining a forward and backward SOR step, see e.g. [Stoer and Burlisch (1990)] for a discussion. For convergence, it is sufficient to restrict attention to only one direction (i.e. forward QSOR) as in the case of the SSOR algorithm, see e.g. [Stoer and Burlisch (1990)].

Remark 3.2

- *The isotonicity check is a slight modification of the Metzner algorithm in [Metzner (1997)], p. 113, where monotonicity is only tested on the left–hand side which does not produce permissible vectors y in every iteration step. This leads to severe problems in the case of active isotonicity constraints. Indeed, the Metzner variant does not converge to the restricted minimum in some situations, see Example 3.3.*
- *The height correction step for active constancy intervals is another modification of the original Metzner procedure. Without it convergence is not guaranteed, see Example 3.3.*
- *The modified QSOR algorithm will be shown to converge later on.*

- *The double sum which has to be calculated in the height correction step (if the boundary of the data is not touched) needs to be calculated only once from the smoothness matrices and has not to be calculated in each iteration, see the following Lemma 3.7.*

Lemma 3.7 *Let $[x_\nu, x_{\nu+k}]$ be a constancy interval of y of length $k \geq 1$ which is not at the boundary of the data, $2 \leq \nu \leq n-2$. For $Q = Q_M$, we have*

$$\sum_{j=\nu}^{\nu+k} (Qy)_j = 2y_{\nu-2} - 6y_{\nu-1} + 8y_\nu - 6y_{\nu+k+1} + 2y_{\nu+k+2}$$

and

$$\sum_{i,j=\nu}^{\nu+k} Q_{ij} = 8.$$

For $Q = Q_L$, $k \geq 2$ and $3 \leq \nu \leq n-3$, we have similar results:

$$\begin{aligned} \sum_{j=\nu}^{\nu+k} (Qy)_j &= y_{\nu-3} + y_{\nu-2} - 8y_{\nu-1} + 12y_\nu - 8y_{\nu+k+1} + y_{\nu+k+2} + y_{\nu+k+3} \\ \sum_{i,j=\nu}^{\nu+k} Q_{ij} &= 12. \end{aligned}$$

For $k = 1$, we have

$$\sum_{j=\nu}^{\nu+k} (Qy)_j = y_{\nu-3} + y_{\nu-2} - 9y_{\nu-1} + 14y_\nu - 9y_{\nu+k+1} + y_{\nu+k+2} + y_{\nu+k+3}$$

and

$$\sum_{i,j=\nu}^{\nu+k} Q_{ij} = 14.$$

Proof of Lemma 3.7: We only calculate the sums for Q_M, the proof of the results for Q_L follows the same lines. We will need two properties of Q_M: symmetry and vanishing row sums (see Lemma 3.6). Fix $k = 1$, then

$$\sum_{i,j=\nu}^{\nu+1} Q_{ij} = 2 \cdot 12 - 2 \cdot 8 = 8.$$

If we set $k \leftarrow k+1$, an entire row of Q_M will be added which sums to 0:

$$\begin{array}{cc|c} 12 & -8 & \mathbf{2} \\ -8 & 12 & \mathbf{-8} \\ \hline \mathbf{2} & \mathbf{-8} & \mathbf{12} \end{array}$$

The other equation follows by simple arithmetic:

$$\begin{aligned}\sum_{j=\nu}^{\nu+k}(Qy)_j &= 2y_{\nu-2}-6y_{\nu-1}+\left(\sum_{i,j=\nu}^{\nu+1}Q_{ij}\right)y_\nu-6y_{\nu+k+1}+2y_{\nu+k+2}\\ &= 2y_{\nu-2}-6y_{\nu-1}+8y_\nu-6y_{\nu+k+1}+2y_{\nu+k+2}.\end{aligned}$$

This completes the proof. □

Now we demonstrate the previously mentioned necessity of the modifications of QSOR by a small example.

Example 3.3 *We only regard the case $Q = Q_M$, but a similar example can easily be constructed for Q_L. We consider*

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
l_i	*-10*	*-1*	*-1*	*-1*	*-1*	*-1*	*1*	*10*
u_i	*-10*	*-1*	*1*	*1*	*1*	*1*	*1*	*10*

which is apparently the first set of points of Example 3.1. In this case, the mean of the bounds

$$\frac{l_i+u_i}{2}$$

is already the optimal solution which we were able to calculate with the help of quadratic programming. We investigate whether the Metzner–type QSOR and our modified version alter this optimal vector (only forward–QSOR is considered).

- y_0, y_1, y_6, y_7 *cannot be changed due to the bounds. What about y_2?*

$$\begin{aligned}\tilde{y}_2 &= y_2-\frac{\omega}{Q_{22}}(Qy)_i\\ &= 0-\frac{\omega}{12}(2\cdot(-10)-8\cdot(-1)+12\cdot 0-8\cdot 0+2\cdot 0)\\ &= \omega\end{aligned}$$

Since $\omega > 0$, isotonicity to the right ($y_3 = 0$) is no longer valid,

$$\hat{y}_2 = 0.$$

y_3, y_4 *cannot be changed due to the isotonicity constraint, and*

$$\begin{aligned}\tilde{y}_5 &= y_5-\frac{\omega}{12}(2\cdot 0-8\cdot 0+12\cdot 0-8\cdot 1+2\cdot 10)\\ &= -\omega,\end{aligned}$$

thus isotonicity to the left is active,

$$\hat{y}_5 = y_4 = 0.$$

Thus, the modified QSOR method does not alter this vector, i.e. we have reached a limit. We have

$$S_n^M(y) = 330n^3$$

which is the minimum attainable. Thus, the introduction of the right and left isotonicity check does not allow QSOR to run out of the minimum.

- *Now we take a look at Metzner–type QSOR. As above,* y_0, y_1, y_6, y_7 *are fixed because of the bounds. Again,*

  $$\tilde{y}_2 = \omega,$$

  but this time right isotonicity will not be considered as constraint. Thus with respect to the upper bound, we set

  $$y_2 \leftarrow \min(\omega, 1).$$

  For simplicity we set $\omega = 1$*, hence* $y_2 = 1$*. Therefore for the following points, left isotonicity becomes active. Thus we set*

  $$y_3, y_4, y_5 \leftarrow 1.$$

  What happens in the next forward–QSOR step?

  $$\begin{aligned}\tilde{y}_2 &= 1 - \frac{1}{12}(2 \cdot (-10) - 8 \cdot (-1) + 12 \cdot 1 - 8 \cdot 1 + 2 \cdot 1) \\ &= \frac{3}{2}\end{aligned}$$

  This does not satisfy the upper bounds,

  $$y_2 \leftarrow 1$$

  and the following y_3, y_4, y_5 *are forced to 1 again. Thus, Metzner–type QSOR has converged to a suboptimal limit, attaining a higher function value of*

  $$S_n^M(y) = 349n^3.$$

- *We consider the modified QSOR version with another start vector, say the result of Metzner–type QSOR of the last point,*

  $$y_i = u_i.$$

It is easy to see that no single y_i is changed following the same lines as in the last point. The only notable difference is the postcorrection of the height of the constancy interval $[x_2, x_6]$: the corrected height for $2 \leq j \leq 6$ is

$$\begin{aligned} y'_j &= y_j - \frac{\omega}{8}(2y_0 - 6y_1 + 8y_j - 6y_6 + 2y_7) \\ &= y_j - \omega y_j < y_j, \end{aligned}$$

which is consistent with the constraints for any choice of $\omega \in (0, 2)$,

$$y_j \leftarrow y'_j.$$

Thus, the modified QSOR does not leave the suboptimal vector unaltered. For $\omega = 1$, the optimum is attained immediately, for other allowed values of omega, we have

$$y'_j = (1 - \omega)^l y_j \to 0 \quad (l \to \infty)$$

after l iterations. Again, this is consistent with the constraints in every single iteration step. Thus, for all $\omega \in (0, 2)$, the modified QSOR algorithm converges to the minimum.

We have seen that situations may occur in which QSOR of Metzner-type will not do very well. The convergence of the modified QSOR can be shown which is the main result of this section. The proof could earn some interest because it reveals the construction method of the modifications which we made in the algorithm.

Theorem 3.7 *The QSOR algorithm converges. Its limit is the unique minimum of S_n over the constrained set C.*

Proof of Theorem 3.7: Again, we consider only isotonicity. The proof can easily be extended for piecewise monotonicity. We prove that a QSOR step reduces the functional value unless we have arrived in the global minimum. First, given a vector $y \in \mathbb{R}^{n+1}$, we change the value of $y_i,\ 0 \leq i \leq n$ according to the modified QSOR method. The vector which arises from y by substituting

$$y_i \leftarrow y_i - \frac{\omega}{Q_{ii}}(Qy)_i$$

followed by a constraint check is called $\vec{y}$. Let $Q \in \{Q_M; Q_L\}$ (actually, we need less: symmetry, positive definiteness, vanishing row sums, and positivity of diagonal elements). The proof consists of several steps. We recall the notation of Definition 3.2.

1. First, we assume that there are no constraints which can be invalidated by a change in y_i. Then, denoting

$$\delta = (0, \dots, 0, \frac{\omega}{Q_{ii}}(Qy)_i, 0, \dots, 0)^t,$$

we have

$$\begin{aligned} y^t Q y - \vec{y}^{\,t} Q \vec{y} &= y^t Q y - (y-\delta)^t Q (y-\delta) \\ &= 2y^t Q (y - (0, \dots, 0, \frac{\omega (Qy)_i}{Q_{ii}}, 0, \dots, 0)) - Q_{ii}\left(\frac{\omega (Qy)_i}{Q_{ii}}\right)^2 \\ &= \frac{(2\omega - \omega^2)(Qy)_i^2}{Q_{ii}} \geq 0, \end{aligned}$$

since $\omega \in (0, 2)$. Equality is attained only if $(Qy)_i = 0$ which coincides with local optimality of the value to be corrected. Thus, if no constraint is present, we have reduced the function value by correcting y in the i-th component or kept it because it was locally optimal in the first place.

2. What happens if a boundary is reached? Without loss of generality, we might assume that the lower bound is attained, i.e. $\hat{y}_i = l_i$ and $(Qy)_i \neq 0$. Hence

$$\tilde{y}_i < l_i \leq y_i.$$

and

$$y_i - \frac{\omega}{Q_{ii}}(Qy)_i \leq y_i,$$

and so

$$(Qy)_i > 0.$$

Let now denote

$$\hat{\omega} = (y_i - l_i)\frac{Q_{ii}}{(Qy)_i}.$$

We write

$$\vec{y}_i = l_i = y_i - \frac{\hat{\omega}}{Q_{ii}}(Qy)_i,$$

yielding

$$y^t Q y - \vec{y}^{\,t} Q \vec{y} = \frac{(2\hat{\omega} - \hat{\omega}^2)(Qy)_i^2}{Q_{ii}} \tag{3.11}$$

again, but now with $\hat{\omega}$ instead of ω. The right hand side of (3.11) is strictly greater than 0 iff $\hat{\omega} \in (0, 2)$. Clearly, $\hat{\omega} \geq 0$ by definition. Suppose now

$$\hat{\omega} \geq 2.$$

Then

$$y_i - l_i \geq \frac{2(Qy)_i}{Q_{ii}} > \frac{\omega(Qy)_i}{Q_{ii}},$$

thus

$$y_i - \frac{\omega(Qy)_i}{Q_{ii}} > l_i$$

which is contrary to our assumption of the lower bound to be touched. Thus, the right hand side of (3.11) is nonnegative and strictly positive when

$$\hat{\omega} \neq 0,$$

which coincides with the case of the lower bound having been an active constraint before.

3. Let us assume now that there is no constraint active apart from isotonicity. Clearly,

$$(Qy)_i \neq 0.$$

Of the two possibilities

$$\hat{y}_i < y_{i-1} \text{ and } \hat{y}_i > y_{i+1}$$

we can restrict attention to only one alternative without loss of generality, say

$$\tilde{y}_i < y_{i-1}.$$

which means

$$\bar{y}_i = y_{i-1}.$$

First we note that from

$$\tilde{y}_i < y_{i-1} \leq y_i$$

the relation

$$(Qy)_i > 0$$

can be derived. We set

$$\bar{\omega} = (y_i - \bar{y}_i)\frac{Q_{ii}}{(Qy)_i}$$

and hence we can write

$$\vec{y}_i = y_{i-1} = y_i - \frac{\bar{\omega}(Qy)_i}{Q_{ii}}.$$

Again, we just have

$$y^t Qy - \vec{y}^t Q\vec{y} = \frac{(2\bar{\omega} - \bar{\omega}^2)(Qy)_i^2}{Q_{ii}}, \tag{3.12}$$

now with $\bar{\omega}$ appearing in the right hand side of (3.12). To assure non-negativity, we only have to show that $\bar{\omega} \in (0,2)$. $\bar{\omega} > 0$ by construction. Suppose $\bar{\omega} > 2$, then

$$y_i - \bar{y}_i = \frac{\bar{\omega}(Qy)_i}{Q_{ii}} > \frac{2(Qy)_i}{Q_{ii}} > \frac{\omega(Qy)_i}{Q_{ii}}.$$

Thus $\tilde{y}_i > \bar{y}_i = y_{i-y}$ which contradicts the assumption that the the isotonicity constraint is active. Thus the right hand side of (3.12) is nonnegative and even positive if the isotonicity constraint was not active before.

4. Now we take a close look at the height correction step by the same techniques. Let δ denote the vector of the height correction of, say, one constancy interval,

$$\delta = \left(0, \dots, 0, \frac{\omega \sum_{j=\nu}^{\nu+k}(Qy)_j}{q}, \dots, \frac{\omega \sum_{j=\nu}^{\nu+k}(Qy)_j}{q}, 0, \dots, 0\right)^t,$$

and

$$q = \sum_{i,j=\nu}^{\nu+k} Q_{ij}.$$

Hence,

$$y^t Q y - (y-\delta)^t Q (y-\delta) = \frac{(2\omega + \omega^2)\left(\sum_{j=\nu}^{\nu+k}(Qy)_j\right)^2}{q} \geq 0.$$

Equality in the latter relation is attained if

$$\sum_{j=\nu}^{\nu+k}(Qy)_j = 0,$$

which coincides with the constancy interval having been at optimal height before. Again, the same techniques as above can be applied if the constraints do not allow the local optimal height: We get another ω for the above representation and still a reduction of the function value.

Up to now, we have shown that a correction of the y according to the modified QSOR procedure reduces the function value or keeps it if

- all free y_i (i.e. y_i where neither a bound is touched nor a monontonicity constraint is active) are locally optimal, i.e.

$$(Qy)_i = 0.$$

- a bound is attained.
- y_i belongs to a constancy interval of local optimal height, i.e.

$$\sum_{j=\nu}^{\nu+k}(Qy)_j = 0$$

Constancy intervals which touch bounds can be treated as one point which is in touch of the bounds, so we can restrict to free constancy intervals.

Clearly, free constancy intervals cannot be left unchanged until

$$\sum_{j=\nu}^{\nu+k}(Qy)_j = 0$$

is fulfilled because neighbouring points (which can prohibit the optimum height to be achieved) have strictly greater or less height, thus the height can only remain unchanged if the local optimal height was attained before.

It is left to show now that this vector already represents the global minimum. We need a standard lemma.

Lemma 3.8 *Let $C \subset \mathbb{R}^{n+1}$ be a convex set and $S_n \in C^1(\mathbb{R}^{n+1})$ a convex real valued function. If, for all $y \in C$*

$$\sum_{i=0}^{n} \frac{\partial F}{\partial x_i}(y^\star)(y - y^\star)_i \geq 0,$$

then $y^\star$ is a solution of the restricted minimum problem: Minimize S_n over C .

This is p. 19, Lemma 1.7 of [Großmann and Terno (1993)]. In other words: We only have to show that a vector which is not altered by the modified QSOR method fulfills the criterion of Lemma 3.8.

Now we are able to finish the proof of Theorem 3.7. Let $y^\star$ be a vector which is not altered by another QSOR step and $y \in C$ an arbitrary vector consistent with all constraints. Then,

$$\begin{aligned}
\sum_{i=0}^{n} \frac{\partial S_n}{\partial x-_i}(y^\star)(y - y^\star)_i &= \sum_{i=0}^{n}(Qy^\star)_i(y - y^\star)_i \\
&= \sum_{\text{free}} \underbrace{(Qy^\star)_i}_{=0}(y - y^\star)_i + \sum_{\text{bounds}}(Qy^\star)_i(y - y^\star)_i \\
&\quad + \sum_{\substack{\text{constancy}\\ \text{intervals}}}(Qy^\star)_i(y - y^\star)_i \\
&= \sum_{\text{bounds}}(Qy^\star)_i(y - y^\star)_i + \sum_{\substack{\text{constancy}\\ \text{intervals}}}(Qy^\star)_i(y - y^\star)_i
\end{aligned}$$

Think of $y^\star$ touching the lower bound at point i. Then $y_i \geq y_i^\star$ and

$$y_i^\star - \frac{\omega(Qy^\star)_i}{Q_{ii}} \leq y_i^\star,$$

hence

$$(Qy^\star)_i \geq 0 \Longrightarrow (Qy^\star)_i(y_i - y_i^\star) \geq 0.$$

The same argument works for the upper bound. Now, what about constancy intervals? Since on constancy intervals, we have

$$\sum_{i=\nu}^{\nu+k}(Qy^\star)_i = 0$$

and $y_\nu \leq \ldots y_{\nu+k}$ because y is isotonic. We have

$$\sum_{i=\nu}^{\nu+k}(Qy^\star)_i(y_i - y_i^\star) = \sum_{i=\nu}^{\nu+k}(Qy^\star)_i(y_i - y_\nu).$$

Denoting $w_j = (Qy^\star)_j$ and $\varepsilon_j = y_{\nu+j}$, $0 \leq j \leq k$, we have $\varepsilon \geq 0$ and additionally ε is isotonic.

$$\begin{aligned}\sum_{j=\nu}^{\nu+k} w_j\varepsilon_j &= \sum_{w_j>0} w_j\varepsilon_j + \sum_{w_j<0} w_j\varepsilon_j \\ &\geq \left(\min_{w_j>0}\varepsilon_j\right)\sum_{>0} w_j + \left(\max_{w_j<0}\varepsilon_j\right)\sum_{<0} w_j \end{aligned} \tag{3.13}$$

Clearly,

$$(Qy^\star)_\nu < 0$$

and

$$(Qy^\star)_{\nu+k} > 0,$$

since the starting point of the constancy interval attempts to increase to optimal height but is not allowed to do so because of isotonicity (similarly for the end point which wants to decrease but is not allowed). First, let $k = 1$, then $w_0 = -w_1$ and the right hand side of (3.13) becomes

$$\varepsilon_1 w_1 - \varepsilon_0 w_1 \geq 0.$$

For $k \geq 2$, we note that

$$\begin{aligned}(Q_M y^\star)_{\nu+1} &= 2y_{\nu-1}^\star - 8_\nu^\star + 12y_{\nu+1}^\star - 8y_{\nu+2}^\star + 2y_{\nu+3}^\star \\ &\leq 2y_{\nu-1} - 2y_\nu < 0.\end{aligned}$$

The same arguments give

$$(Q_M y^\star)_{\nu+k-1} \geq 0.$$

Thus, we have

$$\min_{w_j>0} \varepsilon_j = \varepsilon_{\nu+k-1} \geq \varepsilon_{\nu+1}$$

and

$$\max_{w_j<0} \varepsilon_j = \varepsilon_{\nu+1},$$

hence the right hand side of (3.13) is

$$\geq \varepsilon_{\nu+1} \sum_{\nu}^{\nu+k} w_j = 0.$$

Only slight modifications are necessary to extend these calculations for Q_L. We skip the technical details. In summary,

$$\sum_{i=0}^{n} \frac{\partial S_n}{\partial x_i}(y^\star)(y-y^\star)_i \geq 0$$

for all $y \in C$. Hence $y^\star$ is a global minimum of S_n over C. Since there is exactly one minimum of S_n over C, the proof is completed. □

There are several technicalities which have to be considered before we can use the procedure. First of all, we take a look at the choice of the relaxation parameter $\omega \in (0,2)$. In the SOR context, ω corresponds to the spectral radius of the so called SOR operator

$$(D-\omega L)^{-1}\left((1-\omega)D+\omega L^t\right),$$

where $Q = D - L - L^t$ with D a diagonal matrix and L an upper triangle matrix. Even in this situation, the calculation of the ω which delivers the most convergent operator (i.e. the operator which has lowest spectral radius < 1) is a difficult matter. In our situation of QSOR, the constraints may effect the speed of convergence and the optimal choice of ω as well.

Example 3.4 *For each of the four Donoho and Johnstone data sets we performed 1000 full QSOR steps with different ω. We now examine the value of S_n for the different ω. Figure 3.3 makes clear that different constraints can influence the choice of ω. For the first two data sets, any value of ω is a good choice if it is not too small (not smaller than 1 actually). The Blocks data which features many active constraints (and a relatively high amount of S_n — about hundred times less smooth than the Heavisine data of the second panel), exhibits a considerable amount of variation, but the overall tendency is to choose a small value ω. For the last of the four data sets the result improves with the relaxation parameter increasing.. We think that it is not possible to find a completely satisfying default value for ω. It is a logical conclusion to try a data driven choice of ω. We will not go into detail here and leave this problem open to future research, but the next example describes a sort of brute force approach to the problem.*

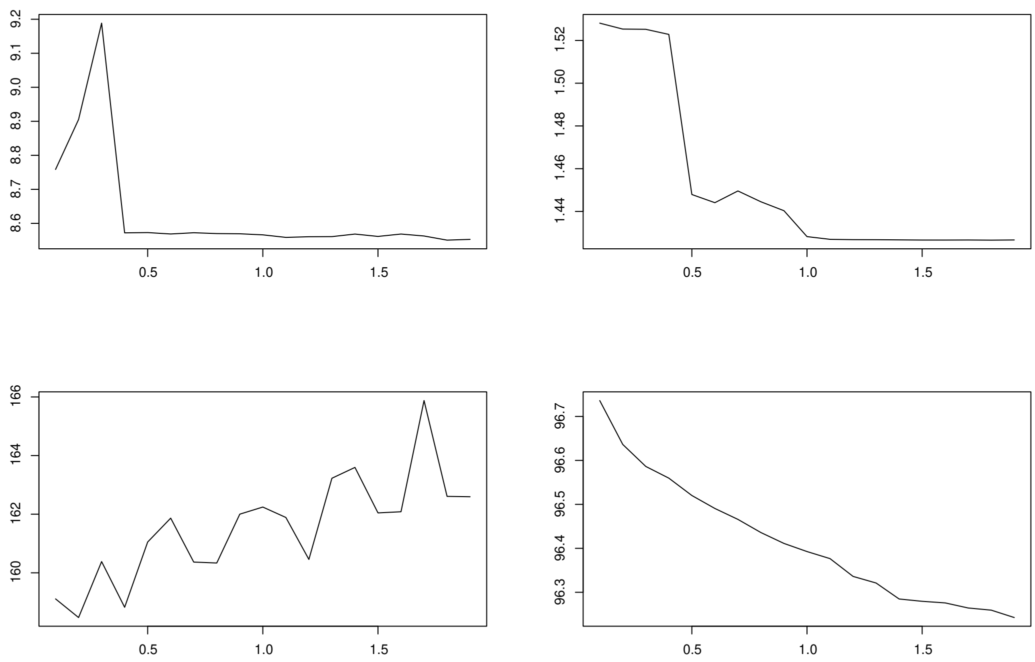

Figure 3.3: The values of S_n for the Donoho data B.3 for different relaxation parameters ω. There is no optimal choice for all four data sets. $\omega \geq 1.5$ seems to be a reasonable choice.

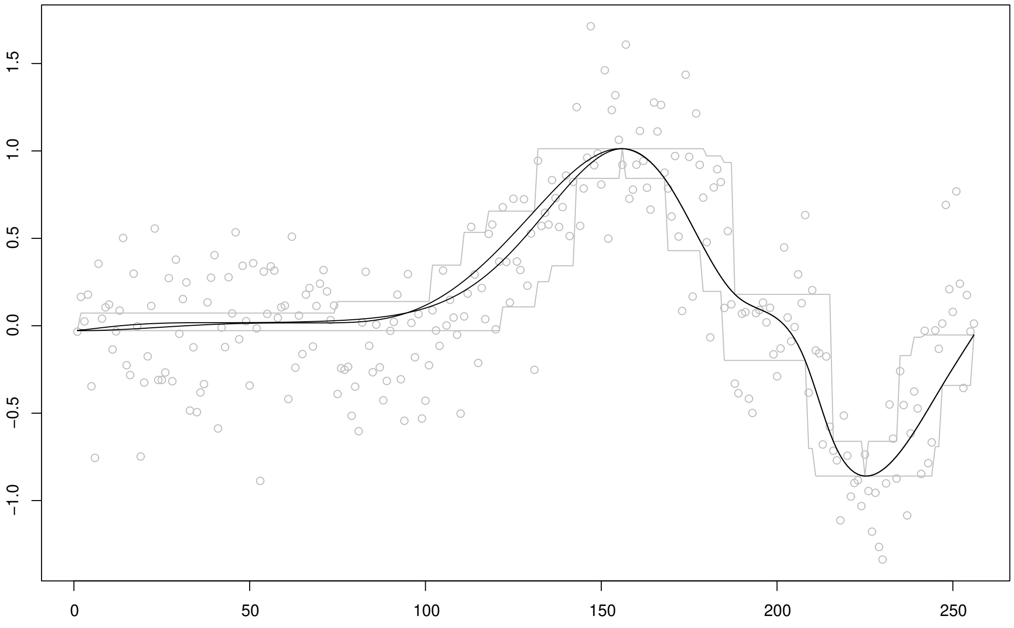

Figure 3.4: 'Exploratory convergence analysis': The prechosen $\omega = 1.8$ against the default $\omega = 1$ for the motorcycle data B.2. The two curves differ considerably.

Example 3.5 *We propose a procedure which can be seen as 'exploratory convergence analysis' in a way. We make some test iterations on our data set for different ω, say 100. Then we choose the optimal parameter value ω_0 and start our routine again with this parameter. We try to demonstrate the power of this approach by the motor cycle data B.2. In Figure 3.4, we can see the results for this prechosen relaxation parameter versus the default $\omega = 1$ for 10.000 iterations: The convergence speed is clearly superior. We can term that in values of S_n as well: The optimized version is about 1% better as the default version which is a considerable amount if we take into account that the value of S_n behaves roughly like $-\log$ in the number of iterations. In all subsequent examples, we used QSOR with this data driven technique of the choice of the relaxation parameter. To achieve the limit of QSOR (which is the optimal solution we already calculated with quadratic programming) optically, it needs about 250.000 full QSOR steps.*

The promising result in Figure 3.5 lets us feel more confident in our minimum

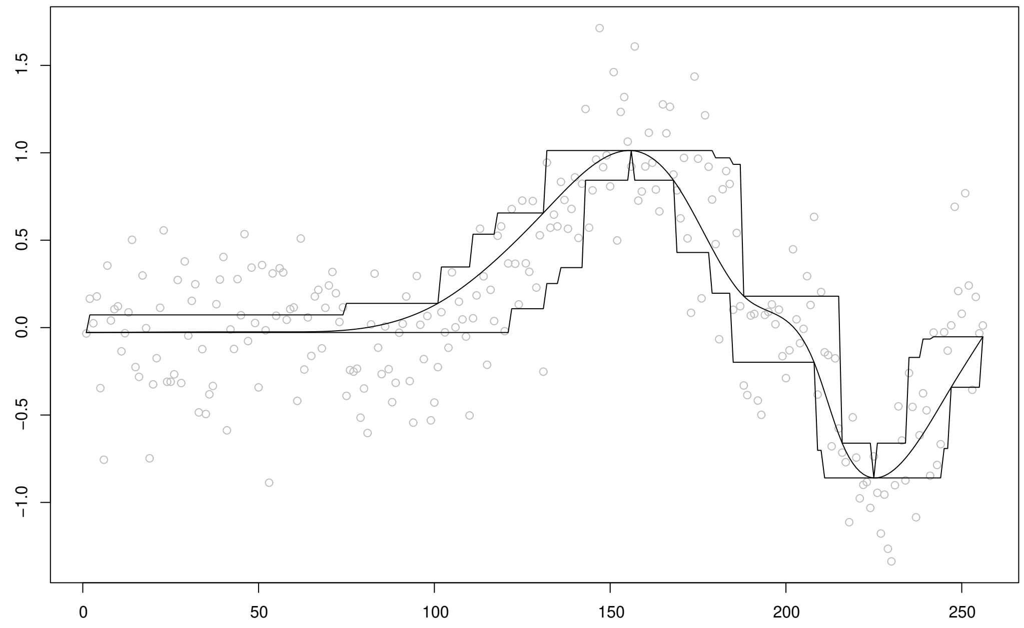

Figure 3.5: The regression function for data set B.2 using the string feature. The modified QSOR algorithm is iterated until no further changes are detected visually after another 10.000 iterations.

roughness approach. The quadratic programming approach is of no use with larger data sets. Thus, we are forced to apply the QSOR method. But even after 100.000 QSOR iterations (for a data set of length 256!) and carefully chosen relaxation parameter, the algorithm has not yet converged completely. It can be seen clearly in Figure 3.4 that further iterations will optimize the constancy interval on the left boundary of the data to touch the lower bound. Unfortunately QSOR takes a very long time until this optimization is achieved, and the value of S_n changes only on a very small scale of less than 0.01 % from 10.000 to 250.000 iterations. For larger data sets, the problem of slow convergence is even worse. We refer to the the Donoho and Johnstone data B.3. The second problem that can be seen from the Härdle data B.2 is that it seems difficult to formulate a satisfactory stopping criterion for the iterations. The tiny difference of less than 0.01% in value for S_n yields graphically important differences in the curves. This makes us think of other criteria than such constructed from the value of S_n.

The next example demonstrates again the power of the smoothness approach by really pleasing figures and thus strongly animates us to construct suitable stopping criteria and faster algorithms.

Example 3.6 *We iterated the QSOR algorithm to death for the four Donoho–Johnstone test signals B.3. Figure 3.6 shows that our method performs well on these data which are extremely hard to analyse , delivering very smooth pictures without artificial modalities. Kernel estimators, wavelets, smoothing splines, local polynomials and other existing nonparametric techniques would contain additional modality and/or be biased at the extrema if the bandwidth parameter are chosen to make the resulting estimates comparable smooth. In the case of the Doppler signal wavelet thresholding performs very well as local polynomials do for the Blocks signal. This is due to the fact that the underlying signals themselves belong to the class of functions which is used to approximate the data. But even for those signals, our method delivers reasonable models. Again, we have to admit that the computation time is crucial: We had to invest 500.000 iterations to achieve the curves, each of (at least) order* n. *But if computation time is of minor importance, the results of QSOR are really pleasing. If only 100,000 iterations less are used then the eye still detects deviations from the result of half a million iterations. Recall that quadratic programming is practically impossible for this size of data sets because of both time and memory consumption.*

Let us summarize now the conclusions of this section: The QSOR algorithm can be shown to converge to the optimal solution. It is only linear in memory consumption and each iteration step is of order n. The convergence speed is very slow, in particular when there are few active constraints. For a wider applicability of the minimum roughness approach, either a method must be

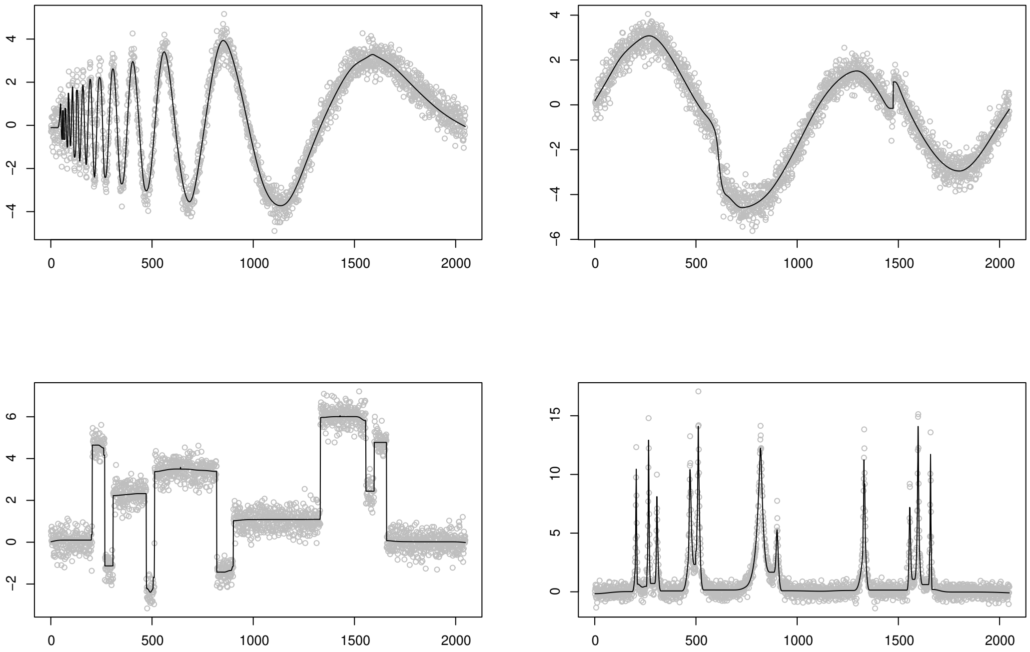

Figure 3.6: The Donoho–Johnstone test signals are hard to analyse, but the minimum–smoothness–under–constraints approach gives pleasingly smooth curves without artificial modality

found of reducing the number of iterations required or some completely different approach must be developed. Both will be done in this thesis. We return to the second point later in Chapter 4, but let us now concentrate on the first.

In an attempt to reduce the number of iterations we make use of our knowledge of the form of the solution for the continuous problem gathered in Section 3.1. We assume that the minimum of the discrete problem is the vector $(z(x_i))^t_{i=0,\dots,n}$ with z being the solution of the continuous problem, i.e. z a generalized spline with knots in some x_i and some intervals of constancy according to Theorem 3.1.

This assumption motivates the following hybrid algorithm:

Step 1 Perform some iterations with QSOR and mark all points where the bounds are attained.

Step 2 Calculate the spline with the correct monotonicity behaviour through these knots.

Step 3 If the spline satisfies all constraints of the discrete problem then stop. Otherwise, return to Step 1.

In order to implement such an algorithm, we require a fast algorithm for Step 2 which we discuss in the next section.

3.6 Smooth isotonic interpolation on a grid

Much research has been devoted to shape restricted interpolation, for example [Kvasov (1996)], [Beatson and Wolkowitz (1989)], [Diereckx (1980)], [Reinsch and Dauner (1989)], [Reinsch (1988)], [Andersson and Elfving (1988)], [Späth (1990)], [Constantini (1986)], and [Schumaker (1983)] to name just a few. But as far as we are aware attention has been restricted to the construction of an interpolating function which is smooth and minimizes the approximation error.

There are some basic principles underlying the construction of shape retaining interpolation functions. Isogeometry is defined from the interpolation data: The interpolation function should be convex/concave and monotone just like the interpolation points. See [Wichmann (1998)] for a survey and a comparison on this topic.

As a matter of fact cubic splines with knots at the interpolation data do not have enough degrees of freedom to satisfy the additional inequality constraints on the derivative of order 0, 1, 2 or some combinations. Two different ways to overcome this difficulty are possible:

- additional knots outside the interpolation data (generalized splines).
- higher order splines (e.g. quintic splines or rational splines).

Higher order splines do not apply to our problem because we clearly *must* remain in the class of cubic splines to assure the minimum roughness property. On the other hand, introducing one or two additional knots in the data is not sufficient in our situation because only at some constance intervals we have the need to change the cubic interpolation spline. If we do not follow this, we cannot be sure to get close to the QSOR solution by this interpolation technique.

We are interested in the smoothest interpolating function. This problem is more difficult and we are not able to present a solution. Since a possibly countable number of additional knots is required to preserve monotonicity the problem cannot be solved even theoretically in general. However if the global monotonicity problem is relaxed to requiring monotonicity at a finite number of additional knots then in theory this problem can be solved numerically [Wright and Wegman (1978)].

As far as we are aware there is no algorithm which directly computes the generalized spline solution given by [Hornung (1978)]. We describe here an ad hoc method which seems to work well in the context of non–parametric regression. To simplify the notation we restrict ourselves to the case of isotonic interpolation. Given some points (t_i, y_i) with $t_i \in (0,1)$ satisfying

$$t_i < t_j \Rightarrow y_i \leq y_j,$$

we construct a fine regular grid of say m design points $i/(m+1)$. We assume that all t_i are grid points. We now construct a sequence of non-decreasing values at the grid points as follows:

- Construct the uniquely defined natural cubic spline z which interpolates the points (t_i, y_i) and evaluate it at all grid points.
- Examine each interval $K_i = (t_i, t_{i+1})$. Since z is a cubic polynomial on K_i there can be at most two local extrema of z on K_i. If there is no local extremum monotonicity is automatically fulfilled. We consider the two remaining cases:

1. There is exactly one local extremum. If it is a minimum then an interval of constancy is introduced which commences at the left endpoint of the interval K_i. If the local extremum is a maximum then an interval of constancy is introduced which terminates at the right endpoint of K_i. The interval is lengthened until the spline interpolating the knots (t_i, y_i) and the grid points in the interval of constancy is isotone.
2. There are exactly two extrema. This is the most difficult case because the height of the interval of constancy which must be introduced is not known. For the moment, we choose the height of the interval in an adhoc manner: we introduce a knot in the middle between the two extrema (Corrections of the height of knot is necessary if the knots leaves the interval of the heights of the old knots on the left and right). Thus we are in the case of one extremum, applying the above method to both half intervals. Note that the height of constance intervals is corrected by a single QSOR height correction step see Definition 3.2.

Note that it is not clear whether the result of this adhoc interpolation method is of the right monotonicity behaviour because additional knots unfortunately do influence the *global* behaviour of the spline. We can have still some hope that this does not take place because the impact of an additional knots decays roughly exponential with the distance. But we can think of situations, especially when two knots are located very close at different heights, where the spline gets so steep that intervals that were corrected before are strongly influenced. There is no proof that the resulting spline of our adhoc approach satisfies the monotonicity in every possible constellation of original knots, and we believe this is not the case. It seems plausible that our method is not too bad because

- it follows the spirit of Corollary 3.1 and
- we experimented with numerous constellations of knots and our algorithm worked well.

It is comfortable from a computational viewpoint to construct a routine which is fed with the knots and the regular grid and returns a spline at all grid points which retains automatically the shape of the interpolation points. We designed a program like this called `monspl` which can be found in the appendix A.5. For more details, see the appendix where the algorithm is printed out with detailed comments.

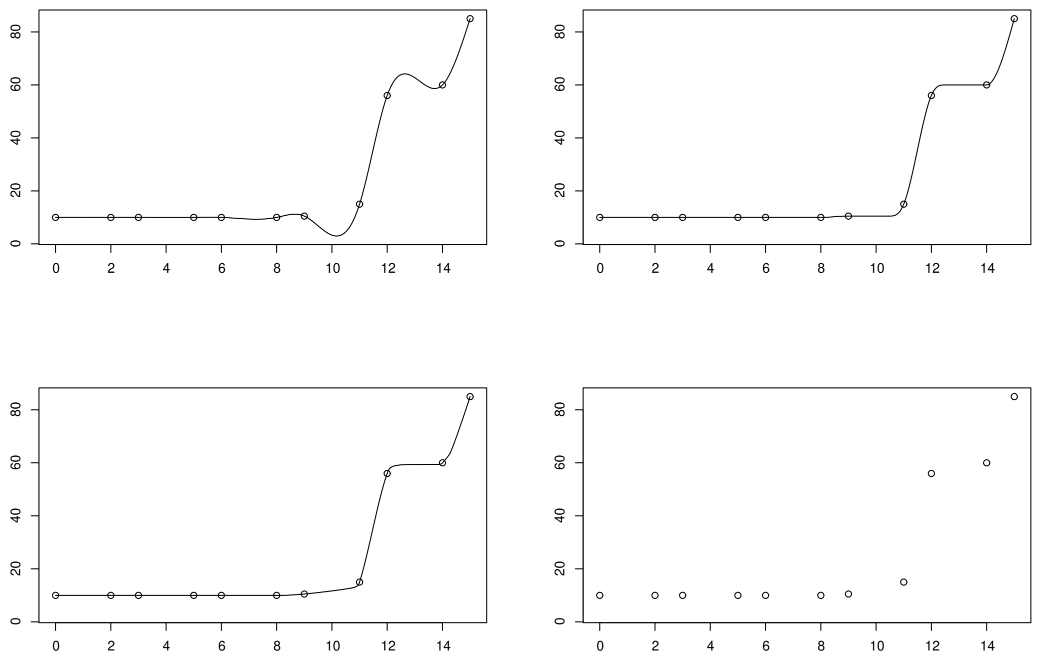

Figure 3.7: The Akima data B.4. Upper panel: unconstrained cubic interpolation and our adhoc method `monspl`, below Kvasov–type interpolation with additional knots and Späth–type rational spline interpolation

We have a look at some examples to get an impression of the power of the adhoc algorithm, analysing the performance and comparing it to existing isogeometric interpolation techniques.

Example 3.7 *Two interpolation data sets are common in the constrained interpolation literature, the Akima data B.4 and the Späth data B.5. The Akima data are interesting because all three defects of the cubic interpolation spline appear in those, while the Späth data come up with a change in monotonicity. To demonstrate that our method is able to compete with other techniques commonly used in numerics, we depict two other interpolation techniques. The approach of [Kvasov (1996)] which is the most recent uses the principle of adding knots if necessary, whilst the rational spline approach of [Späth (1990)] attains additional degrees of freedom by higher order splines. For completeness, we added the unconstrained natural cubic interpolation spline. Note that our adhoc method resembles the constance intervals of the optimal solution from Corollary 3.1 while the other techniques do not.*

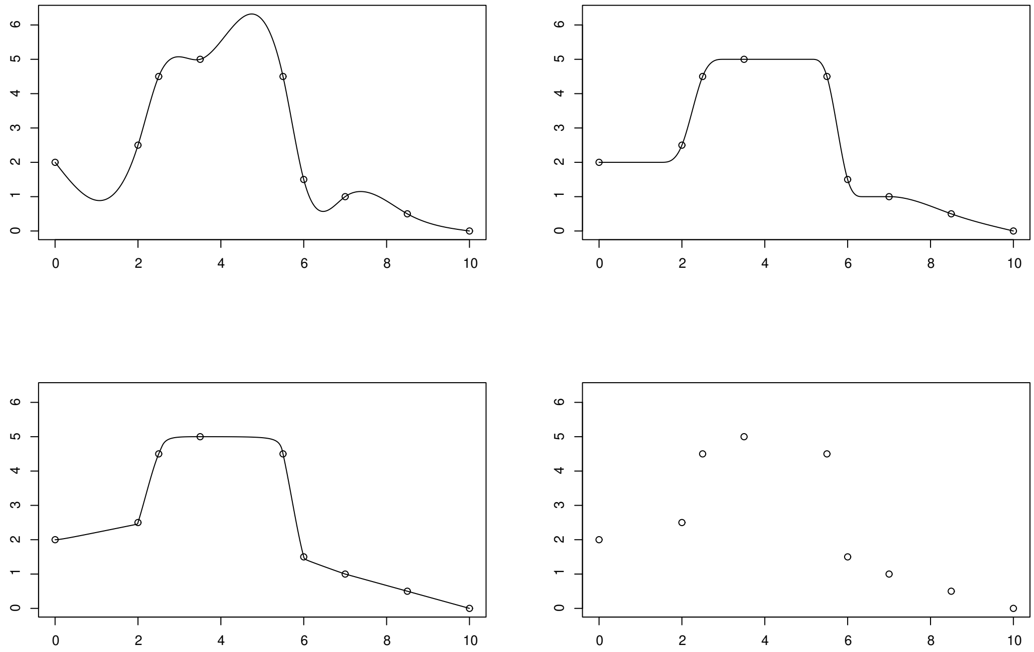

Figure 3.8: The Späth data B.5. Upper panel: unconstrained cubic interpolation and our adhoc method `monspl`, below Kvasov–type interpolation with additional knots and Späth–type rational spline interpolation

The computational complexity of our interpolation method is at most of order m^2 which corresponds with every grid point being added for monotonicity. This gives a piecewise constant function and is not very typical and not the aim of smoothing either. In the regression context, we experienced that constance intervals usually are not very likely and so, we have better performance of the adhoc interpolation method. The concise expected computational complexity is not clear.

3.7 Hybrid algorithm

In the context of convergence acceleration for QSOR, we insert the result of the monotone grid interpolation algorithm of active bound positions based on the design mesh into QSOR. Clearly, the result of this algorithm satisfies the shape constraints but possibly not the bounds. This fact makes us believe that there are active bounds positions in the optimal solution which are still not found by QSOR. Thus, a few more QSOR iterations have to be taken into account. On the other hand, if the bounds are satisfied by the `mon.spline` output, we have the strong hope that this must have been the optimal knots and thus, our vector is close to the solution.

We want to use the QSOR algorithm to detect knots, i.e. those points of the curve which touch the boundaries l and u. These knots are the ones in the interpolation routine described in the last section.

Note that we have tackled two severe problems with just one idea: the problem of termination criterion and of convergence acceleration.

Definition 3.3 *We give a more concise description of what we call the Hybrid algorithm. Fix a relaxation parameter ω for the QSOR steps.*

- *Start with some, say k QSOR iterations. Mark all points touching the bounds (up to a given tolerance level which can be specified in our code). These points will be called knots.*
- *Interpolate the knots using the adhoc interpolation method of the last section,* `monspl`.
- *Check whether the bounds are satisfied by the spline (up to a tolerance level).*
- *If the spline respects the bounds, stop. If not, start again.*

After termination of the iterations, a subsequent QSOR step is made to correct the height of possible constance intervals. More algorithmic and coding details may be found in the appendix, A.6, where the R code of the algorithm `hybrid` *is printed out with detailed comments.*

Remark 3.3

- *There is no proof that the algorithm* must *terminate, because something might happen in the adhoc interpolation step. But in that case we have nothing less than a QSOR approximation of the solution in each step. The hybrid algorithm is therefore definitely not worse than the QSOR approach. In our program code (see A.6), we solved the problem of endless iterations by the specification of an upper limit of QSOR iterations which should not be exceeded in the flow of the algorithm.*

- *Clearly, if it terminates, the solution is permissible with respect to the bounds and monotonicity constraints since it respects the bounds and their monotonic behaviour according to the construction of the spline.*

- *It seems plausible that the spline solution after termination is the smoothest function through the knots and thus as good a candidate for a regression model as the QSOR solution is.*

- *Each iteration of the hybrid algorithm involves k QSOR steps, the adhoc interpolation and a check of the bounds constraints, i.e. we have an overall computional complexity of at worst some order n^2 because of the interpolation step. In this case, the whole solution consists of constance intervals and we do not need any further hybrid iterations. If the interpolation is not crucial, we have some order n complexity for each iteration.*

- *In our implementation, we use $k = 10$ and $\omega = 1.5$ as default values.*

QSOR converges very slow when few constraints are active (Figure 3.5). Roughly speaking, it takes a while until the alternation of second order differences (derivatives) has a considerable impact on the function values themselves. We can try to speed up this convergence if we insert spline pieces of the interpolation spline.

Definition 3.4 *Fix a relaxation parameter ω for the QSOR steps and an additional parameter $\alpha > 1$.*

- *Start with k QSOR iterations. Mark all knots.*

- *Interpolate the knots using* `monspl`*.*
- *Check whether the bounds are satisfied by the spline.*
- *If the spline respects the bounds, stop. If not, start over using the spline (projected to the the set of permissible vectors) as start vector for QSOR. Set $k \leftarrow \alpha k$.*

Again, a subsequent QSOR for height correction is made. This variant of `hybrid` *in A.6 is activated by the parameter setting* `insert.spline=T`*.*

Remark 3.4

- *The parameter α in the definition of the algorithm is important because there are situations where $\alpha \leq 1$ leads to loops which cannot be left by further iterations. Consider wrongly placed knots and too few QSOR steps to correct them up to the specified tolerance level. Then, having the same knots again, the spline method delivers the same non–permissible vector. Again, the next QSOR phase will not be able to correct this. If we use increasing QSOR sequences (which tend to ∞) like $k\alpha^n$ for $\alpha > 1$, this problem is avoided.*
- *Since we introduced the spline in the QSOR iterations, it might happen that the spline is worse than the last QSOR result. Thus, in opposite of the first variant of the algorithm, the degree of smoothness might be decreased in the flow of the algorithm. Possibly, the performance might be worse than QSOR alone. This is contrary to our empirical experience. We will demonstrate that later on.*

First of all, we demonstrate the power of the hybrid approach for the Härdle data.

Example 3.8 *Again, we used the bounds construction using the string feature for the Härdle data B.2. Four different methods for the calculation of the vector of minimal roughness were applied: quadratic programming (QP) using the Goldfarb–Idnani type algorithm of [Turlach (1998)], QSOR, and finally the hybrid algorithm with and without spline insertion. It takes more than 400.000 iterations to make the maximum absolute difference of the QSOR result and the QP solution less than 0.001, but visually the curves are indistinguishable after at least 300.000 iterations which corresponds to a maximum absolute distance of less than 0.01. Thus it is natural to terminate QSOR after 300.000 iterations.*

Method	*CPU time*
QP	*6.00 sec.*
QSOR	*1 min.*
Hybrid (`insert.spline=F`)	*7.00*
Hybrid (`insert.spline=T`)	*0.07*

In this particular case, the hybrid method with spline insertion clearly outperforms all other methods while QSOR cannot compete. The hybrid method with spline insertion reduces the number of QSOR iterations required dramatically: only 34 iterations (which corresponds to 3 spline insertions) were necessary for termination (depends on default settings, here $k = 10$ and $\alpha = 1.1$). Quadratic programming in this case is realisable, but for larger data sets is inappropiate in time and memory consumption even if more modern convex quadratic programming algorithms of some order n^3 are used which yield polynomial worst case complexity (see [Goldfarb and Liu (1993)]). The hybrid approach without spline insertion is tolerable in calculation time and needs about 20.000 QSOR iterations which means that we saved about 90 % of the iterations compared to QSOR alone.

We do some simulation experiments to benchmark the different algorithms.

Example 3.9 *For each simulation, we generated a new version of the test signals B.3 of length 2048 and contaminated it with Gaussian white noise, i.e. $\mathcal{N}(0, 0.4)$ pseudo random numbers. These data sets are versions of the test signals exactly as they were introduced before, e.g. in Figure 3.6.*

We tested the hybrid approach with and without spline insertion and gathered the success rate (which is the percentage of experiments where the hybrid method needs less than 500,000 QSOR iterations) as well as the QSOR steps needed for the successful hybrid attempt. After having reached the limiting number of 500,000 iterations, the algorithm was stopped and the experiment was counted as failure. The limiting number of 500,000 was chosen carefully to produce a QSOR curve close to the minimum visually. Because of this, success in this context means that the hybrid algorithm has effectively saved some QSOR iterations and has speeded up computation, failure stands for the hybrid algorithm not yet having terminated. The tolerance parameter was fixed at very small 10^{-6}.

Test signal	*spline insertion*	*success rate in % (based on 100 exp.)*	MED *of QSOR steps if success*	MAD *of QSOR steps if success*
Doppler	*T*	*0.06*	*5712*	*4433*
Heavisine	*T*	*0.16*	*8068*	*7983*
Blocks	*T*	*0.68*	*34*	*12*
Bumps	*T*	*0.53*	*47*	*13*
Doppler	*F*	*0.00*	-	-
Heavisine	*F*	*0.02*	*285832*	*13597*
Blocks	*F*	*0.15*	*127*	*111*
Bumps	*F*	*0.17*	*6767*	*4925*

We took median and MAD as location and dispersion measures because there are some cases of success where a unusually big number of QSOR iterations is necessary. This would damage the mean and give a too pessimistic impression of the average required iteration number. As we would have expected smaller success rates are achieved for the same experiment done without spline insertion because spline insertion has a greater chance to get it right before the algorithm is forced to terminate when the limiting number of QSOR iterations is reached.

There are some considerable differences in performance among the different test signals. This shows the strong dependence of the QSOR performance on the specific nature of the data driven constraints. This was already reported by Metzner in [Metzner (1997)]. The hybrid approach inherits this from QSOR and shows the same significant differences between the test signal. If the bounds are narrow, QSOR is able to detect the correct touching points very quickly and the hybrid algorithm terminates. Thus the success rates for the narrow bounds of the Blocks and the Bumps signal are much higher than those of Doppler and Heavisine. It is particularly difficult to find a spline through the oscillating and tough bounds of the Doppler signal which makes it astonishing that the hybrid method did terminate at all in some realizations.

Clearly, the success rate heavily depends on the specified tolerance level at which the spline is tested to be permissible and at which points are counted to be touching the bounds: smaller tolerance levels will give better success rates, but there is the danger of accepting a suboptimal spline solution which satisfies the bounds up to the tolerance level by "bad luck". The upper simulation example gives a pessimistic view of the success of the hybrid algorithm because the tolerance level is chosen very small. There is some justification to be more generous with the tolerance because the constraints that have to be fulfilled are derived from the data and so come up with some noise.

We demonstrate the effect of tolerance level setting by another simulation experiment. This time we only used the Heavisine test signal again consisting of 2048 data points (each time a new realization was generated). We fixed some different tolerance levels and calculated success rates and success iteration numbers for spline insertion as above. Clearly, the success rate and the required iterations in case of success tends to decrease with decreasing tolerance level. In order to measure whether the hybrid method had reached the optimum, we applied another 10,000 QSOR iterations on the solution in the case of success and evaluated the maximum absolute distance between both. This number should be small. To have a reference, we made an experiment with QSOR alone up to the limiting number of iterations, i.e. 500,000. Then, another 10,000 iterations were applied and maximum absolute distance is calculated, too. We took the mean of about 10 experiments of this kind. Thus, we can calculate a relative maximum absolute distance as the ratio of the hybrid method result and the reference distance. This gives an answer to the question whether the hybrid approach delivers the optimum of the minimum problem or not: Is the relative maximum absolute distance far from 1, we can conclude that the hybrid method gives a suboptimal result.

Tolerance	*success rate*	*iterations in success*	*median of max. abs. dis. in success*	*relative max. abs. dis.*
1E-5	*0.6*	*1754*	*0.14*	*6.1*
1E-4	*0.3*	*188*	*0.19*	*8.26*
1E-3	*0.5*	*61*	*0.34*	*14.78*
1E-2	*0.9*	*47*	*0.35*	*15.21*

It is obvious that we have to pay for the higher success rates since then: The higher the tolerance level is, the more confidence we can have in the result of the hybrid approach but on the other hand, success rates decrease. This tradeoff makes it complicated to give a reasonable default for the tolerance level in practice, because success heavily depends on the constraints given.

We have to mention a strange fact in the context of the tolerance level problem: Sometimes too big tolerance levels make it more complicated to find the spline solution in the hybrid approach. The reason for this effect is that the algorithm tends to accept suboptimal knots (i.e. touching points with the bounds) for big tolerance levels. Thus the interpolating spline may reveal some big scale errors. This is shown in the following table.

tolerance	*success rates (relative max. abs. dis.) for*			
	Doppler	*Heavisine*	*Blocks*	*Bumps*
0.001	*0.2 (6.8)*	*0.1 (1.4)*	*0.5 (1.8)*	*0.8 (8.4)*
0.0001	*0.3 (4.1)*	*0.1 (2.55)*	*0.7 (1.4)*	*0.6 (10.9)*

Based on ten experiments, the numbers of the table are somewhat noisy. But we can see that there is no clear tendency to an improvement of the success rate for the smaller tolerance level. If we take a look on the relative maximum absolute distances we see that the smaller tolerance level can lead to higher values which means suboptimal termination of the algorithm, e.g. for the Doppler and the Bumps signal. Besides, it seems reasonable to talk of optimal termination of the hybrid approach only for two of the four signals Doppler and Blocks.

Finally, we try to give an estimation for the improvement in overall computational complexity we made with the help of the hybrid algorithm. Clearly, we have not lost anything compared with the QSOR algorithm. To measure the improvement, we downweight the median computational complexity of successful applying the hybrid method with the success rate r in the following way:

$$\frac{(1-r)\cdot \mathit{QSOR\ iteration\ limit} + r \cdot \mathit{median\ hybrid\ iteration}}{\mathit{QSOR\ iteration\ limit}}.$$

This number can be interpreted as a mean rate of computational complexity of the hybrid method compared to the QSOR method alone. We give the numbers for two reasonable choices of the tolerance level based on 10 repetitions of the experiment.

Tolerance level	*mean comput. compl. rate of Hybrid versus QSOR*			
	Doppler	*Heavisine*	*Blocks*	*Bumps*
0.0001	*0.99*	*0.70*	*0.30*	*0.40*
0.001	*0.90*	*0.80*	*0.50*	*0.20*

Again, the dependence on the tolerance level for different signals is not a simple function and we have considerable computation saving, e.g. for the Blocks and the Bumps signal, moderate for the Heavisine and few or no savings for the Doppler signal.

We conclude that the hybrid method sometimes is very powerful (see Example 3.8), but the performance depends heavily on the data at hand. It is even possible that almost no improvement occurs compared to QSOR alone. On the other hand the acceleration can be of a factor of 100 or more in certain situations. Since no loss is made compared to QSOR alone using the hybrid method, it seems reasonable to use the hybrid method with a certain QSOR limit which delivers a function sufficiently close to the optimum without any uncertainty. It is also possible to use a decreasing sequence of tolerance levels to speed up computations but no work in this direction has been made up to now. The choice of the tolerance level remains crucial and we are not able to give a default value.

It is obvious that we were able to accelerate the QSOR procedure considerably, but the effort we made is somewhat insecure or instable. Even in case of successful performance of the hybrid algorithm, we sometimes have to accept a certain high amount of QSOR iterations. The main reason for the complexity of the problem is the spline solution being not "local" enough even though the amplitude of interaction decreases roughly exponential with increasing distance.

Maybe it is too demanding to exactly fulfill constraints which are somewhat noisy because they are constructed from noisy data. It seems reasonable then to satisfy data driven constraints only in spirit and not exactly.

In the next chapter, we try a completely different approach to the smoothing problem which is

- local,
- has the exact monotonic behaviour, and
- does not satisfy the bounds exactly, but in spirit.

Chapter 4

Suboptimal But Sufficiently Smooth Functions

saepe creat molles aspera spina rosa

Ovid

4.1 Strings Revisited

Since we have seen up to now that the calculation of the smoothest adequate regression model seems to be too hard to handle, we propose another technique which produces a smooth curve which comes close to the optimal model but can be calculated very quickly. We are able to compute an at least continuous model through the bounds and monotonicity constraints: the taut string between the bounds. It can be seen very easily that this candidate has the right number of extrema, because the taut string is the function of shortest length between the bounds. Thus there can be no dips or bumps in it except those that are forced to be there by the bounds, see [Davies and Kovac (1999)].

Example 4.1 *In Figure 4.1 the Donoho and Johnstone data are shown and the string through the bounds constructed by the string method. The edges of the string, i.e. points touching the bounds, are marked. It is remarkable that the strings resemble the underlying signals very well. The only disadvantage is that the string does not look very smooth. Perhaps the model would look much better if we round the edges.*

Motivated by these results, our idea is to polish the string estimator to be twice differentiable by substituting the linear pieces of the string by a parametric function which is smooth (at least twice differentiable) and retains the

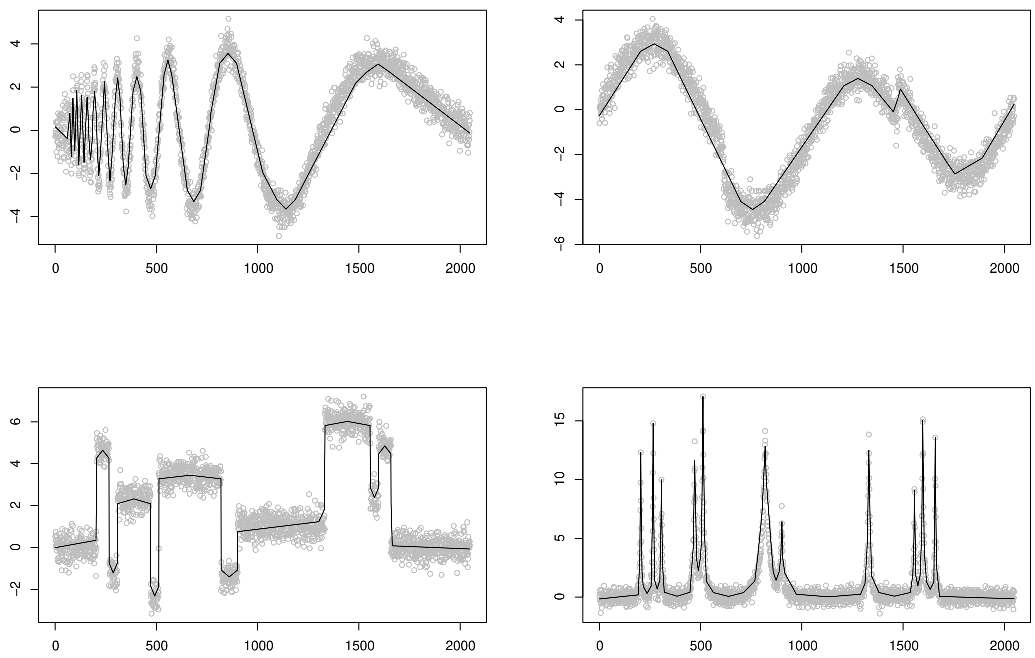

Figure 4.1: The Donoho et al. data and the string through the bounds.

modality structure of the string. In order to preserve the modality, we need a parametric family of functions that resemble the monotonicity of the string and is able to reproduce prescribed boundary values for first and second order derivatives. We develop a family like that in Section 4.2.

Another part of the algorithm is to give rules for the prescription of boundary derivatives for the knots of the string that allow for a parametric representation which resembles montonicity and convex/concaveity. We propose an algorithm which ties up the different parametric pieces and gives adhoc rules for the boundary values in Section 4.3. Some simulation study is carried out.

It is clear that the piecewise parametric function is calculable very quickly because the complexity of calculations only depends on the number of linear pieces of the string which is small compared to n. This is perhaps the greatest advantage of the polished string approach of this Chapter.

4.2 A parametric family

Since we want to be sure that no additional extrema and inflection points are introduced by substituting a straight line piece of the string by a smooth parametric function of prescribed boundary derivatives, the shape of parametric function is restricted.

First, simple considerations show that without loss of generality we can assume that the function is defined on the unit interval $[0, 1]$. It is immediately clear that adding a straight line being as steep as the prescribed value of first order derivative in 0 to the function will not alter isotonic behaviour and the second order derivative (and thus convexity/concaveity). This allows to restrict our attention to functions with a vanishing first order derivative in 0.

Suppose now we have a smooth function g which is convex and satisfies the following boundary conditions:

$$
\begin{aligned}
g(0) &= 0 && (4.1)\\
g(1) &= y_1 && (4.2)\\
g'(0) &= 0 && (4.3)\\
g'(1) &= y_1' && (4.4)\\
g''(0) &= y_0'' && (4.5)\\
g''(1) &= y_1'' && (4.6)
\end{aligned}
$$

for some given (prescribed) nonnegative values y_1, y_1', y_0'', y_1''. If either y_1 or y_1' vanishes, then the other one has to vanish as well, then forcing $y_0'' = y_1'' = 0$.

Thus we have the trivial solution $g \equiv 0$. Therefore we can restrict to positive y_1, y_1'. Note that $y_0'', y_1'' = 0$ with positive y_1, y_1' does make sense and we do not want to exclude this case.

Now we have to face the following problem: Find a parametric family of functions

$$G_{\tilde{P}} = \{g_{\tilde{P}}(t)\},$$

$\tilde{P}$ denoting the set of possible parameters, which is convex on $[0,1]$ and flexible enough to reconstruct all possible combinations of boundary values y_1, y_1', y_0'', y_1''.

First we examine if there are any combinations of boundary values which cannot be resembled by a convex function g.

Lemma 4.1 *Let g be a smooth convex function satisfying (4.1)–(4.6). Then the following holds:*

- *g is isotonic.*
- *There exists a convex function g retaining the boundary values iff*

$$\frac{g(1)}{g'(1)} < 1.$$

Proof of Lemma 4.1: Since $g''(t)$ is nonnegative for all t, $g'(t)$ must be isotonic. Thus the minimum of $g'(t)$ is attained in 0. Hence

$$g'(t) \geq g'(0) = 0$$

which proves the first point. Since g' is isotonic, we have

$$\max_{t\in[0,1]} g'(t) = g'(1)$$

and therefore

$$g(1) = \int_0^1 g'(t)dt \leq \max_{t\in[0,1]} g'(t) \int_0^1 dt = g'(1).$$

Equality is only attained if $g'(t) = g'(1)$ for all $t \in [0,1]$ thus $g'(1) = 0$ which is contrary to our previous assumption of positive y_1, y_1'. Thus

$$\frac{g(1)}{g'(1)} < 1.$$

□

We have stated that boundary values y_1, y_1' can only be derived from a convex function if the ratio y_1/y_1' is less than 1. We must now design a parametric family which can deal with all those boundary values.

Our proposal is a family of seven parameters, i.e.

$$\tilde{P} = \{a, b, c, \alpha, \beta, \gamma, \delta\},$$

of the following type:

$$g''_{\tilde{P}}(t) = at^{\alpha} + b(1-t)^{\beta} + c(1-t)^{\gamma}t^{\delta}. \tag{4.7}$$

We only allow nonnegative values for the parameters, thus the second derivative of g is positive so that the desired convexity is guaranteed. By

$$g'_{\tilde{P}}(t) = \int_0^t g''_{\tilde{P}}(x)dx,$$

the non–negativity of the first order derivative (i.e. isotonicity) is satisfied as well. The integration of (4.7) is only possible in a closed form if one of the exponents of the mixed products is an integer, thus allowing for partial integration. Because of this, we set $\delta = 1$, and hope that we do not restrict the family too much. It is also possible to set $\gamma = 1$. This will cause only minor differences. We return to this later.

Now, with $P_\gamma = \{a, b, c, \alpha, \beta, \gamma, 1\}$, the above integral can be explicitely calculated. The result is

$$\begin{aligned} g'_{P_\gamma}(t) &= \frac{a}{\alpha+1}t^{\alpha+1} - \frac{b}{\beta+1}(1-t)^{\beta+1} + \frac{b}{\beta+1} - \frac{c}{\gamma+1}(1-t)^{\gamma+1}t \\ &\quad - \frac{c}{(\gamma+1)(\gamma+2)}(1-t)^{\gamma+2} + \frac{c}{(\gamma+1)(\gamma+2)}. \end{aligned} \tag{4.8}$$

Partial integration yields

$$\begin{aligned} g_{P_\gamma}(t) &= \frac{a}{(\alpha+1)(\alpha+2)}t^{\alpha+2} \\ &\quad + \frac{b}{(\beta+1)(\beta+2)}(1-t)^{\beta+2} - \frac{b}{(\beta+1)(\beta+2)} + \frac{b}{\beta+1}t \\ &\quad + \frac{c}{(\gamma+1)(\gamma+2)}(1-t)^{\gamma+2}t + \frac{2c}{(\gamma+1)(\gamma+2)(\gamma+3)}(1-t)^{\gamma+3} \\ &\quad - \frac{2c}{(\gamma+1)(\gamma+2)(\gamma+3)} + \frac{c}{(\gamma+1)(\gamma+2)}t. \end{aligned} \tag{4.9}$$

We sample the values at the left and right border, which must have the prescribed values:

$$
\begin{aligned}
g_{P_\gamma}(0) &= 0 \\
g_{P_\gamma}(1) &= \frac{a}{(\alpha+1)(\alpha+2)} - \frac{b}{(\beta+1)(\beta+2)} + \frac{b}{\beta+1} \\
&\quad - \frac{2c}{(\gamma+1)(\gamma+2)(\gamma+3)} + \frac{c}{(\gamma+1)(\gamma+2)} && (4.10) \\
g'_{P_\gamma}(0) &= 0 \\
g'_{P_\gamma}(1) &= \frac{a}{\alpha+1} + \frac{b}{\beta+1} + \frac{c}{(\gamma+1)(\gamma+2)} && (4.11) \\
g''_{P_\gamma}(0) &= b && (4.12) \\
g''_{P_\gamma}(1) &= a. && (4.13)
\end{aligned}
$$

Thus, the parameters a, b coincide with y''_1, y''_0. But: Is it possible to calculate parameters c, α, β, γ in order to fit the other prescribed values y_1, y'_1?

First, we note that (4.10) can be written as

$$c = \left(g(1) - \frac{a}{(\alpha+1)(\alpha+2)} + \frac{b}{(\beta+1)(\beta+2)} - \frac{b}{\beta+1} \right) (\gamma+2)(\gamma+3), \tag{4.14}$$

and (4.11) as

$$c = \left(g'(1) - \frac{a}{\alpha+1} - \frac{b}{\beta+1} \right) (\gamma+1)(\gamma+2). \tag{4.15}$$

Hence, joining (4.14) and (4.15), we have

$$\frac{p(\alpha,\beta)}{q(\alpha,\beta)} = \frac{\gamma+1}{\gamma+3} \tag{4.16}$$

with the abbreviations

$$
\begin{aligned}
p(\alpha,\beta) &= g(1) - \frac{a}{(\alpha+1)(\alpha+2)} + \frac{b}{(\beta+1)(\beta+2)} - \frac{b}{\beta+1} \\
&= g(1) - \frac{a}{(\alpha+1)(\alpha+2)} - \frac{b}{\beta+2} && (4.17) \\
q(\alpha,\beta) &= g'(1) - \frac{a}{\alpha+1} - \frac{b}{\beta+1}. && (4.18)
\end{aligned}
$$

Note that $p(\alpha,\beta), q(\alpha,\beta)$ must be positive to give permissible values for c. Clearly, (4.16) is solvable for γ if and only if

$$\frac{1}{3} \le \frac{p}{q} < 1, \tag{4.19}$$

yielding

$$\gamma = \frac{q - 3p}{p - q}. \tag{4.20}$$

Recalling the second point of Lemma 4.1, we see that the range (4.19) does not cover the interval $(0, 1)$ for permissible values completely. Thus we have restricted the family too much by setting $\delta = 1$. Now, what happens if we choose $P_\delta = \{a, b, c, \alpha, \beta, 1, \delta\}$? We have

$$\begin{aligned} g'_{P_\delta}(t) &= \frac{a}{\alpha+1}t^{\alpha+1} - \frac{b}{\beta+1}(1-t)^{\beta+1} + \frac{b}{\beta+1} \\ &\quad + \frac{c}{\delta+1}t^{\delta+1} - \frac{c}{\delta+2}t^{\delta+2} \end{aligned}$$

and

$$\begin{aligned} g_{P_\delta}(t) &= \frac{a}{(\alpha+1)(\alpha+2)}t^{\alpha+2} \\ &\quad + \frac{b}{(\beta+1)(\beta+2)}(1-t)^{\beta+2} - \frac{b}{(\beta+1)(\beta+2)} + \frac{b}{\beta+1}t \\ &\quad + \frac{c}{(\delta+1)(\delta+2)} - \frac{c}{(\delta+2)(\delta+3)}t^{\delta+3}. \end{aligned}$$

Thus, boundary values are

$$\begin{aligned} g_{P_\delta}(0) &= 0 \\ g_{P_\delta}(1) &= \frac{a}{(\alpha+1)(\alpha+2)} - \frac{b}{(\beta+1)(\beta+2)} + \frac{b}{\beta+1} \\ &\quad + \frac{2c}{(\delta+1)(\delta+2)(\delta+3)} \\ g'_{P_\delta}(0) &= 0 \\ g'_{P_\delta}(1) &= \frac{a}{\alpha+1} + \frac{b}{\beta+1} + \frac{c}{(\delta+1)(\delta+2)} \\ g''_{P_\delta}(0) &= b \\ g''_{P_\delta}(1) &= a \end{aligned}$$

which leads to

$$\frac{p(\alpha, \beta)}{q(\alpha, \beta)} = \frac{2}{\delta+3}$$

for all $\delta \geq 0$. Again, we have positivity of $p(\alpha, \beta), q(\alpha, \beta)$. Hence, reasonable values for δ can only be achieved if

$$0 < \frac{p(\alpha, \beta)}{q(\alpha, \beta)} \leq \frac{2}{3}. \tag{4.21}$$

Then,

$$\delta = \frac{2q - 3p}{p}. \tag{4.22}$$

Because the union of the ranges (4.19) and (4.21) covers the whole interval $(0, 1)$, we can find a parameter constellation for c, γ, δ if

$$0 < \frac{p(\alpha, \beta)}{q(\alpha, \beta)} \leq 1.$$

But this is quite clear because of

$$\lim_{\alpha,\beta \to \infty} \frac{p(\alpha, \beta)}{q(\alpha, \beta)} = \frac{g(1)}{g'(1)} < 1,$$

where the inequality is the second point of Lemma 4.1. Therefore every combination of boundary values derived from a smooth convex function can be modelled by a parametric function. Since the ratio of p and q appears to have a deeper meaning, we give it a name of its own.

Definition 4.1 *The discriminant function of the parametric function g_P of a distinct parameter constellation is denoted by $d(\alpha, \beta)$ and defined as*

$$\begin{aligned} d(\alpha, \beta) &= \frac{p(\alpha, \beta)}{q(\alpha, \beta)} \\ &= \frac{g(1) - \frac{a}{(\alpha+1)(\alpha+2)} - \frac{b}{\beta+2}}{g'(1) - \frac{a}{\alpha+1} - \frac{b}{\beta+1}}. \end{aligned}$$

The following lemma holds.

Lemma 4.2 *For every twice differentiable convex function g satisfying the boundary conditions (4.1) – (4.6), there is a parameter constellation such that g_P replicates the boundary values of g, g', and g''.*

The proof of the Lemma is given by setting α, β large enough to be close to the limes. This is an existence result. But for the practical implementation we need a concise construction of parameter values that replicates the desired boundary values.

A first approach to an algorithm of this kind is to set $a = b = 0$, i.e. to set the second order derivative $y_0'' = y_1'' = 0$. This leads to a less complicated situation, yielding the following corollary.

Corollary 4.1 *Let g be as stated in the Lemma before and $g''(0) = g''(1) = 0$. Then the parametric function g_P with the below parameter setting replicates the boundary values. The parameters are continuous in the prescribed boundary values $g(1), g'(1)$. Set $a, b, \alpha, \beta = 0$ and,*

- *if*
$$\frac{g(1)}{g'(1)} < \frac{1}{2},$$
set
$$\begin{aligned} c &= g'(1)\left(\frac{2g(1) - 3g'(1)}{g(1)} + 1\right)\left(\frac{2g(1) - 3g'(1)}{g(1)} + 2\right) \\ \gamma &= 1 \\ \delta &= \frac{2g(1) - 3g'(1)}{g(1)}, \end{aligned}$$

- *if*
$$\frac{g(1)}{g'(1)} \geq \frac{1}{2},$$
set
$$\begin{aligned} c &= g'(1)\left(\frac{g'(1) - 3g(1)}{g(1) - g'(1)} + 1\right)\left(\frac{g'(1) - 3g(1)}{g(1) - g'(1)} + 2\right) \\ \gamma &= \frac{g'(1) - 3g(1)}{g(1) - g'(1)} \\ \delta &= 1. \end{aligned}$$

Proof of corollary 4.1: The proof is given above since the equations are deduced from (4.20) and (4.22). The only thing worth mentioning is that the ranges (4.19) and (4.21) overlap. We decided to separate them at $1/2$ for the following reason. If the discriminant function satisfies

$$d(0,0) = \frac{1}{2},$$

both γ and δ must be 1, which means both methods yield the same parameter values. This is a natural splitting point which makes our parameter decision continuous in the boundary values. □

There is an interesting feature that does not have any practical relevance but is a little pleasing and gives rise to a more distinct feeling that some deeper property of the convex parametric function is expressed by the discriminant function. Suppose we have two parameter constellations γ_1, δ_1 and γ_2, δ_2 which are mirrored, i.e.

$$\gamma_1 = \delta_2 = 1, \gamma_2 = \delta_1.$$

How does this affect the two discriminants d_1, d_2 of the constellations? Since $\gamma_1 = 1$ we have

$$d_1 = \frac{p_1}{q_1} \in (0, 2/3]$$

and

$$\delta_1 = \frac{2q_1 - 3p_1}{p_1}.$$

On the other hand, $\delta_2 = 1$ implies

$$d_2 = \frac{p_2}{q_2} \in (1/3, 1]$$

and

$$\gamma_2 = \frac{q_2 - 3p_2}{p_2 - q_2}.$$

Combining the two parameter expressions we get

$$\frac{2q_1 - 3p_1}{p_1} = \frac{q_2 - 3p_2}{p_2 - q_2}$$

which can be written equivalently as

$$\frac{p_1}{q_1} = \frac{q_2 - p_2}{q_2}$$

or

$$d_1 = 1 - d_2$$

which is a nice feature. Note that this a generalization of continuity at the point where the discriminant function is $1/2$ showing that $1/2$ is the only split point possible to have continuity in the boundary values.

Now we try to extend this construction for $a \neq 0$ and $b = 0$. From the restrictions

$$\begin{aligned} \alpha &\geq 0 \\ p(\alpha, \beta) &\geq 0 \\ q(\alpha, \beta) &> 0 \\ d(\alpha, \beta) &< 1, \end{aligned}$$

we easily deduce by rather tedious standard calculations

$$\begin{aligned}
\alpha &\geq 0 \\
\alpha &\geq \sqrt{\frac{a}{g(1)} + \frac{1}{4}} - \frac{3}{2} \\
\alpha &> \frac{a}{g'(1)} - 1 \\
\alpha &> \frac{a}{g'(1) - g(1)} - 2
\end{aligned}$$

as sufficient conditions for the parameter α. Remember that $a = g''(1)$ is fixed by the boundary conditions. Thus if we want a permissible parameter value for α, we can get it in a reliable way by

$$\alpha = w \cdot \max\left\{0, \sqrt{\frac{a}{g(1)} + \frac{1}{4}} - \frac{3}{2}, \frac{a}{g'(1)} - 1, \frac{a}{g'(1) - g(1)} - 2\right\} \tag{4.23}$$

for some $w > 1$ to assure the strict inequalities to be satisfied. In practice, we may use $c = 1.5$ to have a secure distance to possible singularities. Obviously, only if the $\max$ vanishes w cannot assure the strict inequalities. But in this case every positive value of α is permissible and we may choose $\alpha = 1$ for simplicity.

The same procedure can be imposed on the case $b \neq 0$, $a = 0$, leading to the somewhat symmetrical parameter choice

$$\beta = w \cdot \max\left\{0, \sqrt{\frac{b}{g'(1) - g(1)} + \frac{1}{4}} - \frac{3}{2}, \frac{b}{g'(1)} - 1, \frac{b}{g(1)} - 2\right\}. \tag{4.24}$$

Again, in the case where the right hand side of the above is 0, we may set $\beta = 1$. Now it is easy to propose a permissible parameter choice if $a, b \neq 0$. We just have to notice that we can choose say α first according to (4.23) and then β according to (4.24) by replacing

$$g(1) \leftarrow g(1) - \frac{a}{(\alpha + 1)(\alpha + 2)}$$

and

$$g'(1) \leftarrow g'(1) - \frac{a}{\alpha + 1}.$$

Then, we can apply the parameter choice method of Corollary 4.1 by the final replacements

$$g(1) \leftarrow g(1) - \frac{a}{(\alpha + 1)(\alpha + 2)} - \frac{b}{\beta + 2}$$

and

$$g'(1) \leftarrow g'(1) - \frac{a}{\alpha+1} - \frac{b}{\beta+1}.$$

This choice makes it more precise what we stated as 'close to the limit' of $\alpha, \beta \to \infty$. We derived a construction of a distinct parameter choice and so made the existence results of Lemma 4.2 computable. Unfortunately, the given construction is not continuous in the given boundary values any more since we set $\alpha, \beta \leftarrow 1$ if the $\max$ terms vanish. This case can actually happen.

Example 4.2 *Let the boundary values be*

$$\begin{aligned} g(1) &= 1 \\ g'(1) &= 2 \\ g''(0) &= 0 \\ g''(1) &= 2. \end{aligned}$$

Then,

$$\max\left\{0, \sqrt{\frac{a}{g(1)} + \frac{1}{4}} - \frac{3}{2}, \frac{a}{g'(1)} - 1, \frac{a}{g'(1) - g(1)} - 2\right\} = 0$$

and $\alpha = 1$ *according to (4.23). Note that* $\alpha = 0$ *results in* $q(\alpha) = 0$ *and thus an infinite discriminant function* d. *But the limes*

$$\lim_{a \uparrow 2} \alpha(a, g(1))$$

of α *as a function of* $g(1)$ *and* $a = y_1''$ *is* 0 *which is not permissible. Thus the parameter constellation is not continuous in* y_1''.

Surely this discontinuity is not desirable but this deficiency can be cured easily: we have to set the parameter in a secure distance from 0 to avoid singularities, see Corollary 4.2. We compress the results of the above calculations in the following theorem which is the most important fact of this section.

Theorem 4.1 *Let* $g : [0,1] \to \mathbb{R}$ *be a twice differentiable strictly convex function with boundary values*

$$g(0) = g'(0) = 0$$

and $g''(0), g(1), g'(1), g''(1)$. *Then there is a parameter choice such that the parametric function* $g_P(t)$ *according to (4.7) replicates these boundary values. A possible choice* *for the parameters is*

$$\begin{aligned}
a &= g''(1) \\
b &= g''(0) \\
\alpha &= w \cdot \max\left\{0, \sqrt{\frac{a}{g(1)} + \frac{1}{4}} - \frac{3}{2}, \frac{a}{g'(1)} - 1, \frac{a}{g'(1) - g(1)} - 2\right\} \qquad (4.25) \\
\textit{or} &= 1 \textit{ if the above is } 0 \\
\beta &= w \cdot \max\Bigg\{0, \sqrt{\frac{b}{g'(1) - g(1) - \frac{a}{(\alpha+1)(\alpha+2)}} + \frac{1}{4}} - \frac{3}{2}, \\
&\quad \frac{b}{g'(1) - \frac{a}{\alpha+1}} - 1, \frac{b}{g(1) - \frac{a}{(\alpha+1)(\alpha+2)}} - 2\Bigg\} \qquad (4.26) \\
\textit{or} &= 1 \textit{ if the above is } 0,
\end{aligned}$$

and,

- *if*

  $$d(\alpha, \beta) < \frac{1}{2},$$

  set

  $$\begin{aligned}
  c &= q(\alpha,\beta)\left(\frac{2p(\alpha,\beta) - 3q(\alpha,\beta)}{p(\alpha,\beta)} + 1\right)\left(\frac{2p(\alpha,\beta) - 3q(\alpha,\beta)}{p(\alpha,\beta)} + 2\right) \\
  \gamma &= 1 \\
  \delta &= \frac{2p(\alpha,\beta) - 3q(\alpha,\beta)}{p(\alpha,\beta)}.
  \end{aligned}$$

- *If*

  $$d(\alpha, \beta) \geq \frac{1}{2},$$

  set

  $$\begin{aligned}
  c &= q(\alpha,\beta)\left(\frac{q(\alpha,\beta) - 3p(\alpha,\beta)}{p(\alpha,\beta) - q(\alpha,\beta)} + 1\right)\left(\frac{q(\alpha,\beta) - 3p(\alpha,\beta)}{p(\alpha,\beta) - q(\alpha,\beta)} + 2\right) \\
  \gamma &= \frac{q(\alpha,\beta) - 3p(\alpha,\beta)}{p(\alpha,\beta) - q(\alpha,\beta)} \\
  \delta &= 1,
  \end{aligned}$$

with $w > 1$ and p, q according to (4.17) and (4.18).

Now we have a constructive parameter algorithm for every possible boundary values realized by a convex function. There is some arbitrage and discontinuity in this setting, but experience shows that the visual form of the parametric function does not depend too heavily on the distinct parameters we use.

To cure the deficiency of discontinuity by brute force, we might set

$$\alpha = w \cdot \max\left\{\varepsilon, \sqrt{\frac{a}{g(1)} + \frac{1}{4}} - \frac{3}{2}, \frac{a}{g'(1)} - 1, \frac{a}{g'(1) - g(1)} - 2\right\}$$

for an arbitrary $\varepsilon > 0$ instead of (4.25) and similiar for β in (4.26). We might set $\varepsilon = 1/w$ to achieve an integer exponent α, β in the case ot the $\max$ taking the first of the four values for simplicity of the resulting integrals. We state the possibility of continuous parameter choice in the next Corollary.

Corollary 4.2 *The parameter choice of Theorem 4.1 can be made continuous in the boundary values by substituting the* 0 *in the* $\max$ *terms of (4.25) and (4.26) by any positive number.*

4.3 String polishing

With the help of the last section, we are now able to give a local reconstruction of a linear piece of the taut string by a parametric convex resp. concave function with given boundary values for the function itself and its first two derivatives. The problem of polishing the string between the bounds thus reduces to the problem of devising suitable boundary derivatives that allow the convex/concave reconstruction. We give several more or less adhoc methods. As a matter of fact, this section does not reveal so much structure and you might see it as a bunch of ideas we followed which are at best heuristic and at worst simple thumb rules.

Proposal 1: Let $K = \{(k_1^x, k_1^y), \ldots, (k_l^x, k_l^y)\}$ be the set of edges (i.e. the touching points of the string with the bounds) of the string between the bounds. Then we propose the following values for the first order derivative:

- For $2 \leq j \leq l-1$, set

$$m_j = \frac{1}{2}\left(\frac{k_j^y - k_{j-1}^y}{k_j^x - k_{j-1}^x} + \frac{k_j^y - k_{j+1}^y}{k_j^x - k_{j+1}^x}\right),$$

  which is the mean of the slope of the string to the left and to the right.

- Set
$$m_0 = m_1 = \frac{k_1^y - k_0^y}{k_1^x - k_0^x},$$
and
$$m_{l-1} = m_l = \frac{k_{l-1}^y - k_l^y}{k_{l-1}^x - k_l^x}.$$
This implicates that the first and last piece of the string is replicated and that we export an infinitesimal linear trend in areas of no data. We refer to Chapter 3 where we also introduced vanishing second order derivative at the boundary.

- Set
$$m_j = 0$$
whenever the j–th knot is a local extreme of the string.

It is easy to see that the prescription of the first order derivative is permissible in the sense of monotonicity: There is a function interpolating the set of knots K which has first order derivatives m_j in the knots and possesses the same monotonicity as the string.

We also have to propose values for the second order derivative. Thus we want to introduce two different policies:

- Each knot carries a vanishing second order derivative.

- Each knot has second order derivatives which are the obvious weighted second order differences of the knots left and right,
$$\frac{k_j^y - k_{j-1}^y}{k_j^x - k_{j-1}^x} - \frac{k_j^y - k_{j+1}^y}{k_j^x - k_{j+1}^x}.$$
For inflection intervals, i.e. intervals having second order differences of different sign, the values are set to zero. Boundary intervals are treated as inflection intervals.

Remark 4.1

1. *Note that the piecewise parametric function inherits the monotonicity behaviour from the string because we used the shape preserving representation. Even more, the function has the same intervals of convexity and concaveity. Clearly, the bounds constraint from which we constructed the string cannot be satisfied in general. But since the polished string does not distance itself too far from the bounds, they might be called satisfied in spirit. Because the polished string is always less than the string in areas of convexity and greater in areas of concaveity, lower resp. upper bound restrictions are fulfilled in those intervals.*

2. *Obviously, we imagine conflicts between e.g. local extrema and inflection points where Proposal 1 is not able to give permissible values for the derivatives. It can specifically happen that there is only one linear piece of the string between two extrema. Because such an interval is an inflection interval, we would have to set both second order boundary derivatives to* 0. *This conflicts with the vanishing first order derivative due to the extrema on the left and right. Thus the prescription of vanishing first* and *second order derivatives at the boundary of this interval does not make any sense. Treating all cases separately would complicate the computer code without need because in nearly all constructable situations like this, we do not have any better choice than to take the straight line in the conflict interval. Our computer code is thus designed to give back the straight line (and a warning) if there is some conflict such that a convex/concave reconstruction is impossible.*

3. *The obvious approach of devising the second order differences for the prescription of the second derivatives is often outperformed by the easier way of assigning zeros which is demonstrated by Figure 4.3. We think that the reason for this is that the parametric function has more freedom to get away from the string if its second derivative is zero. An illustration can be seen in Figure 4.2.*

We want to present yet another method of devising permissible second order derivative values. Let for the moment the left hand second order derivative be fixed. Then we have some freedom in choosing the value $g''(1)$ on the right hand side. It seems plausible in our context to choose $g''(1)$ as the value minimizing the smoothness

$$S(g) = \int_0^1 (g''(t))^2 dt$$

of the parametric function. For the parametric function g_P we can calculate the integral S explicitely,

$$\begin{aligned} S(g_P) &= \frac{a}{2\alpha+1} + \frac{b}{2\beta+1} \\ &\quad +2abB(\alpha+1,\beta+1) + 2acB(\alpha+\delta+1,\gamma+1) \\ &\quad +2bcB(\gamma+\beta+1,\delta+1) + c^2B(2\gamma+1,2\delta+1). \end{aligned} \tag{4.27}$$

Note that all but the last term vanish in the case of zero second order derivatives which corresponds to $a = b = 0$. This may be seen as a theoretical justification why the pictures of the functions look particularly smooth if the second order boundary derivatives disappear. Since we can approximate the Beta function B, we can evaluate the right hand side of (4.27) quickly for different values and pick the one out which gives the smallest value. This is brute force minimizing the roughness in the second order derivative. To be sure to

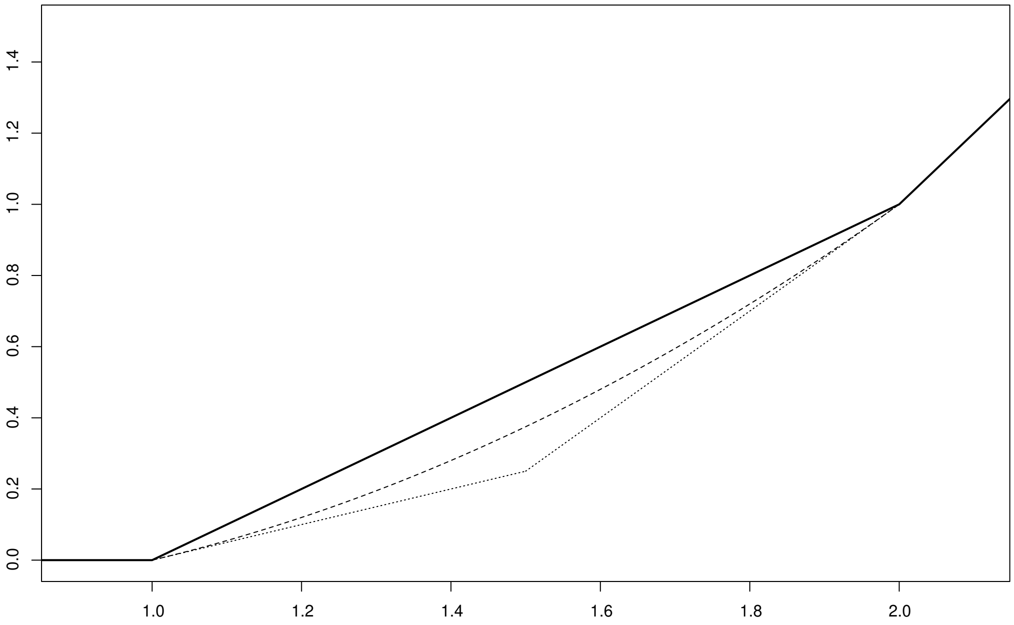

Figure 4.2: Space for the parametric substitute of the straight line if the second derivatives are 0 (dotted) and if they are second order differences (dashed). The area of the dotted line is greater and thus the parametric function can leave the taut string and its edges more easily.

produce permissible values for $g''(1)$ by the minimizing step, we only test a collection of, say, ten permissible values.

One might think of direct calculation of the minimum of $S(g_P)$ by differentiation by a, but this is not very promising since $\alpha, \beta, \gamma, \delta$ and c are somewhat complicated functions of a which are not differentiable in general. Thus we arrive at the following.

Proposal 2: Choose the first order derivative values according to some rule, e.g. Proposal 1. Choose the second order derivative to vanish at the left boundary. Now start the procedure in the first interval.

- Minimize $S(g_P)$ over a test set of permissible values for the second order derivative in the right end of the interval under examination.
- Pick out the argument of the minimum and fix it.
- Iterate, i.e. take the next interval. The second order derivative on the left is already known: it is what we calculated in the last step.

Some intervals such as the end of the data have already fixed second order derivatives to the right. We have to treat them separately. For more details, see the Appendix A.7 where the exact computer code is printed out with detailed comments.

Now we have all our necessary tools at hand and merge it into our algorithm.

Definition 4.2 *We give a description of what we call the string polishing algorithm in future.*

- *Given a string through the bounds, assign permissible values for the derivatives at all knots of the string. This can happen e.g. by use of the policies given above.*
- *Now look at each single interval between two neighbouring knots. Transfer it to standard position. Calculate a parameter fit for the parametric family g_P according to Theorem (4.1). Transfer it back to the original position. Evaluate at the design mesh.*
- *Merge the pieces to one vector.*

The resulting vector will be referred to as polished string.

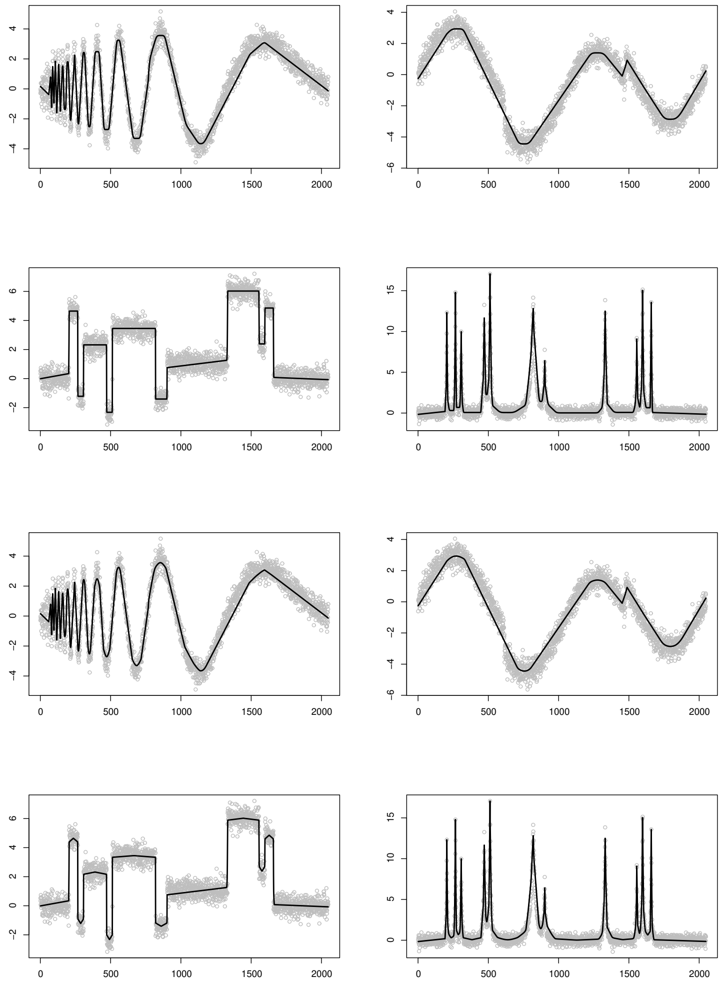

Figure 4.3: The Donoho et al. test signals and the polished string. Bounds were constructed using the Davies/Kovac string estimator. The second order derivatives were set to second order differences in the upper panel, to zero in the lower panel. Note that vanishing second order derivative enables straight lines for the Blocks signal. The polished strings seem to leave the data sometimes. This can be improved by the construction of bounds that are closer to the data.

Example 4.3 *We demonstrate the performance of the algorithm for the Donoho et al. test signals. The greatest advantage of the polished string is that it is very quickly calculable. The number of parameter choices and boundary derivatives to devise is equal to the number of linear pieces of the string. Each evaluation of the polished string once it is calculated costs at most a complexity of calculation which is of the same order as this number of linear pieces. Thus, even for huge data sets, it is possible to perform the calculation of the polished string online once the bounds are constructed from the data. Note that the polished string clings to the original taut string very close, but the visual impression of the smoothed curves is much better than the piecewise linear string which is the fabric of the smoothed version. Also interesting to note is that in the case of the Blocks signal, where smoothing does not seem to make very much sense, the polished string almost keeps the piecewise constant structure of the signal and does not smooth away the edges. We have tested Proposal 2 with the four Donoho et al. Test Signals. The results were that in the utmost situations, the optimum was attained by vanishing second order derivatives. Thus the curves very much looked like those we got with this simpler method.*

It seems that we cannot optimize the results of the polished string algorithm too much by devising the second order boundary derivative. Perhaps we would have more impact on the estimator by optimizing the first order derivatives. But it is not so easy devising permissible values then. We have the strong impression that further improvement of the method cannot be expected. If we are still left unsatisfied with the performance of the polished string, we have to choose another smoothing method, perhaps one of those we described in the last chapter.

Note that the polished string algorithm can as well be used as isogeometric interpolation algorithm by setting the bounds to the data points. The polished string will then interpolate the data, retaining the montonicity and convexity/concaveity of the interpolation data. The resulting interpolation technique is easy and fast calculable and perhaps competetitive to other existing methods in performance. We leave this for future examination.

Chapter 5

Conclusion and Future Tasks

Die Wissenschaft kann in ebenderselben Weise
den menschlichen Geist ergötzen wie die Kunst

Georg Cantor

Concluding theses:

1. The modality structure is an important feature of the data. Thus a non-parametric model should be constructed from the data under control of the modality.

2. Smoother models are often considered to be simpler and thus better compared to less smooth models. Smoothness is a user demand in particular for non–statisticians.

3. The visual impression of smoothnes is well met by the mathematical smoothness measure of integral of squared second order derivative.

4. Bounds and monotonicity constraints constructed from modality constrained but piecewise constant estimators such as the taut string and the run method resemble the modality structure very well.

5. The smoothest curve between the bounds is uniquely defined and can be shown to have a certain characteriztion.

6. The calculation of the smoothest curve between the bounds having the correct monotonicity behaviour by quadratic programming is memory– and time consumptive.

7. The approximation of the smoothest curve between the bounds having the correct monotonicity behaviour by an iterative procedure (modified QSOR) is time consumptive.

8. The convergence acceleration of QSOR by some hybrid method, i.e. using the characterization of the unique solution of the maximum roughness problem is instable, but great in case of success.

9. The taut string between the bounds is an easy–to–calculate and at least continuous model.

10. The polished string inherits the good property of the taut string but additionally is smooth.

Future tasks:

1. What happens if we try to incorporate the special diagonal structure of the constraints and smoothness matrix of the minimum roughness problem into a modern quadratic programming algorithm, in particular the one of [Goldfarb and Liu (1993)] which has at worst a complexity of calculations of order n^3)? We conjecture that this can be improved to n^2 in our situation.

2. How could the concept of modality be generalized for higher dimensions? Can we construct modality controlled estimators? In the case of two dimensions and regular design (which is in fact image processing), there are approaches based on projections.

3. How can we construct algorithms for the calculation of the additional knots of inequality constrained cubic splines, e.g. for the case of monotonic interpolation?

4. If we do not have enough time to wait for QSOR to converge and if we are not pleased by the result of the polished string, what different modality constrained smoothing methods can be applied?

Appendix A

Code

The complete code can be downloaded from the statistics group web site

`http://wwwstat.mathematik.uni-essen.de/Software.html`

The code is contained in an add–on library for R. A detailled installation guide is included. The R statistical language can be downloaded from the STATLIB archive.

For smoothing under constraints by use of our software, the procedere should be

- extract bounds from the data by use of the `create.bounds` module.
- find a smooth function connected to the bounds by one of the three modules `mqsor` (for modified QSOR), `hybrid` (for the hybrid spline–QSOR method), or `fsmooth` (for the polished string).

Since the software will certainly undergo several modifications in future, we decided to print out the complete code used for this thesis. Additionally, the coding is a eminent part of this dissertation and thus the code deserves being listed in the appendix.

A.1 Code for Run Bounds and Taut strings

The code is provided by Davies and Kovac. The function calls are `finalreg`

which is a R module and

`runbnd`

which is a FORTRAN subroutine we linked dynamically to R to get it work under a graphic display. Also, we used the C routine `bindfaden` from Kovac. We do not give the source code of those here because they do not reveal further insight in what we did in this thesis and are not part of our work, but they are of course contained in our software package which is downloadable from our web site.

A.2 The R Bounds Creator

We now present the main programs of bounds creating which are self–commenting.

```
################################################################################
# creates a bounds object from data.
# parameters.  squeeze.mode.par gives
# method="run"
# squeeze.mode="asymptotic" 0<alpha<1 for quantile
#      "runlength" max. allowed runlength
#       "modal"            number of local extrema desired
# (if possible)
# in all variants, the runlength is decreased until a further extremum
# occurs.
#
# method="string"
# squeeze.mode="local" threshold factor
# squeeze.mode="global" -"-
#
# an extremum policy extpol for choosing distinctive
# position of the extrema is required.
# This is done by pick.modes. Possible policies are "interactive",
# "roof","quadratic" and "unimodal". "none" gives back raw bounds.
#
# also, a policy for boundary treatment bdrpol is desired.
# Possible choices:
# method="run": "median","linreg","interactive".
# method="string: "mean","linreg","interactive".
# "none" gives back raw bounds.
#
# winsorize=T winsorizes the data before picking the extremum which is
# recommended under use of the run method.
#
#  verbose=T gives screenshots at run time.

create.bounds<-function(data, method="string", squeeze.mode="local",
      squeeze.mode.par=2.5, extpol="interactive", bdrpol="interactive",
      winsorize=T, verbose=F){
```

```
n<-length(data)

# method switch

method<-match(method,c("string","run"))

if (is.na(method)) stop("Invalid construction method")

if (method==1) {

########################################

 squeeze.mod<-match(squeeze.mode,c("global","local"))

 if (is.na(squeeze.mod)) stop("Invalid squeeze.mode for string method")
 squeeze.mod<-(squeeze.mod==2)

 # construction of string bounds

 tmp<-finalreg(1:n,data,data,localsqueeze=squeeze.mod)

 # simultaneous confidence bands

 kn<-length(tmp$knotsind)-1

 # calculation of sigma

 sigma<-mad(data-tmp$y) # 1.48 already included
 # sigma<-tmp$sigma

 # prepare the bounds half width
 # 3.0 proved a good compromise but is somewhat arbitrary

 h<-3.0*sigma*sqrt(2*log(as.double(kn)))/
      sqrt(as.double(tmp$knotsind[2:(kn+1)]-tmp$knotsind[1:kn]))
 h<-approx(tmp$knotsind,c(h,h[length(h)]),c(1:n),method="constant")$y
 up<-tmp$y+h
 lo<-tmp$y-h

 if (verbose){
plot(data,col='grey',xlab='',ylab='')
lines(up)
lines(lo)
title('Raw bounds')
readline()
 }

 # find extrema

 tmp$knotsind[kn+1]<-n
 tmp$knotsy<-tmp$y[tmp$knotsind]
 locl<-integer(0)
 locr<-integer(0)
 for (i in 2:kn){
if ((tmp$knotsy[i-1]-tmp$knotsy[i])*(tmp$knotsy[i]-tmp$knotsy[i+1])<0){
locl<-c(locl,tmp$knotsind[i])
locr<-c(locr,tmp$knotsind[i+1]-1)
}
 }

 bounds<-list(lower=lo,upper=up,num.ext=length(locl),
      loc.ext=list(left=locl,right=locr))

 # fix extrema
```

```
 if (bounds$num.ext>0)
  bounds<-pick.modes(data,bounds,policy=extpol,method="string")

 # fix boundary values

 bounds<-fix.boundary(data,bounds,bdrpol,"string")

########################################

} else {

 # construction of run bounds

 # change defaults.

 if (squeeze.mode=="local") squeeze.mode<-"asymptotic"
 if (squeeze.mode.par==2.5) squeeze.mode.par<-0.5

 strmode<-squeeze.mode
 squeeze.mode<-match(squeeze.mode,c("asymptotic",
"runlength","modal"))
 if (is.na(squeeze.mode)) stop("Invalid squeeze.mode for run method")

 if (squeeze.mode==1) {

####################

  # check alpha

  alpha<-squeeze.mode.par
  if ((alpha<=0)||(alpha>=1))
       stop("Invalid squeeze.mode.par for method=run, squeeze.mode=',
            strmode,'\n must be between 0 and 1")

  # perform run method with max. run length quantile alpha

  tmp<-.Fortran("runbnd",as.double(data),u=double(n),l=double(n),
double(n),double(n),double(n),double(n),as.integer(n),
locl=integer(n),locr=integer(n),as.integer(n),as.double(alpha),
nr=integer(1))
  maxrun<-tmp$nr

  # kill zeros in locs

  tmp$locl<-tmp$locl[tmp$locl!=0]
  tmp$locr<-tmp$locr[tmp$locr!=0]
  nex<-length(tmp$locl)

  }
##########

  if (squeeze.mode==2) {

  nr<-as.integer(squeeze.mode.par)
  if ((nr<=1)||(nr>=n-1))
       stop("Invalid squeeze.mode.par for method=run, squeeze.mode=',
            strmode,'\n must be between 2 and length of data-1")

  # perform run method with max. run length quantile alpha

  tmp<-.Fortran("runbnd",as.double(data),u=double(n),l=double(n),
double(n),double(n),double(n),double(n),as.integer(n),
locl=integer(n),locr=integer(n),as.integer(n),as.double(0),
nr=nr)
```

```
  # kill zeros

  tmp$locl<-tmp$locl[tmp$locl!=0]
  tmp$locr<-tmp$locr[tmp$locr!=0]
  nex<-length(tmp$locl)
  maxrun<-tmp$nr

  }

 if (squeeze.mode==3) {

####################

  nex<-as.integer(squeeze.mode.par)

  nbl<-as.integer(2)
  nbr<-as.integer(n-1)

  while (nbl+1<nbr) {

  nbm<-as.integer((nbl+nbr)/2)

  # perform run method

  tmpl<-.Fortran("runbnd",as.double(data),u=double(n),l=double(n),
double(n),double(n),double(n),double(n),as.integer(n),
locl=integer(n),locr=integer(n),as.integer(n),as.double(0),
nr=as.integer(nbl))$locl
  tmpr<-.Fortran("runbnd",as.double(data),u=double(n),l=double(n),
double(n),double(n),double(n),double(n),as.integer(n),
locl=integer(n),locr=integer(n),as.integer(n),as.double(0),
nr=as.integer(nbr))$locl
  tmpm<-.Fortran("runbnd",as.double(data),u=double(n),l=double(n),
double(n),double(n),double(n),double(n),as.integer(n),
locl=integer(n),locr=integer(n),as.integer(n),as.double(0),
nr=as.integer(nbm))$locl

  # nex contained in one?

  nrl<-sum(tmpl!=0)
  nrr<-sum(tmpr!=0)
  nrm<-sum(tmpm!=0)

  if ((nrl>=nex)&&(nex>=nrm)) {
    nbr<-nbm
  } else if ((nrm>=nex)&&(nex>=nrr)) {
    nbl<-nbm
  } else {
    stop(cat("Modality confusion:",nex,
         " extrema are not consistent with the data"))
  }

  }

 # definitive calculation of bounds.

 tmp<-.Fortran("runbnd",as.double(data),u=double(n),l=double(n),
double(n),double(n),double(n),double(n),as.integer(n),
locl=integer(n),locr=integer(n),as.integer(n),as.double(0),
nr=as.integer(nbr))
 maxrun<-tmp$nr
 nex<-sum(tmp$locl!=0)
 }

####################
```

```
  mx<-2*rep(max(data),n)
  mn<-2*rep(min(data),n)

  lo<-pmax(mn,tmp$l)
  up<-pmin(mx,tmp$u)

  if (verbose) {
plot(data,col='grey')
lines(lo)
lines(up)
lines(0.5*(up+lo),col='red')
title('Raw bounds')
readline()
  }

  bounds<-list(lower=lo,upper=up,num.ext=nex,loc.ext=list(left=tmp$locl,
    right=tmp$locr),maxrun=maxrun)

  # winsorize if desired

  if (winsorize) data<-winsor(data,bounds)

  # find & fix extrema if any

  if ((bounds$num.ext>0)&&(extpol!="none"))
       bounds<-pick.modes(data,bounds,extpol,"run")

  # boundary values

  bounds<-fix.boundary(data,bounds,bdrpol,"run")

}
# a last verbose output.

 if (verbose) {
plot(data,col='grey',xlab='',ylab='')
lines(bds$lower)
lines(bds$upper)
readline()
 }

bounds
}

############################################################################
# some helpers

iso.mark<-function(lower,upper,loc){

# marks the isotonicity intervals
# in iso: 1 isotone -1 antitone

n<-length(lower)
iso<-rep(-1,n)

for (i in 2*(1:(length(loc)/2))) {
left<-loc[i-1]
right<-loc[i]

iso[left:right]<-iso[left:right]*(-1)
}

# first interval iso or anti?
```

```
if (lower[loc[1]]>lower[loc[2]]) iso<-iso*(-1)
list(iso=iso,loc=loc)}

###############################################################################
#
# makes bounds iso/antitonic according to iso.mark

iso.make<-function(lower,upper,loc){

n<-length(lower)

iso<-iso.mark(lower,upper,loc)

for (i in 2:length(iso$loc)){
left<-iso$loc[i-1]
right<-iso$loc[i]-1
what<-(iso$iso[(left+right)/2]==1)
n.rel<-right-left+1

# FORTRAN calls for the sake of computation speed

if (what){

tml<-.Fortran('lowiso',
erg=as.double(lower[left:right]),
as.integer(n.rel))
lower[left:right]<-tml$erg

tmu<-.Fortran('uppiso',
erg=as.double(upper[(left+1):(right+1)]),as.integer(n.rel))
upper[(left+1):(right+1)]<-tmu$erg

} else {
tml<-.Fortran('lowanti',
erg=as.double(lower[(left+1):(right+1)]),as.integer(n.rel))
lower[(left+1):(right+1)]<-tml$erg

tmu<-.Fortran('uppanti',
erg=as.double(upper[left:right]),
as.integer(n.rel))
upper[left:right]<-tmu$erg

}

}
list(lower=lower,upper=upper,loc=iso$loc[2:(length(iso$loc)-1)],iso=iso$iso)}

###############################################################################
#
# pick.modes picks out extrema position and heights.
#
# extremum policies: "interactive","roof","quadratic","unimodal"
# for method="string", "mean" of the bounds is possible, too

pick.modes<-function(data,bounds,policy="interactive",method="run"){

pol<-match(policy,c("interactive","roof","quadratic","unimodal","mean"))

if ((is.na(pol))||(pol<1)||(pol>5)) stop(cat("Unknown mode policy",
       pol,))
if ((pol==5)&&(method=="run")) stop("mean is not allowed for method=run")

ind.ext<-integer(bounds$num.ext)

##########
```

```
if (pol==1) {

 for (i in 1:bounds$num.ext){

 left<-bounds$loc.ext$left[i]
 right<-bounds$loc.ext$right[i]

 # bounds for the height

 if (method=="run") {
  upper<-min(min(bounds$upper[left:right]),max(data[left:right]))
  lower<-max(max(bounds$lower[left:right]),min(data[left:right]))
 } else {
  upper<-bounds$upper[(left+right)/2]
  lower<-bounds$lower[(left+right)/2]
 }

 # get defaults

tr<-roof(data[left:right])
xr<-tr$x+left-1
yr<-max(min(tr$y,upper),lower)

tq<-quadratic(c(left:right),data[left:right])
xq<-tq$x
yq<-max(min(tq$y,upper),lower)

tumax<-unimodal(data[left:right],T)
tumin<-unimodal(data[left:right],F)

if (tumax$score<=tumin$score) tu<-tumax else tu<-tumin
xu<-tu$x+left-1
yu<-max(min(tu$y,upper),lower)

 # initialize plot

 plot(data,t='n')
 lines(pmax(bounds$lower,min(data)))
 lines(pmin(bounds$upper,max(data)))
 polygon(c(left,right,right,left),c(lower,lower,upper,upper),col="lightgreen")
 points(data,col='grey')
 title("Choosing the extremum position interactively")

 # mark other policies

 text(xr,yr,"r")
 text(xq,yq,"q")
 text(xu,yu,"u")

 perm<-F

 while (!perm) {
  perm<-T
  # read out mouse click position
  pos<-locator(1)
  ind<-as.integer(pos$x)
  height<-pos$y

  if ((ind<left)||(ind>right)) perm<-F
  if ((height<lower)||(height>upper)) perm<-F
 }

 ind.ext[i]<-ind
```

```
 # set bounds to picked extremum

 bounds$upper[ind]<-height
 bounds$lower[ind]<-height


 }

}

##########

if (pol==2) {

# roof

 for (i in 1:bounds$num.ext){

 left<-bounds$loc.ext$left[i]
 right<-bounds$loc.ext$right[i]

# bounds for the height

 if (method=="run") {
  upper<-min(min(bounds$upper[left:right]),max(data[left:right]))
  lower<-max(max(bounds$lower[left:right]),min(data[left:right]))
 } else {
  upper<-bounds$upper[(left+right)/2]
  lower<-bounds$lower[(left+right)/2]
 }

 tmp<-roof(data[left:right])
 ind.ext[i]<-tmp$x+left-1
 height<-max(min(tmp$y,upper),lower)

 # set bounds to picked extremum

 bounds$upper[ind.ext[i]]<-height
 bounds$lower[ind.ext[i]]<-height


 }

}

##########

if (pol==3) {

# quadratic
 for (i in 1:bounds$num.ext){

 left<-bounds$loc.ext$left[i]
 right<-bounds$loc.ext$right[i]

# bounds for the height

 if (method=="run") {
  upper<-min(min(bounds$upper[left:right]),max(data[left:right]))
  lower<-max(max(bounds$lower[left:right]),min(data[left:right]))
 } else {
  upper<-bounds$upper[(left+right)/2]
  lower<-bounds$lower[(left+right)/2]
 }
```

```
 tmp<-quadratic(c(left:right),data[left:right])
 ind.ext[i]<-tmp$x
 height<-max(min(tmp$y,upper),lower)

 # set bounds to picked extremum

 bounds$upper[ind.ext[i]]<-height
 bounds$lower[ind.ext[i]]<-height

 }
}

##########

if (pol==4) {

# least unimodal

 for (i in 1:bounds$num.ext){

 left<-bounds$loc.ext$left[i]
 right<-bounds$loc.ext$right[i]

  # bounds for the height

  if (method=="run") {
   upper<-min(min(bounds$upper[left:right]),max(data[left:right]))
   lower<-max(max(bounds$lower[left:right]),min(data[left:right]))
  } else {
   upper<-bounds$upper[(left+right)/2]
   lower<-bounds$lower[(left+right)/2]
  }

  # need to know if max or min
  # choose the one with lesser score

  tmpmax<-unimodal(data[left:right],T)
  tmpmin<-unimodal(data[left:right],F)

  if (tmpmax$score<=tmpmin$score) tmp<-tmpmax else tmp<-tmpmin
  ind.ext[i]<-tmp$x+left-1
  height<-max(min(tmp$y,upper),lower)

  # set bounds to picked extremum

  bounds$upper[ind.ext[i]]<-height
  bounds$lower[ind.ext[i]]<-height

 }
}

###############

if (pol==5) {

# mean
 for (i in 1:bounds$num.ext){

 left<-bounds$loc.ext$left[i]
 right<-bounds$loc.ext$right[i]

# bounds for the height

  upper<-bounds$upper[(left+right)/2]
  lower<-bounds$lower[(left+right)/2]
```

```
 height<-(upper+lower)/2
 ind.ext[i]<-as.integer(0.5*(left+right))

 # set bounds to picked extremum

 bounds$upper[ind.ext[i]]<-height
 bounds$lower[ind.ext[i]]<-height

 }
}

# iso make

tmp<-iso.make(bounds$lower,bounds$upper,c(1,ind.ext,length(data)))
bounds$lower<-tmp$lower
bounds$upper<-tmp$upper
bounds$loc<-tmp$loc

# a last plot in case of policy="interactive"

if (pol==1) {
 plot(data,col='grey')
 lines(bounds$lower)
 lines(bounds$upper)
 title("Chosen extremum positions")
}

bounds
}

################################################################################
#
# roof policy
#

roof<-function(y){

n<-length(y)
x<-c(1:n)

sl<-double(n)
sr<-double(n)
score<-double(n)

sl[1]<-1e30
sr[1]<-1e30
sr[n]<-1e30
sl[n]<-1e30

for (i in 2:(n-1)) {

sl[i]<-.Fortran("lsreg",as.double(x[1:i]),as.double(y[1:i]),as.integer(i),
                m=double(1),b=double(1),score=double(1))$score
sr[i]<-.Fortran("lsreg",as.double(x[i:n]),as.double(y[i:n]),as.integer(n-i+1),
                m=double(1),b=double(1),score=double(1))$score
score<-sl+sr

}

ind<-as.integer(min(c(1:n)[score==min(score)]))

# height calculation

t1<-.Fortran("lsreg",as.double(x[1:ind]),as.double(y[1:ind]),as.integer(ind),
```

```
             m=double(1),b=double(1),score=double(1))
t2<-.Fortran("lsreg",as.double(x[ind:n]),as.double(y[ind:n]),
             as.integer(n-ind+1),m=double(1),b=double(1),score=double(1))

y<-double(n)
y[1:ind]<-c(1:ind)*t1$m+t1$b
y[ind:n]<-c(ind:n)*t2$m+t2$b

if (t1$m>0) {
height<-max(t1$b+t1$m*ind,t2$b+t2$m*ind)
}else{
height<-min(t1$b+t1$m*ind,t2$b+t2$m*ind)
}
y[ind]<-height

list(x=ind,y=height,fit=y)

}

################################################################################
#
# quadratic L2 regression in the plane for x,y

quadratic<-function(x=c(1:n),y){

n<-length(y)

mx<-mean(x)
my<-mean(y)

z2<-sum((x-mx)**2)
z3<-sum((x-mx)**3)
z4<-sum((x-mx)**4)
w1<-sum((y-my)*(x-mx))
w2<-sum((y-my)*(x-mx)**2)

# calculate quadratic regression coeffs

a<-(n*z2*w2-n*z3*w1)/(-n*z3**2+n*z4*z2-z2**3)
b<-(w1-z3*a)/z2
ce<-(w2-z4*a-z3*b)/z2

y<-a*(x-mx)**2+b*(x-mx)+ce+mean(y)

# position and height of extremum

ind<-min(max(as.integer((2*a*mx-b)/(2*a)),min(x)),max(x))
height<-a*(ind-mx)**2+b*(ind-mx)+ce+mean(y)

list(x=ind,y=height,fit=y)
}

################################################################################
#
# L2 least unimodal policy

unimodal<-function(y,maxi=T){

n<-length(y)
x<-c(1:n)

sl<-double(n)
sr<-double(n)
score<-double(n)
sl[1]<-1e30
```

```
sr[1]<-1e30
sr[n]<-1e30
sl[n]<-1e30

for (i in 2:(n-1)) {

if (maxi){
 # calculate monotone estimator up to i

 tmp<-monsmo(y[1:i])

 # squared residuals

 sl[i]<-sum((tmp-y[1:i])**2)

 # calculate monotone estimator from i

 tmp<--monsmo(-y[i:n])
 sr[i]<-sum((tmp-y[i:n])**2)
} else {
 # calculate monotone estimator up to i

 tmp<--monsmo(-y[1:i])

 # squared residuals

 sl[i]<-sum((tmp-y[1:i])**2)

 # calculate monotone estimator from i

 tmp<-monsmo(y[i:n])
 sr[i]<-sum((tmp-y[i:n])**2)

}
}

score<-sl+sr

ind<-as.integer(min(c(1:n)[score==min(score)]))

# height calculation

if (maxi) {
t1<-monsmo(y[1:ind])
t2<--monsmo(-y[ind:n])
} else {
t1<--monsmo(-y[1:ind])
t2<-monsmo(y[ind:n])
}
y<-double(n)
y[1:ind]<-t1
y[ind:n]<-t2

if (maxi) {
height<-max(t1[ind],t2[1])
}else{
height<-min(t1[ind],t2[1])
}
y[ind]<-height

list(x=ind,y=height,fit=y,score=score[ind])

}

################################################################################
```

```
#
# winsorizing

winsor<-function(data,bounds,const=3){

n<-length(data)
maxim<-max(data)
minim<-min(data)
wdata<-data

for (i in 1:bounds$num.ext) {

 # boundary of interval

 left<-bounds$loc.ext$left[i]
 right<-bounds$loc.ext$right[i]

 # max or min?

 if (bounds$upper[as.integer((left+right)/2)]>maxim) maxi<-T else maxi<-F

 if (maxi) {

  # shift lower bound and calculate mad

  lo<-bounds$lower[left:right]+median(data[left:right])-
median(bounds$lower[left:right])
  bd<-const*mad(data[left:right]-lo,0,1)

  # pull back outlying data

  tmp<-pmax(pmin(data[left:right]-lo,rep(bd,right-left+1)),
rep(-bd,right-left+1))+lo

 } else {

  # shift upper bound and calculate mad

  up<-bounds$upper[left:right]+median(data[left:right])-
median(bounds$upper[left:right])
  bd<-const*mad(data[left:right]-up,0,1)

  # pull back outlying data

  tmp<-pmax(pmin(data[left:right]-up,rep(bd,right-left+1)),
rep(-bd,right-left+1))+up

 }
 # reset data
 wdata[left:right]<-tmp

 }
wdata
}

################################################################################
#
# boundary values
#
# policies: "median","linreg","interactive","mean"
# "median" only if method="run",
# "mean" only if method ="string"

fix.boundary<-function(data,bounds,policy="median",method="run"){
```

```
n<-length(data)

pol<-match(policy,c("median","linreg","interactive","mean"))

# regularity check

if (is.na(pol)) stop("Unknown boundary policy")
if ((pol==1)&&(method=="string")) stop("bdrpol=median only for run bounds")
if ((pol==4)&&(method=="run")) stop("bdrpol=mean only for string bounds")

# split up in policies

##########

if (pol==1) {
 # median of first and last k

 k<-bounds$maxrun-1

 left<-median(data[1:k])
 right<-median(data[(n-k+1):n])

}

##########

if (pol==2) {
 # linear regression of the first k

 if (method=="run"){
   k<-bounds$maxrun-1 }
 else {
   k<-as.integer(3.0*(log(n))**(1/3)*n**(2/3))
 }

 # one should make robust regression here, not l2

 left<-data[1]-lsfit(1:k,data[1:k])$residuals[1]
 right<-data[n]-lsfit(1:k,data[(n-k+1):n])$residuals[k]
}

##########

if (pol==3) {
 # interactive

 # bounds for the height

 upperl<-bounds$upper[1]
 lowerl<-bounds$lower[1]
 upperr<-bounds$upper[n]
 lowerr<-bounds$lower[n]

 # get defaults

 if (method=="run") {
  k<-bounds$maxrun-1
 } else {
  k<-as.integer(0.1*(log(n))**(1/3)*n**(2/3))
print(k)
 }
 leftl<-data[1]-lsfit(1:k,data[1:k])$residuals[1]
 rightl<-data[n]-lsfit(1:k,data[(n-k+1):n])$residuals[k]

 if (method=="run") {
```

```
 k<-bounds$maxrun-1
 leftm<-median(data[1:k])
 rightm<-median(data[(n-k+1):n])
} else {
 leftm<-0.5*(bounds$lower[1]+bounds$upper[1])
 rightm<-0.5*(bounds$lower[n]+bounds$upper[n])
}

# initialize plot for left boundary

plot(data,t='n')
lines(bounds$lower)
lines(bounds$upper)
polygon(c(1,n/3,n/3,1),c(lowerl,lowerl,upperl,upperl),col="lightgreen")
points(data,col='grey')
title("Choosing the boundary value interactively")

# mark other policies

lines(c(1,n/3),c(leftl,leftl),col='red')
lines(c(1,n/3),c(leftm,leftm),col='red')
text(n/6,leftl,'l',col='red')
text(n/6,leftm,'m',col='red')

# start loop

perm<-F

while (!perm) {
 perm<-T
 # read out mouse click position
 pos<-locator(1)
 ind<-as.integer(pos$x)
 left<-pos$y

 if ((ind<1)||(ind>n/3)) perm<-F
 if ((left<lowerl)||(left>upperl)) perm<-F
}

# initialize plot for right boundary

plot(data,t='n')
lines(bounds$lower)
lines(bounds$upper)
polygon(c(2*n/3,n,n,2*n/3),c(lowerr,lowerr,upperr,upperr),col="lightgreen")
points(data,col='grey')
title("Choosing the boundary value interactively")

# mark other policies

lines(c(2*n/3,n),c(rightl,rightl),col='red')
lines(c(2*n/3,n),c(rightm,rightm),col='red')
text(5*n/6,rightl,'l',col='red')
text(5*n/6,rightm,'m',col='red')

# start loop

perm<-F

while (!perm) {
 perm<-T
 # read out mouse click position
 pos<-locator(1)
 ind<-as.integer(pos$x)
 right<-pos$y
```

```
  if ((ind<2*n/3)||(ind>n)) perm<-F
  if ((right<lowerr)||(right>upperr)) perm<-F
 }

}

##########

if (pol==4) {
 # mean of the bounds

 left<-0.5*(bounds$lower[1]+bounds$upper[1])
 right<-0.5*(bounds$lower[n]+bounds$upper[n])
}
##########

# assign left and right to the bounds

left<-min(max(left,bounds$lower[1]),bounds$upper[1])
right<-min(max(right,bounds$lower[n]),bounds$upper[n])

bounds$lower[1]<-left
bounds$upper[1]<-left

bounds$lower[n]<-right
bounds$upper[n]<-right

# make the bounds monotonic again

tmp<-iso.make(bounds$lower,bounds$upper,c(1,bounds$loc,n))
bounds$lower<-tmp$lower
bounds$upper<-tmp$upper
bounds
}

################################################################################
```

A.3 Quadratic Programming

We used source code from Berwin Turlach for quadratic programming [Turlach (1998)]. In order to get the constraints formatted, we designed an interface.

The output delivers computation time for the minimization (`$time`) and the minimal vector (`$min`).

```
###############################################################################
#
# this function transforms the bounds and monotonicity constraints
# into the format required by the solve.QP of Berwin Turlach, see Ref.
# consumes very much resources because constraints are stored in a giant
# n x 3*n-2 matrix where zeros are almost everywhere.

prepare.constraints<-function(bds){
```

```
n<-length(bds$lower)

A<-double(n*n*3)
dim(A)<-c(n,3*n)

# shift bounds a little to achieve
# tractability in the constraints.
# due to machine precision

b<-c(bds$lower-0.5*.Machine$double.eps**.25,
             -bds$upper-0.5*.Machine$double.eps,rep(0,n-1))

for (i in 1:n){
A[i,i]<-1
A[i,n+i]<--1
A[i,2*n+i]<--1
A[i,2*n+i-1]<-1
}
A[1,2*n]<-0
A[n,3*n]<-0

for (i in 1:(n-1)) {

A[,2*n+i]<-bds$iso[i+1]*A[,2*n+i]

}
A<-A[,-(3*n)]

list(b=b,A=A)
}

################################################################################
#
# constructing the Q matrix
# very RAM consuming, too.

prepare.Q<-function(n,mode='metzner'){

if (mode=='metzner') {

if (n>4) {
Q<-double(n*n)
dim(Q)<-c(n,n)

for (i in 3:(n-2)) {
Q[i,i]<-12
Q[i-1,i]<--8
Q[i+1,i]<--8
Q[i-2,i]<-2
Q[i+2,i]<-2
}

Q[1,1]<-4
Q[1,2]<--4
        Q[2,1]<--4
        Q[2,2]<-10
        Q[3,2]<--8
        Q[3,1]<-2
        Q[4,2]<-2
Q[n,n]<-4
        Q[n-1,n]<--4
        Q[n,n-1]<--4
        Q[n-1,n-1]<-10
        Q[n-2,n-1]<--8
```

```
        Q[n-2,n]<-2
Q[n-3,n-1]<-2

} else {
stop(cat('n too small for smoother matrix',mode,'\n'))
}
} else if (mode=='loewendick') {

if (n>6) {
Q<-double(n*n)
dim(Q)<-c(n,n)

for (i in 4:(n-3)) {
Q[i,i]<-16
Q[i-1,i]<--9
Q[i+1,i]<--9
Q[i-3,i]<-1
Q[i+3,i]<-1
}

Q[1,1]<-6
Q[1,2]<--7
        Q[2,1]<--7
Q[1,3]<-2
Q[3,3]<-16
        Q[3,1]<-2
Q[4,1]<-1
Q[2,2]<-16
        Q[3,2]<--10
        Q[2,3]<--10
        Q[5,2]<-1
        Q[4,3]<--9
Q[6,3]<-1

Q[n,n]<-6
Q[n,n-1]<--7
        Q[n-1,n]<--7
Q[n,n-2]<-2
Q[n-2,n-2]<-16
        Q[n-2,n]<-2
Q[n-3,n]<-1
Q[n-1,n-1]<-16
        Q[n-2,n-1]<--10
        Q[n-1,n-2]<--10
        Q[n-4,n-1]<-1
        Q[n-3,n-2]<--9
Q[n-5,n-2]<-1
} else {
stop(cat('n too small for smoother matrix',mode,'\n'))
}
} else {
stop(cat('unknown smoother matrix mode',mode,'\n'))
}
Q
}

################################################################################
#
# This is the call for QP in our smoothing context

qp<-function(bds,mode){

tmp<-prepare.constraints(bds)
Q<-prepare.Q(length(bds$lower),mode)
```

```
tmp1<-system.time(tmp2<-solve.QP(Q,rep(0,length(bds$lower)),tmp$A,tmp$b))[1]
list(time=tmp1,min=tmp2$solution)
}
```

A.4 Modified QSOR

We give the source code of the modified version of the QSOR procedure. For the sake of computation speed, we wrote the routines in FORTRAN. Only minor changes have to be adopted to the program to use Q_L instead of Q_M. For this reason, we only present the algorithm for Q_M.

```
******************************************************************************

      subroutine msnval(f,n,ssum)
      double precision f(n)
      double precision ssum

* calculates the value of S_n
* using the Metzner discretization
* The correction for positive
* definiteness is made

      ssum=f(1)**2+f(n)**2

      do 10 i=2,n-1
ssum=ssum+(f(i-1)+f(i+1)-2D0*f(i))**2
10    continue

* This is the function value up
* to a factor of n**3

      ssum=ssum/2D0
      end

******************************************************************************

      subroutine fqsor(y,fn,fo,l,u,mon,n,omega,active)
      double precision y(n),fn(n),fo(n),l(n),u(n),omega
      integer active(n),iter,mon(n)

* performs a single forward-QSOR-step with Q_M


      fn(1)=(1D0-omega)*fo(1)-omega*(-4D0*fo(2)
     +   +2D0*fo(3))/4D0
      call bdschk(l(1),u(1),fn(1),active(1))

* monotonicity is of no importance
* in the first data point

      fn(2)=(1D0-omega)*fo(2)-omega*(-8D0*fo(3)
     +     +2D0*fo(4)-4D0*fn(1))/10D0
      call monchk(fn(1),fn(2),fo(3),mon(2),
     + active(1),active(2),active(3))
      call bdschk(l(2),u(2),fn(2),active(2))
```

```
* it is important to first check
* monotonicity, then bounds.
* otherwise, the maxima and minima
* are cutted

      do 10 i=3,n-2
      fn(i)=(1D0-omega)*fo(i)-omega*(-8D0*fo(i+1)+2D0*fo(i+2)
     +     +2D0*fn(i-2)-8D0*fn(i-1))/12D0
      call monchk(fn(i-1),fn(i),fo(i+1),mon(i),
     + active(i-1),active(i),active(i+1))
      call bdschk(l(i),u(i),fn(i),active(i))
10    continue

      fn(n-1)=(1D0-omega)*fo(n-1)-omega*(-4D0*fo(n)
     +     +2D0*fn(n-3)-8D0*fn(n-2))/10D0
      call monchk(fn(n-2),fn(n-1),fo(n),mon(n-1),
     + active(n-2),active(n-1),active(n))
      call bdschk(l(n-1),u(n-1),fn(n-1),active(n-1))

      fn(n)=(1D0-omega)*fo(n)-omega*(2D0*fn(n-2)
     +      -4D0*fn(n-1))/4D0
      call bdschk(l(n),u(n),fn(n),active(n))

* monotonicity check for the last point
* is nonsense

*now we check the height of the constance intervals

      call optcon(fn,l,u,active,omega,n)

      return
      end

*****************************************************************************

      subroutine bqsor(y,fn,fo,l,u,mon,n,omega,active)
      double precision y(n),fn(n),fo(n),l(n),u(n),omega
      integer active(n),iter,mon(n)

* performs a single backward-QSOR-step with Q_M


      fn(n)=(1D0-omega)*fo(n)-omega*(-4D0*fo(n-1)
     +   +2D0*fo(n-2))/4D0
      call bdschk(l(n),u(n),fn(n),active(n))

* monotonicity is of no importance
* in the last data point

      fn(n-1)=(1D0-omega)*fo(n-1)-omega*(-8D0*fo(n-2)
     +     +2D0*fo(n-3)-4D0*fn(n))/10D0
      call monchk(fn(n),fn(n-1),fo(n-2),-mon(n-1),
     + active(n),active(n-1),active(n-2))
      call bdschk(l(n-1),u(n-1),fn(n-1),active(n-1))

* it is important to first check
* monotonicity, then bounds.
* otherwise, the maxima and minima
* are cutted

      do 10 i=n-2,3,-1
      fn(i)=(1D0-omega)*fo(i)-omega*(-8D0*fn(i+1)+2D0*fn(i+2)
     +     +2D0*fo(i-2)-8D0*fo(i-1))/12D0
      call monchk(fn(i+1),fn(i),fo(i-1),-mon(i),
```

```
     + active(i+1),active(i),active(i-1))
      call bdschk(l(i),u(i),fn(i),active(i))
10    continue

      fn(2)=(1D0-omega)*fo(2)-omega*(-4D0*fo(1)
     +     +2D0*fn(4)-8D0*fn(3))/10D0
      call monchk(fo(3),fn(2),fn(1),-mon(2),
     + active(3),active(2),active(1))
      call bdschk(l(2),u(2),fn(2),active(2))

      fn(1)=(1D0-omega)*fo(1)-omega*(2D0*fn(3)
     +      -4D0*fn(2))/4D0
      call bdschk(l(1),u(1),fn(1),active(1))

* monotonicity check for the first point
* is nonsense

*now we check the height of the constance intervals

      call optcon(fn,l,u,active,omega,n)

      return
      end

******************************************************************************

      subroutine bdschk(li,ui,yi,acti)
      double precision li,ui,yi
      integer acti

* performs a check of the bounds in
* x_i. acti is set to -1 if the lower,
* to 1 if the upper bound is touched.

      if (yi.lt.li) then
         acti=-1
         yi=li
      end if

      if (yi.gt.ui) then
         acti=1
         yi=ui
      end if

      return
      end

******************************************************************************

      subroutine monchk(yl,yi,yr,moni,actl,acti,actr)
      double precision yl,yi,yr
      integer moni,acti,actl,actr

* performs a check of monotonicity behaviour
* acti is generally set to 0, if
* monotonicity is hurt, acti and the neighbouring
* actl/actr to the left/right is set to 2
* for later height correction.

* must be executed BEFORE bounds check!

      acti=0

      if (moni.eq.-1) then
```

```
         if (yi.gt.yl) then
            acti=2
            actl=2
            yi=yl
         end if

         if (yi.lt.yr) then
            acti=2
            actr=2
            yi=yr
         end if

      end if

      if (moni.eq.1) then

         if (yi.gt.yr) then
            acti=2
            actr=2
            yi=yr
         end if

         if (yi.lt.yl) then
            acti=2
            actl=2
            yi=yl
         end if

      end if

      return
      end

******************************************************************************

      subroutine optcon(fn,l,u,active,omega,n)
      double precision fn(n),l(n),u(n),omega,h,lm,um
      integer active(n),n,left,right

* calculates the height of constance intervals
* marked by active(i)=2.

      do 10 i=3,n-2

* mark the left end of the constance interval

         if ((active(i).eq.2).and.(active(i-1).ne.2)) then
            left=i
         end if

* mark the right end and
*perform the correction

         if ((active(i).eq.2).and.(active(i+1).ne.2)) then
           right=i
           if (right-left.gt.1) then
            h=fn(left)-omega*(-2D0*fn(left-2)+6D0*fn(left-1)
     +                 +6D0*fn(right+1)-2D0*fn(right+2))/8D0
            if (fn(left-1).lt.fn(right+2)) then
               h=min(max(fn(left-1),h),fn(right+1))
            else
               h=max(min(fn(left-1),h),fn(right+1))
            end if

* there could be a max/min or end of data
```

```
* in the constance interval.
* then we have to do nothing at all.

            lm=l(left)
            um=u(left)
            do 3 j=left+1,right
               lm=max(lm,l(j))
               um=min(um,u(j))
 3          continue

            if (lm.ne.um) then

             do 5 j=left,right
                fn(j)=h
5            continue
            end if

          end if
         end if

10    continue

      return
      end

*******************************************************************************

      subroutine mqsor (y,fn,fo,l,u,mon,n,omega,active,k,ssval)
      double precision y(n),fn(n),fo(n),l(n),u(n),omega,ssval(k)
      integer mon(n),active(n),n,k

* performs a k-times iterated combined
* forward/backward modified QSOR step
* with smoothness matrix Q_M

      do 10 i=1,k

* forward / backward combination

      call fqsor(y,fn,fo,l,u,mon,n,omega,active)
      call bqsor(y,fo,fn,l,u,mon,n,omega,active)

* calculate function value after each full QSOR step

      call msnval(fo,n,ssval(i))

 10   continue

      return
      end

*******************************************************************************
```

Now, for the goal of easy usage with different data sets and possible graphical displays, we designed a interface for the R language which will similiarly work for Splus as well.

Some default parameters are set:

- `tmp` is the starting vector for QSOR which is set to be mean of the bounds. Since QSOR reduces the roughness very fast in the beginning but slowly in the end of iterations, the starting vector is of no practical relevance as long as it is permissible, i.e. between the bounds and monotonic.

- omega is the relaxation parameter, see Definition 3.2. The default of 1.5 is somewhat arbitrary. Try some 'exploratory convergence analysis' because omega is rather data–dependent.

- The default for the number of iterations is currently set to be n which gives an n^2 smooothing procedure. Apparently, much higher iteration numbers are necessary to get close to the optimum optically.

```
"mqsor" <-
function (data, bds, tmp = 0.5 * (bds$lower + bds$upper), fl = tmp[1],
    fr = tmp[length(data)], omega = 1.5, iterations = length(data))
{
    n <- length(data)
    tmp <- .Fortran("mqsor", data = as.double(data), double(n),
        y = as.double(tmp), as.double(bds$lower), as.double(bds$upper),
        as.integer(bds$iso), n = n, omega = as.double(omega),
        active = integer(n), iterations = as.integer(iterations),
        Sn.vals = double(iterations))
    list(y = tmp$y, omega = tmp$omega, active = tmp$active, Sn.vals = tmp$Sn.vals,
        iterations = iterations)
}
```

A.5 Shape Preserving Interpolation

We use a spline routine written in C which is part of the R package. spline_coeff evaluates the spline coefficients given the knots (and the interpolationg method, we chose 'natural'). spline_eval computes the values of the spline at some given design points. We call this subroutines in our FORTRAN program which we present now:

```
******************************************************************************

      subroutine zeronum(x,y,b,c,d,nz,k)
      double precision x(k),y(k),b(k),c(k),d(k),
     + h,t1,t2
      integer nz(k-1),k

* calculates the number of
* local extrema of the spline
* in each interval (x_i,x_{i+1})
* using the exact spline coefficients b,c,d
* from the R spline_coef module.


      do 10 i=1,k-1
```

```
        if (d(i).eq.0) then

           if (c(i).eq.0) then

              if (b(i).eq.0) then
                 nz(i)=3
              else
                 nz(i)=0
              end if

           else

              t1=-b(i)/(2D0*c(i))+x(i)

              if ((x(i).lt.t1).and.(x(i+1).gt.t1)) then
                 nz(i)=1
              else
                 nz(i)=0
              end if

              t1=0

           end if

        else

           h=(-3D0*d(i)*b(i)+c(i)**2)/(9D0*d(i)**2)

           if (h.lt.0) then

              nz(i)=0

           else

              t1=sqrt(h)-c(i)/(3D0*d(i))+x(i)
              t2=-sqrt(h)-c(i)/(3D0*d(i))+x(i)

              if ((x(i).lt.t1).and.(x(i+1).gt.t1)) nz(i)=nz(i)+1
              if ((x(i).lt.t2).and.(x(i+1).gt.t2)) nz(i)=nz(i)+1

           end if

        end if

 10   continue

      return
      end

******************************************************************************

      subroutine casenum(x,y,a,b,c,d,k,nz,cn,u,v,n)
      double precision x(k),y(k),a(k),b(k),c(k),d(k)
     + ,u(n),v(n),h1,h2
      integer k,n,nz(k-1),cn(k-1),left,right

* evaluates the case number
* of an interval
* 0 - nothing must be done
* 1 - right border value has to be extended
* 2 - left border has to be extended
* 3 - interval has to be divided due to free constance
*     interval, then left interval being case case 1,
*     right one case 2.
```

```
* zeronum and casenum can be merged
* to decrease computation time.
* did not do it because smaller
* modules are easier to test.

* evaluate the design mesh

       do 1 i=1,n
         u(i)=dble(i)
 1    continue

* get the first spline evaluation
* and the number of local extrema
* in each spline interval

      call spline_coef(2,k,x,y,b,c,d)
      call spline_eval(2,n,u,v,k,x,y,b,c,d)
      call zeronum(x,y,b,c,d,nz,k)

* calculate the case indices

       do 10 i=1,k-1

         cn(i)=0
         left=idint(x(i))
         right=idint(x(i+1))

         if (nz(i).eq.1) then

* is it a max or a min?
* are the x,y isotonic
* or antitonic?


* find max and min of spline in interval

            h1=v(left)
            h2=v(left)

            do 2 j=left+1,right
               if (v(j).lt.h1) h1=v(j)
               if (v(j).gt.h2) h2=v(j)
 2          continue

* first antitonic

            if (v(left).ge.v(right)) then

               if (h1.lt.v(right)) cn(i)=1
               if (h2.gt.v(left)) cn(i)=2

            else

* then isotonic

               if (h2.gt.v(right)) cn(i)=1
               if (h1.lt.v(left)) cn(i)=2

            end if

         end if

         if (nz(i).eq.2) cn(i)=3

 10   continue
```

```
      return
      end

*****************************************************************************

      subroutine splcal(u,v,x,y,b,c,d,n,hk)
      double precision u(n),v(n),b(n),c(n),d(n),x(n),y(n)
      integer n,i,j,hk

* read out knots from u
* (knot positions are marked
* by u(j)=1).

      i=1
      do 1 j=1,n
         if (u(j).eq.1D0) then
            x(i)=dble(j)
            y(i)=v(j)
            i=i+1
         end if
         u(j)=dble(j)
 1    continue
      hk=i-1

* call spline subroutines

      call spline_coef(2,hk,x,y,b,c,d)
      call spline_eval(2,n,u,v,hk,x,y,b,c,d)

* set back u

      do 2 i=1,n
         u(i)=0D0
 2    continue

      do 3 j=1,hk
         u(x(j))=1D0
 3    continue

      return
      end

*****************************************************************************

      function chkint(v,left,right,n)
      double precision v(n),chkint,eps
      integer n,left,right


* help function of minor interest
* checks montonicity in the array v
* if the left and right value are equal,
* it is most practical to set the
* monotonicity value to one. Thus,
* those constance intervals are
* filled with knots!

* accuracy of isotonicity is

      eps=0.000001

      chkint=0D0

      if (v(right)-v(left).gt.0D0) then
```

```
        do 1 i=left,right-1
         if (v(i+1)-v(i).lt.-eps) chkint=1D0
 1      continue
      else
        do 2 i=left,right-1
         if (v(i+1)-v(i).gt.eps) chkint=1D0
 2      continue
      end if
      return
      end

*******************************************************************************

      subroutine defcor(x,y,hx,hy,a,b,c,d,k,nz,cn,u,v,n,
     +ha,hb,hc,hd)
      double precision x(n),y(n),a(n),b(n),c(n),d(n),u(n),v(n)
     + ,hx(n),hy(n),ha(n),hb(n),hc(n),hd(n)
      integer n,k,nz(n),cn(n),left,right,hk,mid,r,l

* call the case number calculation
* the start knots are in x,y and
* filled up with zeros.
* also returns a first spline
* evaluation. u must be 1:n as
* double precision array.

* zeronum and casenum can be merged
* into this program to speed up the
* calculations. did not do it because
* smaller modules are easier to test

      call casenum(x,y,a,b,c,d,k,nz,cn,u,v,n)

* set u back to knot indices

      do 11 i=1,n
         u(i)=0D0
 11   continue

* set marks for original knots
* and corresponding heights

      do 1 j=1,k
            u(x(j))=1
            v(x(j))=y(j)
 1    continue

* each interval is examined separately

      do 10 i=1,k-1

         left=int(x(i)+0.001)
         right=int(x(i+1)+0.001)

         if (cn(i).eq.1) then

* right border value has to be extended
* can be speeded up by bisection methods

            do 2 j=right-1,left+1,-1
               u(j)=1D0
               v(j)=y(i+1)

* check montonicity
```

```
            call splcal(u,v,hx,hy,hb,hc,hd,n,hk)
            if (chkint(v,left,right,n).eq.0) goto 3
 2        continue

* everything is clear

 3        continue

       else if (cn(i).eq.2) then

* left border value has to be extended
* can be speeded up by bisection methods

          do 4 j=left+1,right-1
             u(j)=1D0
             v(j)=y(i)

* check montonicity

             call splcal(u,v,hx,hy,hb,hc,hd,n,hk)
             if (chkint(v,left,right,n).eq.0) goto 5

 4        continue

* everything is clear

 5        continue

       else if (cn(i).eq.3) then

* first, a suitable midpoint has to be introduced.

* find extrema by use of the spline coefficients

            h=(-3D0*d(i)*b(i)+c(i)**2)/(9D0*d(i)**2)
            t1=sqrt(h)-c(i)/(3D0*d(i))+x(i)
            t2=-sqrt(h)-c(i)/(3D0*d(i))+x(i)

* mark midpoint between extremes with suitable height.

            mid=int((t1+t2)*0.5)
            mid=max(min(right-1,mid),left+1)
            u(mid)=1D0
            if (y(i+1)-y(i).gt.0) then
               v(mid)=max(y(i),min(y(i+1),
     +              v(mid)))
            else
               v(mid)=min(y(i),max(y(i+1),
     +              v(mid)))
            end if

* call spline machine

            call splcal(u,v,hx,hy,hb,hc,hd,n,hk)

* then we handle the two halfs like case 1,2.

* set up interval border

            l=int(x(i)+0.0001)
            r=mid

* right border value has to be extended
* can be speeded up by bisection methods
```

```
            do 22 j=r-1,l+1,-1
               u(j)=1D0
               v(j)=v(r)

* check montonicity

               call splcal(u,v,hx,hy,hb,hc,hd,n,hk)
               if (chkint(v,l,r,n).eq.0) goto 33
 22         continue

* everything is clear

 33         continue

* set up interval border

            l=mid
            r=int(x(i+1)+0.0001)

* left border value has to be extended
* can be speeded up by bisection methods

            do 44 j=l+1,r-1
               u(j)=1D0
               v(j)=v(l)

* check montonicity

               call splcal(u,v,hx,hy,hb,hc,hd,n,hk)
               if (chkint(v,l,r,n).eq.0) goto 55
 44         continue

* everything is clear

 55         continue

         end if

 10   continue


* give back number of knots

      k=hk

      return
      end

******************************************************************************
```

For the use in R, we designed an interface.

```
"monspl" <-
function (x, y)
{
# calculated the adhoc isogeometric spline
# through the knots x,y
# x is an integer vector of design mesh indices,
# containing the first (1) and last point (n)
```

```
    k <- length(x)
    n <- x[k]
    tmp <- .Fortran("defcor", x = as.double(x), y = as.double(y),
        hx = double(n), hy = double(n), a = double(n), b = double(n),
        c = double(n), d = double(n), k = as.integer(k), nz = integer(n),
        cn = integer(n), u = as.double(c(1:n)), v = double(n),
        as.integer(n), double(n), double(n), double(n), double(n))
# output list contains the new knots (including the old)
# and the spline evaluation at all design points.

    tmp <- list(knots = list(x = tmp$hx[1:tmp$k], y = tmp$hy[1:tmp$k]),
        val = tmp$v)
}
```

A.6 Hybrid Method

Since loops are much faster in FORTAN than in R code, we designed a FORTRAN routine which checks if a vector y satisfies the bounds and if not, where the bounds are hurt.

```
******************************************************************************
* this routine checks the bounds
* up to tolerance level tol
* made in FORTRAN for computional speed
* marks indices of hurt

      subroutine chkbnds(y,u,l,n,ind,tol,ok)
      double precision y(n),u(n),l(n),tol
      integer n,ind(n)
      logical ok

      ok=.TRUE.

      do 10 i=1,n

         if (y(i)+tol.gt.u(i)) then
ok=.FALSE.
          ind(i)=1
 end if

         if (y(i)-tol.lt.l(i)) then
ok=.FALSE.
ind(i)=1
 end if

 10   continue
      return
      end

******************************************************************************
```

Then, we present the R hybrid algorithm object. Some arguments are needed, and some defaults are given:

- `data` is not really important for the hybrid algorithm, but gives the length of all data–type object (such as bounds) and is plotted if the `verbose` parameter is set `T`.
- `bds` has to be a bounds object like it is produced from the data by our `create.bounds` object.
- `omega` is the relaxation parameter for QSOR, default is 1.5.
- `alpha` (default 1.1) is the increase rate for the number of QSOR steps between two spline interpolation attempts. Must be greater than 1 to avoid loops, should not be too large to avoid unnecessary much QSOR steps.
- `tol` is a tolerance parameter. Controls how near a point must be to the bounds to be counted as touching and how accurate the bounds are satisfied by the solution. Smaller values of `tol` tend to detect knots of the solution later than greater values, but too great tolerance values sometimes allow suboptimal termination of the algorithm.
- `insert.spline` is a switch. 'T' allows the insertion of the spline attempt values to speed up the QSOR algorithm which is often faster than 'F', but does not converge all time against the solution.
- `limit` allows the specification of a upper boundary for QSOR steps. If this limit is reached, the hybrid algorithm is stopped and the `$success` output is set to 'F'. If the algorithm terminates before a total number of `limit` QSOR steps have been made, `$success` is 'T'.
- `verbose` is a switch for graphical display of the iteration process.

```
##########################################################################
# This routine is the hybrid algorithm of QSOR/MONSPL



hybrid<-function(data,bds,omega=1.5,alpha=1.1,tol=0.000001,insert.spline=F,
limit=length(data)**2,verbose=F){

n<-length(data)

# initializing

constraints<-F
alpha<-1.1
iterations<-10
tmp<-list(y=0.5*(bds$lower+bds$upper))
insgesamt<-0

while (!constraints) {

# make QSOR steps and set number of iterations for next time
```

```
tmp<-mqsor(data,bds,tmp=tmp$y,iterations=iterations,omega=omega)

if (verbose) {
plot(data,t='n')
lines(bds$lower,col='grey')
lines(bds$upper,col='grey')
lines(tmp$y,col='red')
}

insgesamt<-insgesamt + iterations
iterations<-as.integer(iterations*alpha)

# calculates if and where bounds are hurt

active<-.Fortran('chkbnds',y=as.double(tmp$y),
u=as.double(bds$upper),
l=as.double(bds$lower),
n=as.integer(n),
active=integer(n),
as.double(tol),
ok=logical(1))$active

active<-c(1:n)[active==1]

# prepare the knots and interpolate

kx<-active
ky<-tmp$y[active]

spl<-monspl(kx,ky)

if (verbose) {
lines(spl$val,col='green')
readline()
}

# check bounds

constraints<-.Fortran('chkbnds',y=as.double(spl$val),
u=as.double(bds$upper),
l=as.double(bds$lower),
n=as.integer(n),
active=integer(n),
as.double(-tol),
ok=logical(1))$ok

# insert spline pieces in next iteration?

if (insert.spline) tmp$y<-spl$val


# check if limit for overall iterations is reached

if (limit<insgesamt) constraints<-T

}

# one single QSOR step for correction of the height of constance intervals

insgesamt<-insgesamt+1
tmp<-mqsor(data,bds,tmp=spl$val,iterations=iterations,omega=omega)


list(qsor.steps=insgesamt,y=tmp$y,success=(limit>insgesamt))
```

```
}
```

A.7 String polishing

For coding convenience, we did all of the parametric function stuff completely in R. For the sake of computation speed, this must be transferred to FORTRAN or C, but since the algorithms are very fast even for large data sets, we do not see the necessity to do so.

```
################################################################################
#
# evaluates the parametric function at t

geval<-function(t,a=0,b=0,ce,alpha=0,beta=0,gamma,delta){

tmp<-double(length(t))

if (delta == 1) {
tmp<-{a/((alpha+1)*(alpha+2))*t**(alpha+2)+
     b/((beta+1)*(beta+2))*(1-t)**(beta+2)-
     b/((beta+1)*(beta+2))+b/(beta+1)*t+
     ce/((gamma+1)*(gamma+2))*t*(1-t)**(gamma+2)+
     2*ce/((gamma+1)*(gamma+2)*(gamma+3))*(1-t)**(gamma+3)-
     2*ce/((gamma+1)*(gamma+2)*(gamma+3))+
     ce/((gamma+1)*(gamma+2))*t}
} else {
tmp<-{a/((alpha+1)*(alpha+2))*t**(alpha+2)+
     b/((beta+1)*(beta+2))*(1-t)**(beta+2)-
     b/((beta+1)*(beta+2))+b/(beta+1)*t+
     ce/((delta+1)*(delta +2))*t**(delta +2)-
     ce/((delta+2)*(delta +3))*t**(delta +3)}
}
list(x=t,y=tmp)
}

################################################################################
#
# evaluates the first order derivative of the
# parametric function at t

dgeval<-function(t,a=0,b=0,ce,alpha=0,beta=0,gamma,delta){

tmp<-double(length(t))

if (delta == 1) {
tmp<-{a/(alpha+1)*t**(alpha+1)-
     b/(beta+1)*(1-t)**(beta+1)+
     b/(beta+1)-ce/((gamma+1)*(gamma+2))*(1-t)**(gamma+2)+
     ce/((gamma+1)*(gamma+2))}
} else {
tmp<-{a/(alpha+1)*t**(alpha+1)-
     b/(beta+1)*(1-t)**(beta+1)+b/(beta+1)+
     ce/(delta+1)*t**(delta +1)-
     ce/(delta+2)*t**(delta +2)}
}
list(x=t,y=tmp)
}
```

```
################################################################################
#
# calculates parameter constellation according Theorem 4.1
# replicating the given boundary values for isotonic convex functions.

find.pars<-function(yr,dyr,ddyl=0,ddyr=0){

# intializing

w<-1.5
a<-ddyr
b<-ddyl

# if the function can be rebuilt
# get suitable parameters.
# if not, give back zeros
# which corresponds to the straight line

d<-yr/dyr

# check whether parametric reconstruction is possible.

if ((!is.na(d))&&(d>0)&&(d<1)) {

 # find a suitable alpha

 a1<-sqrt(a/yr+1/4)-3/2
 a2<-a/dyr-1
 a3<-a/(dyr-yr)-2
 alpha<-w*max(1/w,a1,a2,a3)

 # find a suitable beta

 yr<-yr-a/((alpha+1)*(alpha+2))
 dyr<-dyr-a/(alpha+1)

 b1<-sqrt(b/(dyr-yr)+1/4)-3/2
 b2<-b/dyr-1
 b3<-b/yr-2
 beta<-w*max(1/w,b1,b2,b3)

 # calculate discriminant function

 yr<-yr-b/(beta+2.0)
 dyr<-dyr-b/(beta+1.0)

 d<-yr/dyr
 warn<-F


 if (d<0.5) {
gamma<-1
delta<-(2*dyr-3*yr)/yr
ce<-(delta+1)*(delta+2)*dyr
 } else {
delta<-1
gamma<-(3*yr-dyr)/(dyr-yr)
ce<-(gamma+1)*(gamma+2)*dyr
 }


} else {

# no convex solution, give zeros back to calling routine
```

```
# and warn

warn<-T

alpha<-0
beta<-0
gamma<-0
delta<-0
a<-0
b<-0
ce<-0

}

list(alpha=alpha,beta=beta,gamma=gamma,delta=delta,ce=ce,a=a,b=b,warn=warn)
}

##############################################################################
#
# transforms isotonic convex function to standard position

trafo.hin<-function(x,y,dyl,dyr,ddyl,ddyr){

bn<-x[2]-x[1]
yr<-y[2]

# subtraction of straight line with slope dyl

m<-dyl
b<-m*x[1]-y[1]
y<-y-m*x-b

dyl<-dyl-m # now vanishing
dyr<-dyr-m

# scale to unit interval

dyl<-(x[2]-x[1])*dyl # still vanishes
dyr<-(x[2]-x[1])*dyr

ddyr<-(x[2]-x[1])**2*ddyr
ddyl<-(x[2]-x[1])**2*ddyl

x<-(x-x[1])/(x[2]-x[1])

# meet origin

y<-y-y[1]

list(x=x,y=y,yr=y[length(y)],dyr=dyr,ddyl=ddyl,ddyr=ddyr)
}

##############################################################################
#
# back transform, invers to trafo.hin

trafo.her<-function(x,y,xold,yold,dyl){

xl<-xold[1]
xr<-xold[2]
yl<-yold[1]
yr<-xold[2]

x<-x*(xr-xl)+xl
y<-y+dyl*x
```

```
y<-y+yl-y[1]

list(x=x,y=y)

}

################################################################################
#
# back transform, invers to trafo.hin
# but for first order derivative.

dtrafo.her<-function(x,y,xold,yold,dyl){

xl<-xold[1]
xr<-xold[2]
yl<-yold[1]
yr<-xold[2]

y<-y/(xr-xl)+dyl

list(x=x,y=y)

}

################################################################################
#
# splits string in pieces and devises boundary values.
# calculates the shape

divide<-function(faden){

kn<-length(faden$ind)-1

# set up function values in intervals

xl<-faden$ind[1:kn]
xr<-faden$ind[2:(kn+1)]

yl<-faden$y[1:kn]
yr<-faden$y[2:(kn+1)]

# set up first order derivative

 m<-(yr-yl)/(xr-xl)

 dyl<-c(m[1],(m[2:kn]+m[1:(kn-1)])/2)
 dyr<-c((m[2:kn]+m[1:(kn-1)])/2,m[kn])

 # detect extrema
 # 1  - xl,yl is max of string
 # -1 -          min

 ext<-integer(kn)

 for (i in 2:kn){

 ext[i]<- -((yl[i-1]>=yl[i])&&(yl[i]<=yr[i]))+
           ((yl[i-1]<=yl[i])&&(yl[i]>=yr[i]))
 }

 ext<-c(1:kn)[ext!=0]

 # dy=0 in extrema

  dyl[ext]<-0
```

```
  dyr[ext-1]<-0

# set up second order derivative.

 m<-(dyr-dyl)/(xr-xl)

 ddyl<-c(0,(m[2:kn]+m[1:(kn-1)])/2)
 ddyr<-c((m[2:kn]+m[1:(kn-1)])/2,0)

 # detect inflection intervals
 # those intervals need special derivatives

 infl<-c(2:(kn-1))[(ddyl*ddyr)[2:(kn-1)]<0]

 ddyl[infl]<-0
 ddyr[infl]<-0
 ddyr[infl-c(rep(1,length(infl)))]<-0
 ddyl[infl+c(rep(1,length(infl)))]<-0

 # correct first order derivative

 dyl[infl]<-((yl-yr)/(xl-xr))[infl]
 dyr[infl]<-dyl[infl]
 dyl[infl+c(rep(1,length(infl)))]<-dyl[infl]
 dyr[infl-c(rep(1,length(infl)))]<-dyl[infl]

 dyl[2]<-dyl[1]
 dyr[1]<-dyl[1]
 dyl[kn]<-dyr[kn]
 dyr[kn-1]<-dyr[kn]


# calculate shapes
# -2 left boundary
# +2 right boundary
# -1 convex
# +1 concave
#  0 where infl
# 99 technical flag: interval to small, no grid point
#                    in this interval

shp<-integer(kn)

shp[infl]<-0


shp[1]<--0
shp[kn]<-0

if (kn>2) {
 shp[2:(kn-1)]<--(ddyl[2:(kn-1)]+ddyr[2:(kn-1)]>0)+
                 (ddyl[2:(kn-1)]+ddyr[2:(kn-1)]<0)
}

# mark 1 point intervals

eng<-logical(kn)
eng<-(xr-xl<1.5)
shp[eng]<-99

list(xl=xl,xr=xr,yl=yl,yr=yr,dyl=dyl,dyr=dyr,ddyl=ddyl,ddyr=ddyr,kn=kn,shp=shp)


}
```

```
################################################################################
#
# main function. cuts in pieces and rebuilds parametrically
#
# policies for devising boundary values:
#
# differences  first order differences for dy
# sec. order differences for ddy
# sec.der.zero first order differences for dy
# zero for ddy
# maxsmooth first order differences for dy
# smoothness maximum for ddy on the right.
#
# default policy is sec.der.zero

fsmooth<-function(bounds,policy="sec.der.zero",verbose=F){

# get string through the bounds

faden<-string.through.bounds(bounds)

# cut the string down to handsome size.

n<-faden$ind[length(faden$ind)]
p<-divide(faden)

pol<-match(policy,c("differences","sec.der.zero","maxsmooth"))

if ((is.na(pol))||(pol>3)) stop(cat("unknown policy:",policy))

########## policy differences done automatically

########## policy sec.der.zero

if (pol==2) {

 p$ddyl<-c(rep(0,p$kn));p$ddyr<-c(rep(0,p$kn))

}

########## policy proposal 2

if (pol==3) {

 # call sm.imp1
 p<-sm.imp1(p)

}

########## if nothing has happened, policy is differences

y<-double(n)
dy<-double(n)
warn<-logical(p$kn)
warn<-rep(T,p$kn)
crvt<-double(p$kn)

# done. now transform to standard form
# and get it convex isotonic
# piecewise

while (sum(warn)>0) {

for (i in 1:p$kn){
```

```
 # treat different shapes different.

 if (verbose) print(c(i,"shape",p$shp[i]))


 ##########

 if (p$shp[i]==99) {

  # nothing can be done.

  tmp<-c(p$yl[i],p$yr[i])
  dtmp<-c(rep((p$yr[i]-p$yl[i])/(p$xr[i]-p$xl[i]),2))
  ftmp<-list(warn=F)
crvt[i]<-0
 }

 ##########

 if (p$shp[i]==-1) {

  # convex function.

  tmp<-trafo.hin(c(p$xl[i],p$xr[i]),c(p$yl[i],p$yr[i]),
p$dyl[i],p$dyr[i],p$ddyl[i],p$ddyr[i])
  ftmp<-find.pars(tmp$yr,tmp$dyr,tmp$ddyl,tmp$ddyr)
  t<-0:(p$xr[i]-p$xl[i])/(p$xr[i]-p$xl[i])

  # evaluate

  gtmp<-geval(t,ftmp$a,ftmp$b,ftmp$ce,ftmp$alpha,
ftmp$beta,ftmp$gamma,ftmp$delta)
  dgtmp<-dgeval(t,ftmp$a,ftmp$b,ftmp$ce,ftmp$alpha,
ftmp$beta,ftmp$gamma,ftmp$delta)

  # back transform

  tmp<-trafo.her(gtmp$x,gtmp$y,c(p$xl[i],p$xr[i]),
c(p$yl[i],p$yr[i]),p$dyl[i])$y

  dtmp<-dtrafo.her(dgtmp$x,dgtmp$y,c(p$xl[i],p$xr[i]),
c(p$yl[i],p$yr[i]),p$dyl[i])$y
 }

 ##########

 if (p$shp[i]==1) {

  # concave function.
  tmp<-trafo.hin(c(p$xl[i],p$xr[i]),c(-p$yl[i],-p$yr[i]),
-p$dyl[i],-p$dyr[i],-p$ddyl[i],-p$ddyr[i])
  ftmp<-find.pars(tmp$yr,tmp$dyr,tmp$ddyl,tmp$ddyr)
  t<-0:(p$xr[i]-p$xl[i])/(p$xr[i]-p$xl[i])

# evaluate

  gtmp<-geval(t,ftmp$a,ftmp$b,ftmp$ce,ftmp$alpha,
ftmp$beta,ftmp$gamma,ftmp$delta)
  dgtmp<-dgeval(t,ftmp$a,ftmp$b,ftmp$ce,ftmp$alpha,
ftmp$beta,ftmp$gamma,ftmp$delta)

# back transform

  tmp<-trafo.her(gtmp$x,-gtmp$y,c(p$xl[i],p$xr[i]),
c(p$yl[i],p$yr[i]),p$dyl[i])$y
```

```
  dtmp<-dtrafo.her(dgtmp$x,-dgtmp$y,c(p$xl[i],p$xr[i]),
c(p$yl[i],p$yr[i]),p$dyl[i])$y
 }

 ##########

 if (p$shp[i]==0) {

  # inflection/boundary interval
  # put the straight line in

  t<-p$xl[i]:p$xr[i]

# evaluate

  tmp<-approx(c(p$xl[i],p$xr[i]),c(p$yl[i],p$yr[i]),t)$y
  dtmp<-c(rep((p$yl[i]-p$yr[i])/(p$xl[i]-p$xr[i]),length(tmp)))

crvt[i]<-0

  ftmp<-list(warn=F)

}

 ##########

 y[p$xl[i]:p$xr[i]]<-tmp
 dy[p$xl[i]:p$xr[i]]<-dtmp
 warn[i]<-ftmp$warn

 if (verbose){
  if (p$shp[i]!=99) {
   plot(c(p$xl[i],p$xr[i]),c(p$yl[i],p$yr[i]),t='l')
   lines(c(p$xl[i]:p$xr[i]),y[c(p$xl[i]:p$xr[i])],col='red')
   readline()
  }
 }
}

# were there warnings?

p$shp[c(1:p$kn)[warn]]<-rep(0,sum(warn))

if (verbose) {
print("warnings in following intervals")
print(c(1:p$kn)[warn])
}
}


# put back y values and first order derivative.

list(y=y,dy=dy,crvt=crvt)

}

################################################################################
#
# calculates the curvature of the parametric function given the parameters
#

curvature<-function(p){

# calculations
```

```
t1<-p$a**2/(2*p$alpha+1)+p$b**2/(2*p$beta+1)
t2<-2*p$a*p$b*beta(p$alpha+1,p$beta+1)
t3<-p$ce**2*beta(2*p$gamma+1,2*p$delta+1)
t4<-2*p$a*p$ce*beta(p$alpha+p$delta+1,p$gamma+1)
t5<-2*p$b*p$c*beta(p$gamma+p$beta+1,p$delta+1)

tmp<-t1+t2+t3+t4+t5
tmp

}

################################################################################
#
# maximize smoothness by sec. order der. to the right.
# (proposal 2)

psm.imp1<-function(yr,dyr,ddyl,ddyr){

# test k different values

k<-10

# test value selection is difficult
# other choices will do as well

test<-c(rep(ddyr,k))*c(1:k)/(k/4)

# should better be done by FORTRAN routine
# for the sake of computation speed
# is done in R for the sake of coding simplicity

crvt<-double(k)

for (i in 1:k){

crvt[i]<-curvature(find.pars(yr,dyr,ddyl,test[i]))

}

ddyr<-min(test[min(crvt)==crvt])

list(x=test,ddyr=ddyr,y=crvt)
}

################################################################################
#
# collect the single pieces in a divide-type object
#

sm.imp1<-function(p){

for (i in 1:p$kn) {

 if ((p$shp[i]!=0)&&(abs(p$shp[i])<=1)) {


  if (p$shp[i]==1) {

  # concave function.
  tmp<-trafo.hin(c(p$xl[i],p$xr[i]),c(-p$yl[i],-p$yr[i]),
-p$dyl[i],-p$dyr[i],-p$ddyl[i],-p$ddyr[i])

  ftmp<-psm.imp1(tmp$yr,tmp$dyr,tmp$ddyl,tmp$ddyr)
```

```
  # retransform the second order derivative

  p$ddyr[i]<--ftmp$ddyr/((p$xr[i]-p$xl[i])**2)
  p$ddyl[i+1]<-p$ddyr[i]

  }

  if (p$shp[i]==-1) {

  # convex function.
  tmp<-trafo.hin(c(p$xl[i],p$xr[i]),c(p$yl[i],p$yr[i]),
p$dyl[i],p$dyr[i],p$ddyl[i],p$ddyr[i])

  ftmp<-psm.imp1(tmp$yr,tmp$dyr,tmp$ddyl,tmp$ddyr)

  # retransform the second order derivative

  p$ddyr[i]<-ftmp$ddyr/((p$xr[i]-p$xl[i])**2)
  p$ddyl[i+1]<-p$ddyr[i]

  }
 }
}
# give back modified p
p
}


################################################################################
#
# function call for the taut string through bounds
# calls a C function bindfaden provided by Kovac

"string.through.bounds" <-function (bounds)
{
    n<-length(bounds$upper)
    x<-c(1:n)
    y.low<-bounds$lower
    y.up<-bounds$upper
    tmp <- .C("bindfaden", as.double(x), as.double(y.low), as.double(y.up),
        as.double(y.low[1]), as.double(y.low[n]), as.integer(length(x)),
        knotsind = integer(length(x)), knotsy = double(length(x)),
        nknots = integer(1), nextr = integer(1))
    list(ind = tmp$knotsind[1:tmp$nknots], y = tmp$knotsy[1:tmp$nknots],
        nextr = tmp$nextr)
}
```

Appendix B

Data

B.1 Mixture of Cauchy Densities

We generated the underlying signal by

```
x<-seq(0,1,0.01)
y<-2.0*dcauchy(x,0.4,0.1)
   +1.5*dcauchy(x,0.6,0.05)
   +rnorm(101,0,0.5)
```

which is a weighted mixture of two Cauchy density functions at equidistant design.

B.2 The Härdle data

This data set is taken from [Härdle (1991)]. It is an artificial data set of the form

$$y_i = \left(\sin(2\pi x_i^3)\right)^3 + \varepsilon_i$$

where the ε_i are i.i.d. $\mathcal{N}(0, 0.1)$. The generated data are of length 256 In the original data set, the x_i were uniformly distributed on $[0, 1]$. We used equidistant design instead.

B.3 The Rescaled Donoho–Johnstone Test Signals

These test signals are well known throughout the regression community. They appear for example in [Donoho and Johnstone (1995)]. We are not shure that this reference is the first were the signals were introduced. For the scale factors we refer to [Kovac (1998)] where the original signals are rescaled to have equal power. The test signal can be downloaded from the STATLIB archive and are also contained in our software package.

All test signals were evaluated at an equidistant desgin mesh of the 2048 points $i/n, 0 \leq i \leq 2047$, then adding i.i.d. $\mathcal{N}(0, 0.4)$ noise. We list the underlying signals now.

Rescaled Doppler:

$$y_(t) = 7.646146 \cdot \left(\sqrt{t(1-t)} \cdot \sin\left(\frac{2\pi \cdot 1.05}{t + 0.05}\right)\right)$$

Rescaled Heavisine:

$$y(t) = 0.7439853 \cdot (4 \cdot \sin(4\pi t) - \text{sgn}(t - 0.3) - \text{sgn}(0.72 - t))$$

Rescaled Blocks: The Blocks signal consists of constant pieces of specific heights and widthes. It does not make much sense to display the distinct position of the constance intervals. The scale constant we used is 1.154517.

Rescaled Bumps: The Bumps signal is the weighted sum of 11 bumps of the type

$$\frac{a}{1 + \left(\frac{|t-b|}{c}\right)^4}$$

where different parameter constellations a, b, c are used. We scaled the orginal signal with the factor 3.326416.

B.4 Akima Interpolation Data

x	0	2	3	5	6	8	9	11	12	14	15
y	10	10	10	10	10	10	10.5	15	56	60	85

The source of the data can be found in [Wichmann (1998)].

B.5 Späth Interpolation Data

x	0	2	2.5	3.5	5.5	6	7	8.5	10
y	2	2.5	4.5	5	4.5	1.5	1	0.5	0

The source of the data can be found in [Wichmann (1998)].

Bibliography

[Andersson and Elfving (1988)] L. Andersson and T. Elfving. Interpolation and approximation by monotone cubic splines. *Journal of Approximation Theory*, 1988.

[Barlow et al. (1972)] R. E. Barlow, D. J. Bartholomew, J.M. Bremner, and H.D. Brunk. *Statistical Inference under Order Restrictions*. Wiley, Chichester, 1972.

[Beatson and Wolkowitz (1989)] Beatson and Wolkowitz. Postprocessing piecewise cubics for monotonicity. *SIAM Journal of Numerical Analysis*, 26:480–502, 1989.

[Chaudhuri and Marron (1999)] P. Chaudhuri and J.S. Marron. Sizer for exploration of structures in curves. *Journal of the American Statistical Association*, 1999.

[Constantini (1986)] P. Constantini. On monotone and convex spline interpolation. *Math. Comput.*, 46:203–214, 1986.

[Davies (1995)] P. L. Davies. Data features. *Statistica Neerlandica*, 49:185–245, 1995.

[Davies and Kovac (1999)] P. L. Davies and A. Kovac. Modality, runs, strings, and multiresolution. submitted, 1999.

[Davies and Dümbgen (1999)] P.L. Davies and L. Dümbgen. Isotonic regression with local median, 1999. oral communication at the SFB 475 conference at Dortmund, March 1999.

[DeBoor (1978)] DeBoor. *A practical guide to splines*, chapter IX. Springer, Berlin, New York, 1978.

[Diereckx (1980)] P. Diereckx. An algorithm for cubic spline fitting with convexity constraints. *Computing*, 24:349–371, 1980.

[Donoho and Johnstone (1995)] D. L. Donoho and I. M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90:1200–1224, 1995.

[Donoho et al. (1995)] D. L. Donoho, I. M. Johnstone, Kerkyacharian G., and D. Picard. Wavelet shrinkage: Asymptopia? *Journal of the Royal Statistical Society B*, 57(2):301–369, 1995.

[Dümbgen (1998a)] L. Dümbgen. Application of local rank tests to nonparametric regression. Preprint, 1998a.

[Dümbgen (1998b)] L. Dümbgen. New goodness–of–fit tests and their application to nonparametric confidence sets. *Annals of Statistic*, 26:288–314, 1998b.

[Ekeland and Temam (1976)] I. Ekeland and R. Temam. *Convex Analysis and Variational Problems*. Number 1 in Studies in Mathematics and its applications. North Holland, Amsterdam – Oxford, 1976.

[Fan and Gijbels (1996)] J. Fan and I. Gijbels. *Local Polynomial Modeling and Its Applications*. Chapman & Hall, 1996.

[Fisher et al. (1994)] N. I. Fisher, E. Mammen, and J. S. Marron. Testing for multimodality. *Computional Statistics and Data Analysis*, 18:499–512, 1994.

[Friedman and Tibshirani (1984)] J. Friedman and R. Tibshirani. The monotone smoothing of scatterplots. *Technometrics*, 26:343–350, 1984.

[Goldfarb (1986)] D. Goldfarb. Efficient primal algorithms for strictly convex quadratic programs. In *Numerical Analysis, Proceedings, Guanajuato, Mexico 1984*, volume 1230-4 of *Lecture Notes in Mathematics*, pages 11–25, New York, 1986. Springer.

[Goldfarb and Idnani (1982)] D. Goldfarb and A. Idnani. Dual and primal-dual methods for solving strictly convex quadratic programs. In *Numerical Analysis, Proceedings, Cocoyoc, Mexico, 1981*, volume 909 of *Lecture Notes in Mathematics*, New York, 1982. Springer.

[Goldfarb and Liu (1993)] D. Goldfarb and S. Liu. An $o(n^3 l)$ primal-dual potential reduction algorithm for solving convex quadratic programs. *Mathematical Programming*, 61A(2):161–170, 1993.

[Good and Gaskins (1980)] I. Good and R. Gaskins. Density estimation and bump–hunting by the penalized likelihood method exemplified by scattering and meteorite data. *Journal of the American Statistical Association*, 75: 42–73, 1980.

[Großmann and Terno (1993)] Chr. Großmann and J. Terno. *Numerik der Optimierung*. Teubner, Stuttgart, 1993.

[Hampel (1974)] F. R. Hampel. Beyond location parameters: robust concepts and methods (with discussion). In *Proceedings of the 40th Session of the ISI*, volume 46, pages 375–391, 1974.

[Härdle (1991)] W. Härdle. *Smoothing Techniques With Implementation in S*. Springer, New York, 1991.

[Hartigan and Hartigan (1985)] J. A. Hartigan and P. M. Hartigan. The dip test of unimodality. *Annals of Statistics*, 13:70–84, 1985.

[Hengartner and Stark (1995)] N. Hengartner and P. Stark. Finite–sample confidence envelopes for shape–restricted densities. *Annals of Statistics*, 13: 70–84, 1995.

[Hildreth (1954)] C. Hildreth. Point estimates of ordinates of concave functions. *Journal of the American Statistical Association*, 49:598–619, 1954.

[Hornung (1978)] U. Hornung. Monotone spline–interpolation. In L. Collatz, G. Meinardus, and H. Werner, editors, *Numerische Methoden der Approximationstheorie*, volume 42 of *ISNM*, pages 172–191, Basel, Stuttgart, 1978. Birkhäuser.

[Jones and Wand (1995)] M. C. Jones and M. P. Wand. *Kernel Smoothing*. Chapman & Hall, 1995.

[Kovac (1998)] A. Kovac. *Wavelet Thresholding for Unequally Time-Spaced Data*. PhD thesis, University of Bristol, 1998.

[Kovac and Silverman (1999)] A. Kovac and B. W. Silverman. Extending the scope of wavelet thresholding. *submitted*, 1999.

[Kvasov (1996)] B. I. Kvasov. Isogeometric interpolation by generalized splines. *Journal of Numerical Analysis and Mathematical Models*, 11(3):222–246, 1996.

[Lempio (1972)] F. Lempio. Bemerkungen zur lagrangeschen funktionaldifferentialgleichung. In J. Albrecht and L. Collatz, editors, *Numerische Methoden bei Differentialgleichungen und mit funktionalanalytischen Hilfsmitteln*, volume 19 of *ISNM*, pages 141–146, Basel, 1972. Birkhäuser.

[Leurgans (1982)] S. Leurgans. Asymptotic distributions of slope–of–greatest–convex–minorant estimators. *Annals of Statistics*, 10:287–296, 1982.

[Li et al. (1996)] W. Li, D. Naik, and J. Swetits. A data smoothing technique for piecewise convex/concave curves. *Statistics and Computing*, 17:517–537, 1996.

[Lions and Magenes (1972)] J. L. Lions and E. Magenes. *Non–Homogeneous Boundary Value Problems and Applications I*, volume 181 of *Die Grundlehren der mathematischen Wissenschaft in Einzeldarstellungen*. Springer, Berlin, Heidelberg, New York, 1972.

[Mächler (1995)] M. Mächler. Variational solution of penalized likelihood problems and smooth curve estimation. *Annals of Statistics*, 23, 1995.

[Mammen (1991a)] E. Mammen. Estimating a smooth monotone regression function. *Annals of Statistics*, 19:724–740, 1991a.

[Mammen (1991b)] E. Mammen. Nonparametric regression under qualitative smoothness assumptions. *Annals of Statistics*, 19:741–759, 1991b.

[Mammen et al. (1998)] E. Mammen, J.S. Marron, B.A. Turlach, and M.P. Wand. A general framework for constrained smoothing. Preprint, 1998.

[Mammen and Thomas-Agnan (1998)] E. Mammen and C. Thomas-Agnan. Smoothing splines and shape restrictions. Preprint, 1998.

[Mammen and van de Geer (1997)] E. Mammen and S. van de Geer. Locally adaptive regression splines. *Annals of Statistics*, 1997.

[Marron and Tsybakov (1995)] J. S. Marron and A. B. Tsybakov. Visual error criteria for qualitative smoothing. *Journal of the American Statistical Association*, 90(430), 1995.

[Metzner (1997)] L. Metzner. *Facettierte nichtparametrische Regression*. PhD thesis, Department of Mathematics, University of Essen, Essen, Germany, 1997.

[Mukerjee (1988)] H. Mukerjee. Monotone nonparametric regression. *Annals of Statistics*, 16:741–750, 1988.

[Ramsay (1998)] J. O. Ramsay. Estimating smooth monotone functions. *Journal of the Royal Statistical Society B*, 60(2):365–375, 1998.

[Reinsch (1988)] C. Reinsch. Software for shape preserving spline interpolation. In M. Cox and S. Hammarling, editors, *Reliable Numerical Analysis: Proceedings of a meeting dedicated to the late J. H. Wilkonson*. Oxford University Press, 1988.

[Reinsch and Dauner (1989)] C. Reinsch and H. Dauner. An analysis of two algorithms for shape preserving cubic spline interpolation. *IMA Journal of Numerical Analysis*, 9:299–314, 1989.

[Rousseeuw (1984)] P. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79:871–880, 1984.

[Rousseeuw (1999)] P. Rousseeuw. Regression depth. Preprint, 1999.

[Schumaker (1983)] L.L. Schumaker. On shape preserving quadratic spline interpolation. *SIAM Journal of Numerical Analysis*, 20:845–864, 1983.

[Silverman (1985)] B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting (with discussion). *Journal of the Royal Statistical Society*, 47:1–52, 1985.

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

[Späth (1990)] H. Späth. *Eindimensionale Spline–Interpolations–Algorithmen*. R. Oldenbourg, 1990.

[Stoer and Burlisch (1990)] J. Stoer and R. Burlisch. *Numerische Mathematik 2*. Springer, 1990.

[Stone (1982)] C. Stone. Optimal rates of convergence for nonparametric regression. *Annals of Statistics*, 10:1040–1053, 1982.

[Turlach (1998)] Berwin Turlach. `quadprog`, a quadratic program solver. http://www.ci.tuwien.ac.at/R/src/contrib/PACKAGES.html, 1998.

[Utreras (1983)] F. I. Utreras. Smoothing data under monotonicity constraints: existence, characterization and convergence rates. *Zeitung für Numerische Mathematik*, 47:??, 1983.

[Villalobos and Wahba (1987)] M. Villalobos and G. Wahba. Inequality–constrained mulitvariate splines. *Journal of American Statistical Association*, 82:239–247, 1987.

[Wichmann (1998)] T. Wichmann. Formbewahrende regression mit splines. Master's thesis, University of Essen, 1998.

[Wright and Wegman (1978)] I. W. Wright and E. J. Wegman. Isotonic, convex, and related splines. *Annals of Statistics*, 8:1023–1035, 1978.