Optimal Selection of Regularization Parameters in Wavelet Shrinkage and Visualization using gm-waves

Master's thesis

of

Krasimir Dobrev

Bio-medical Informatics

Faculty of Mathematics and Informatics

Sofia University

developed in Narvik University College

July 20, 2006

Abstract

The problem of non-parametric statistical regression function estimation based on a large number of noisy observations is considered, both in the case of uniform deterministic knots, and uniformly distributed random knots. The smoothed (de-noised) estimator of the target function is wavelet-based, and the smoothing (de-noising) effect is achieved by shrinking the wavelet coefficients towards zero. The usual procedure for wavelet shrinkage is based on thresholding, and is efficient on smooth functions, but results in oversmoothing of points of singularity of piecewise smooth or globally non-smooth functions with fractal behaviour. For such non-smooth cases a non-threshold wavelet shrinkage procedure has been proposed by Dechevsky, Ramsay and Penev in [22] using Tikhonov regularization via a penalization criterion using Besov-norm penalty. In the present work several procedures (least squares, Besov-norm regularization, cross validation) for determining the optimal value(s) of the regularization parameter(s) are studied in detail, including the differences in performance between the case of deterministic, and that of random, knots. These procedures provide a step towards the design of adaptive, data-dependent wavelet shrinkage estimators. A representative selection of examples in the 1d case (curves) and 2d case (surfaces) is given. The visualization software developed for this purpose is in C++ and OpenGL using the library qt and the in-house libraries gm-waves and gm-lib developed at Narvik University College.

Contents

1 Introduction

1.1 Preliminaries

1.1.1 Data fitting by wavelets

I consider the following statistical models for non-parametric regression:

$$Y_i = f(x_i) + \varepsilon_i, \quad i = 1, 2, ..., N \tag{1}$$

and

$$Y_i = f(\hat{X}_i) + \varepsilon_i, \quad i = 1, 2, ..., N \tag{2}$$

where x_i are deterministic knots (in 1-dimensional case $x_i = \frac{1}{2N} + (i-1)\frac{1}{N}$) and $\hat{X}_i$ are independent, uniformly distributed on $[0,1]^d$ random variables. For the independent identically distributed (i.i.d.) errors ε_i I assume $E\varepsilon_i = 0, \quad E\varepsilon_i^2 = \delta^2$. I shall address the problem of estimating f in the above two problems by using orthonormal wavelets. Till the end of this subsection I shall refer to both x_i and $\hat{X}_i$ as X_i, since I'll make a detailed comparison between both cases in the next subsection.

This work is a preliminary stage to creating adaptive estimators that combine both well known techniques for regular smooth functions and the techniques presented here which produce better results for noncontinuous and/or non-smooth functions.

Let $\overset{\bullet}{B}{}^s_{pq}(\mathbb{R}^d)$ and $B^s_{pq}(\mathbb{R}^d)$ be respectively the homogeneous and inhomogeneous Besov spaces with metric indices p, q and smoothness index s. For $f \in B^s_{pq}$, $0 < p \leq \infty$, $0 < q \leq \infty$, $d(\frac{1}{p} - 1)_+ < s < r$,

$$f(x) = \sum_{k \in \mathbb{Z}^d} \alpha_{0k} \phi^{[0]}_{0k}(x) + \sum_{j=0}^{\infty} \sum_{k \in \mathbb{Z}^d} \sum_{l=1}^{2^d - 1} \beta^{[l]}_{jk} \phi^{[l]}_{jk}(x), \quad a.e. \quad x \in \mathbb{R}^d \tag{3}$$

holds, where $\alpha_{0k} = \left\langle \phi^{[0]}_{0k}, f \right\rangle = \int_{R^d} \phi^{[0]}_{0k}(x) \overline{f(x)} dx$, $\beta^{[l]}_{jk} = \left\langle \psi^{[l]}_{jk}, f \right\rangle$ and $\overline{z}$ is the conjugate of $z \in \mathbb{C}$. Convergence in (3) is in the quasi-norm topology of the inhomogeneous Besov space $B^s_{pq}(\mathbb{R}^d)$.

For the homogeneous space $\overset{\bullet}{B}{}^s_{pq}(\mathbb{R}^d)$ it can be shown that

$$f(x) = \sum_{j=-\infty}^{\infty} \sum_{k \in \mathbb{Z}^d} \sum_{i=1}^{2^d - 1} \beta^{[l]}_{jk} \psi^{[l]}_{jk}(x), \quad a.e. \quad x \in \mathbb{R}^d \tag{4}$$

holds modulo polynomials of total degree less than r. An equivalent (quasi)-norm of f in $B^s_{pq}(\mathbb{R}^d)$ is

$$||f||_{B^s_{pq}(\mathbb{R}^d)} = \left\{ \left(\sum_{k\in\mathbb{Z}^d} |\alpha_{0k}|^p \right)^{q/p} + \sum_{j=0}^{\infty} \left[2^{j[s+d(\frac{1}{2}-\frac{1}{p})]} \left(\sum_{k\in\mathbb{Z}^d} \sum_{l=1}^{2^d-1} |\beta^{[l]}_{jk}|^p \right)^{\frac{1}{p}} \right]^q \right\}^{\frac{1}{q}}. \tag{5}$$

Analogously, for the homogeneous space $\dot{B}^{\,s}_{pq}(\mathbb{R}^d)$ we have:

$$||f||_{\dot{B}^{\,s}_{pq}(\mathbb{R}^d)} = \left\{ \sum_{j=-\infty}^{\infty} \left[2^{j[s+N(\frac{1}{2}-\frac{1}{p})]} \left(\sum_{k\in\mathbb{Z}^n} \sum_{l=1}^{2^n-1} |\beta^{[l]}_{jk}|^p \right)^{\frac{1}{p}} \right]^q \right\}^{\frac{1}{q}}. \tag{6}$$

The tensor-product basis $\{\varphi_{jk}, \psi_{jk}\}$ is defined in [12] and [17]. The empirical wavelet estimator $\hat{f}(x)$ is defined via:

$$\hat{f}(x) = \sum_{k\in\mathbb{Z}^d} \hat{\alpha}_{0k}\phi^{[0]}_{0k}(x) + \sum_{j=0}^{\infty} \sum_{k\in\mathbb{Z}^d} \sum_{l=1}^{2^d-1} \hat{\beta}^{[l]}_{jk}\phi^{[l]}_{jk}(x), \quad a.e. \quad x \in \mathbb{R}^d, \tag{7}$$

where in the case of non-parametric regression (1) and (2):

$$\hat{\alpha}_{j_0k} = \frac{1}{N}\sum_{i=1}^{N} \varphi^{[0]}_{j_0k}(X_i)Y_i, \quad \hat{\beta}^{[l]}_{jk} = \frac{1}{N}\sum_{i=1}^{N} \psi^{[l]}_{jk}(X_i)Y_i, \tag{8}$$

The estimator $\hat{f}$ can be obtained simply by replacing the coefficients in (3) and (4) by their empirical versions, but this procedure is not as fast as the discrete wavelet transform.

In the case of non-parametric regression the shrinkage can be obtained for sample sizes N which are an exact power of *2*, by the fast discrete wavelet transform in the following way:

1. Transform data Y into the wavelet domain.
2. Shrink the empirical wavelet coefficients towards zero.
3. Transform the shrunken coefficients back to the data domain.

The methodology to estimate f is based on the principle of shrinking wavelet coefficients towards zero to remove noise, which means reducing the absolute value of the empirical wavelet coefficients $\beta^{[l]}_{jk}$.

Wavelet coefficients having small absolute value contain mostly noise. The important information at every resolution level is encoded in the coefficients on that level which have large absolute value.

One of the most important applications of wavelets (the noise reduction) has begun after observing that shrinking wavelet coefficients towards zero and then reconstructing the signal has the effect of denoising and smoothing. Donoho and Johnstone have made this observation for the first time as a result of empirical numerical experiments. Once made, this observation has found an immediate heuristic and theoretical justification in the parallel with the classical James-Stein estimator for the parametric case (see [32] and [37]). The transition from the non-parametric case (when a finite-dimensional vector parameter is being estimated) to the parametric case (when a function, generally belonging to an infinite dimensional space, is being estimated) has been immediate, thanks to the atomic decomposition of the Besov space scale by biorthonormal and orthonormal wavelets (see (3)-(6)). Indeed, if in (3)-(6) a compactly supported function f is considered, for every j the sums in k are finite; if, additionally, the sums in j in (3)-(6) are truncated (corresponding to a bounded frequency spectrum) then the non-parametric estimation problem is reduced to parametric in a natural way.

1.1.2 Types of Wavelet Shrinkage

A thresholding shrinkage rule sets to zero all coefficients with absolute values below a certain threshold level, $\lambda \geq 0$, whilst a non-thresholding rule shrinks the non-zero wavelet coefficients towards zero, without actually setting to zero any of them.

In general, shrinkage methods of threshold type are appropriate for estimating of relatively regular functions, while those of non-threshold type fit best for estimating of spatially inhomogeneous functions.

A problem when estimating functions with low regularity and fractals is that thresholding methods tend to oversmooth the curve since they are well adapted for functions which gather their value on relatively few large wavelet coefficients. Continuous fractals and functions with jumps do not fall into this case, but rather gather their value from many wavelet coefficients on infinitely many levels. Due to these facts, for such types of functions non-threshold shrinkage is appropriate.

The shrinkage estimator $\tilde{f}$ is obtained by replacing the empirical $\hat{\beta}_{jk}$ in (7) with their shrunken analogues $\tilde{\beta}_{jk}$.

1.1.3 Thresholding shrinkage

The hard and soft thresholding rules proposed by Donoho and Johnstone [27]-[29] for smooth functions are given respectively by:

$$\delta(x;\lambda) = \begin{cases} x & \text{if} \quad |x| > \lambda \\ 0 & \text{if} \quad |x| \leq \lambda \end{cases}$$
$$\text{and} \tag{9}$$
$$\delta(x;\lambda) = \begin{cases} |x| - \lambda & \text{if} \quad |x| > \lambda \\ 0 & \text{if} \quad |x| \leq \lambda \end{cases}$$

where $\lambda \in [0, \infty)$ is the threshold.

The soft thresholding rule (a continuous function) is a 'shrink' or 'kill' rule, while the hard thresholding rule (a discontinuous function) is a 'keep' or 'kill' rule. Due to the discontinuity of the hard threshold rule, the hard shrinkage estimates tend to have bigger variance and can be sensitive to small changes in the data. The soft thresholding estimates tend to have bigger bias, due to the shrinkage of large coefficients.

Gao and Bruce [30] introduced the 2-parameter firm shrinkage rule:

$$\delta(x;\lambda) = \begin{cases} \text{sgn(x)}\frac{\lambda_2 \cdot (|x| - \lambda_1)}{\lambda_2 - \lambda_1} & if \quad |x| \in (\lambda_1, \lambda_2] \\ x & if \quad |x| > \lambda_2 \\ 0 & if \quad |x| \leq \lambda_1 \end{cases} \tag{10}$$

Firm shrinkage has been specially designed to improve upon both hard and soft thresholding. By choosing appropriate thresholds (λ_1, λ_2), firm thresholding outperforms both hard and soft thresholding which are now simply two limiting cases of firm thresholding; it has all the benefits of the best of the hard and soft without the drawbacks of either. The only disadvantage of the firm shrinkage is that it requires two thresholds. This makes threshold selection problems harder and computationally more expensive for adaptive threshold selection procedures such as cross-validation.

1.1.4 Non-thresholding shrinkage

Increasing interest to the study of fractals and singularity points of functions (discontinuities of the function or its derivatives, cusps, chirps, etc.) raises the necessity of non-threshold wavelet shrinkage.

At this point, while threshold rules can be considered as well studied, non-threshold rules, on the contrary, are fairly new and the corresponding theory is so

far in an initial stage. This can be explained by the fact that traditionally only very smooth functions were being estimated.

I shall consider a new family of wavelet-shrinkage estimators of non-threshold type which are particularly well adapted for functions belonging to Besov spaces and have a full, non-sparse, vector of wavelet coefficients. The approach, proposed by Dechevsky, Ramsay and Penev in [22], parallels Wahba's spline smoothing technique (see [40]) for the case of fitting less regular curves. The purpose of the research done in [22] was to study penalized wavelet estimation in Besov spaces, proposed first by Amato and Vuza [1] in the deeper context of the theory of interpolation spaces by using Peetre's K-functional as a penalty criterion.

The regularity of the curve is discussed in terms of the size of its semi-norm in the homogenous Besov spaces, with a relatively small value of the smoothness index $s > 0$. The optimal solution of the penalization problem is in the form of a wavelet expansion whose coefficients are obtained by appropriate level- and space- dependent shrinking of the empirical wavelet coefficients. Thanks to the use of wavelets, both density and regression estimation can be treated in a somehow unified way.

The penalization model in [22] is defined via:

$$L(v, \hat{f}; B^{s_1}_{p_1 p_1}, \overset{\bullet}{B}{}^{s}_{pp}) := K_1(v, \hat{f}; (B^{s_1}_{p_1 p_1})^{p_1}, (\overset{\bullet}{B}{}^{s}_{pp}))^p)), \tag{11}$$

where $K_\eta(t, g; A, B)$ is the Peetre K-functional between the quasiseminormed abelian groups A and B of $g \in A+B$, with step $t > 0$ and parameter $\eta \in (0, \infty]$ and p,p_1,s,s_1 are such that $B^{s_1}_{p_1 p_1} \hookleftarrow B^{s}_{pp}$. Here η corresponds to the norm (quasinorm for $\eta < 1$) $(|\cdot|^\eta + |\cdot|^\eta)^{1/\eta}$, $0 < \eta \leq \infty$. The risk of the estimator is the expected value of the quasi-norm of the difference between the function and the estimator in a certain Besov space, $||f - \tilde{f}||_{p_1,q_1,s_1}$.

The general function for shrinking in Besov spaces is given by
$\tilde{\beta}_{jk} = \mu_{jk}\hat{\beta}_{jk}$, $\mu_{jk} = \mu_{jk}(v) \in [0, 1]$, and can be numerically computed (see [6]) from the following equation:

$$p_1^{\frac{1}{\eta}} \cdot (1-\mu)^{p_1 - \frac{1}{\eta}} \cdot |\hat{\beta}_{jk}|^{(p_1 - p)} = v_j \cdot 2^{\frac{j \cdot \varepsilon}{2}} \cdot p^{\frac{1}{\eta}} \cdot \mu^{p - \frac{1}{\eta}} \tag{12}$$

Here v_j is a smoothing factor [22] and the parameters in (12) are:

$$\begin{aligned} &0 < p \leq p_1 < \infty, \quad \max\left\{\tfrac{1}{p}, \tfrac{1}{p_1}\right\} < \eta < \infty \\ &\varepsilon = 2ps - 2p_1 s_1 + N(p - p_1) \quad \text{(critical-regularity index)} \end{aligned} \tag{13}$$

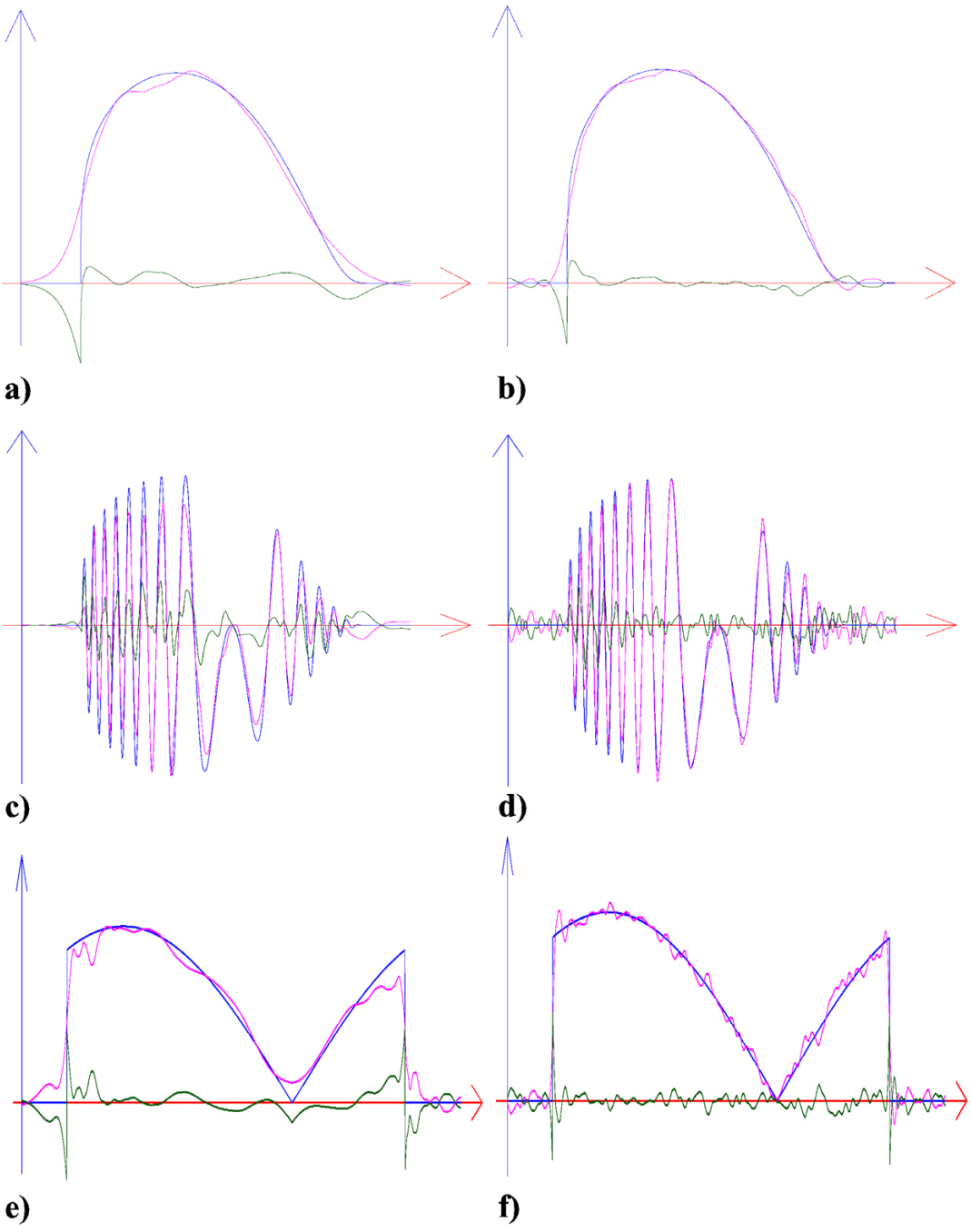


Figure 1:

In general, the equation (12) can not be solved explicitly. However, for any (j,k) the solution can be found by very quickly convergent iterations of the dyadic or Fibonacci bisection method. Moreover, in many important partial cases (12) can be solved explicitly. This is due to the fact that for the constants $A = \left(p_1^{\frac{1}{\eta}}|\hat{\beta}_{jk}|^{p_1-p}\right)^{\frac{1}{p_1-1/\eta}}$ and $B = \left(v_j \cdot 2^{\frac{j\cdot\varepsilon}{2}} \cdot p^{\frac{1}{\eta}}\right)^{\frac{1}{p_1-1/\eta}}$ (12) gets the form:

$$A\,(1-\mu) = B\mu^{\frac{p-1/\eta}{p_1-1/\eta}}, \tag{14}$$

and for $p_1 = p < \infty$, $s > s_1$, $v = t^p$, p, $t \in [0,\infty)$, becomes linear with the unique solution:

$$\mu_{jk} = \mu_j = \frac{1}{1+\left(t\cdot 2^{j(s-s_1)}\right)^{\frac{p}{p-1/\eta}}} \tag{15}$$

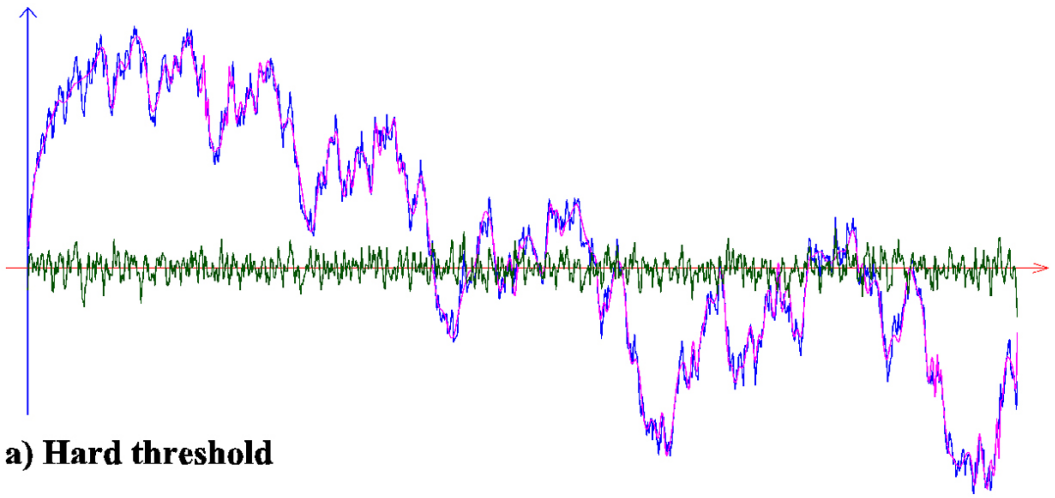
It is seen from (12)-(15) that, the closer p is to $1/\eta$, the more sensitive the model is to variations of s, s_1 and choice of v_j.

It can be seen from these cases, and some others, not shown here, that for particular cases of the metric and smoothness parameters, the Besov rules include both the non-threshold shrinkage rules and soft and hard thresholding rules. The particular combination of Besov parameters, for which the classical shrinkage estimators occur, provide additional insight into some fine properties of functions of one and several variables. For example, soft thresholding occurs in cases where p_1 is much larger than p, while ε has small to moderate values; if ε is large, then the respective Besov rule (with p_1 much larger than p) approximates the hard threshold rule. In contrast, non-threshold shrinkage occurs when $p_1 \approx p$.

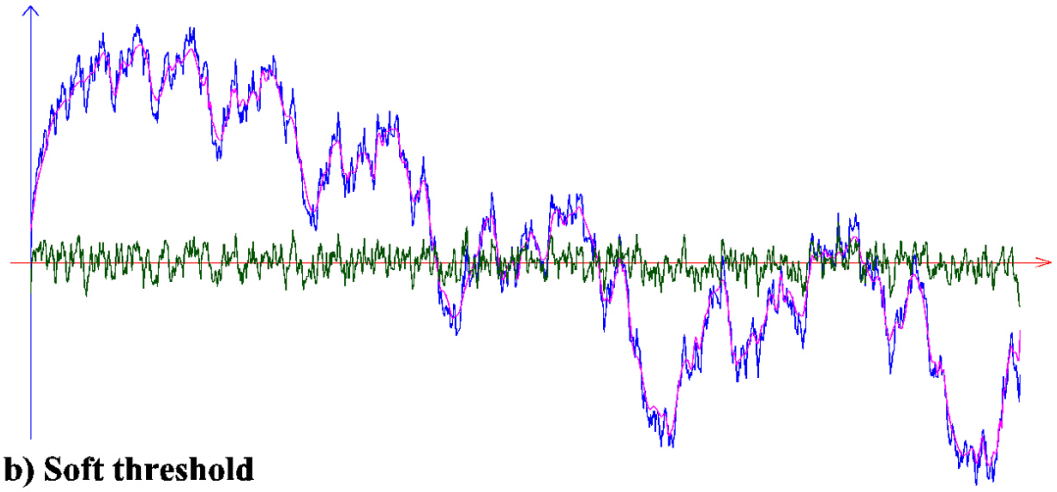
On Figure 2 is compared the distribution of approximation/ estimation error for a typical threshold shrinkage method (soft thresholding - left column) and non-threshold (Besov for $p = p_1$ - right column) for the same set of noisy observations (the non-parametric regression case) for the four functional curves considered as test examples in [6]. It is seen from these two figures that the non-adaptive threshold method oversmooths in singular points, while the non-adaptive non-threshold method overfits in regular points; an adaptive composite estimator is needed. In terms of absolute error (plotted centered around *0*), the non-threshold estimator fares better. In the case of the Weierstrass function which is non-smooth everywhere, the non-threshold estimator performs much better everywhere (see Figure 1.1.4).

Let us note also the important Lorentz-type thresholding (see [14]) in the general context of Besov spaces which was discovered in [6], Appendix B, item B10 (b), where it was shown that this type of thresholding is closely related to the

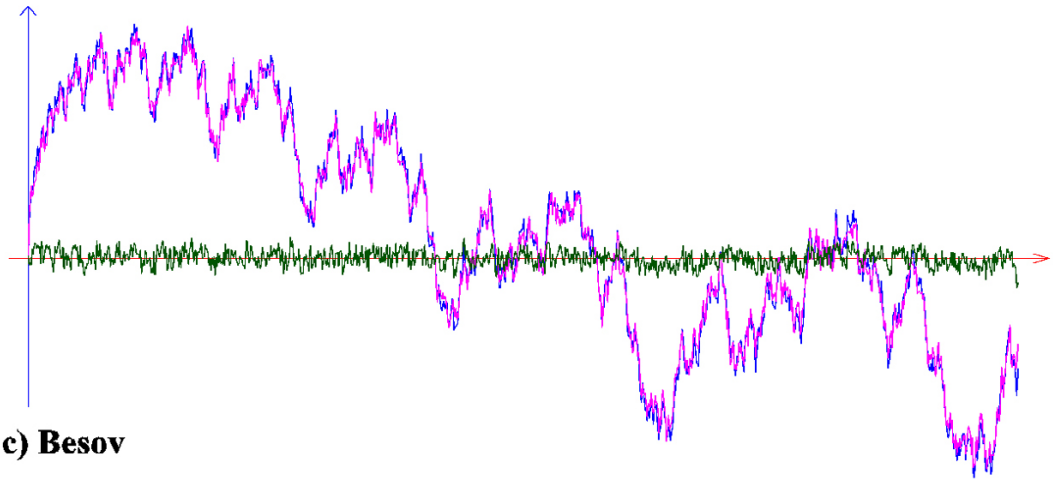
[tb]



a) Hard threshold



b) Soft threshold



c) Besov

Figure 2:

concept of decreasing rearrangement and the respective representation of Peetre's K-functional.

1.2 Differences between deterministic and i.i.d. random nodes

The difference between both models for NR (1) and (2) is that the residual error of the quadrature for is unbiased in case of independent identically distributed (i.i.d.) random nodes (2) and biased in case of deterministic uniformly distributed nodes (1).

Here we simple quadrature formulaes because the risk is decomposed into biased and variance term (see [18] and [19]) which have to be balanced in size. If I use more sophisticated quadrature formulae the biased term of the risk will decrease but the whole risk will increase due to the increase of the variance term.

The results in the theory for i.i.d. random nodes are better but in practice in order to achieve the same results I have to use much more nodes than in the case of deterministic nodes. Another practical disadvantage of i.i.d random nodes is that they are more difficult for implementation.

1.3 Topics to solve

In the context of the above my task is to solve all of the following. Most of the experiments will be performed for λ-tear $f(x) = |x|^{\lambda} e^{\frac{-x^2}{1-x^2}}, \quad x \in (-1,1), \quad f(x) = 0 \quad elsewhere \quad on \quad \mathbb{R}$.

1.3.1

Comparison between deterministic and independent identically distributed random nodes using the least square method (the B^s_{22} norm). In this task I should find optimal values v_j for each wavelet level so as the shrinkage estimator $\tilde{f}$ is obtained by replacing the empirical $\hat{\beta}_{jk}$ in (8) with their shrunken analogues

$$\tilde{\beta}_{jk} = \frac{1}{1 + 2^{2js} v_j} \hat{\beta}_{jk}$$

.

1.3.2

Finding a single optimal value v for all wavelet levels and producing the following estimator:

$$\tilde{\beta}_{jk} = \frac{1}{1 + 2^{2js}v}\hat{\beta}_{jk}$$

and visualizing the results.

1.3.3

Producing the estimator $\tilde{f}$ so as:

$$||\tilde{f}||_{B^s_{pq}(\mathbb{R}^n)} = ||f||_{B^s_{pq}(\mathbb{R}^n)}$$

and computing the average values of M consecutive executions. The values of s, p and q for the lambda tear can be found in [22]. Comparing the improvement when the norm is equalized for each wavelet level. Visualizing the results.

1.4 Cross validation

It has often been remarked in the statistical literature on smoothing that cross validation in the Lebesgue space L_2 may sometimes underestimate the value of the smoothing parameter, resulting in a tendency to overfit the curve. Analytically, this can be explained by the fact that, in general, L_2-cross validation leads to consistent estimation of the function itself, but not of its fractional derivatives. In [23] we introduce in detail the theoretical reasons for this phenomenon and to what extent the problems related to it can be overcome by considering cross validation in the Besov spaces B^σ_{22}, $\sigma > 0$, as previously proposed in [22]. (Note that L_2-cross validation is a particular case of B^σ_{22}-cross validation, with $\sigma = 0$.) If we wish to study both the case of density estimation (DE) and nonparametric regression (NR) in a unified way, we should consider penalized estimation using cross-validation. Cross-validation is based on an intuitively appealing technique for selection of the smoothing parameter in the smoothing-spline technique of statistical estimation of curves with noisy data (see Craven and Wahba [13] and Wahba [40]). This is a generalization of the penalized smoothing-spline technique of Anselone and Laurent [5] for solving deterministic smoothing problems. Many other authors have used cross-validation techniques, both ordinary and generalized, in kernel density estimation, in spline-smoothing and in wavelet approaches for non-parametric regression, density estimation and other statistical problems. We cite, in particular, Amato and Vuza [1, 2], Antoniadis [3, 4], Barron *et al* [13], Birgé and Massart [7], Cox [14] , DeVore *et al* [25], DeVore and Lucier [26], Gasser and Müller [31], Jansen *et al* [33], Nason [35], Tribouley [38], Utreras [41].

In [23] we consider, in particular, cross validation (CV) of weak, or Bowman-Rudemo type, (Bowman [8], Rudemo [36] *i.e.*, CV where the estimation of the pointwise linear interpolation functionals $L_i(f) = f(x_i)$ is replaced by estimation of the linear functionals $f \longmapsto \alpha_{j_0k}^{[0]}$ and $f \longmapsto \beta_{jk}^{[\ell]}$, where $\alpha_{j_0k}^{[0]}$ and $\beta_{jk}^{[\ell]}$ are inner product integrals defining the wavelet coefficients of f. As we shall see, the case of Bowman-Rudemo CV for NR with uniform random design can be treated quite similarly to DE. As a consequence of the random design, we shall be able to study multidimensional CV for both DE and NR; simultaneously namely, the support (supp f) or domain of f, for both DE and NR, will be assumed to be a bounded d-dimensional subset of $\mathbb{R}^d$, $d \in \mathbb{N}$.

Weak type L_2-cross validation is equivalent to applying the cross-validation rule

$$\tilde{\alpha}_{j_0k}^{[0]}\alpha_{j_0k}^{[0]} \longmapsto \frac{1}{n}\sum_{i=1}^{n}\tilde{\alpha}_{j_0k(-i)}^{[0]}\varphi_{j_0k}^{[0]}(X_i), \qquad \tilde{\beta}_{jk}^{[\ell]}\beta_{jk}^{[\ell]} \longmapsto \frac{1}{n}\sum_{i=1}^{n}\tilde{\beta}_{jk(-i)}^{[\ell]}\psi_{jk}^{[\ell]}(X_i),$$

for DE, and

$$\tilde{\alpha}_{j_0k}^{[0]}\alpha_{j_0k}^{[0]} \longmapsto \frac{1}{n}\sum_{i=1}^{n}\tilde{\alpha}_{j_0k(-i)}^{[0]}Y_i\varphi_{j_0k}^{[0]}(X_i), \qquad \tilde{\beta}_{jk}^{[\ell]}\beta_{jk}^{[\ell]} \longmapsto \frac{1}{n}\sum_{i=1}^{n}\tilde{\beta}_{jk(-i)}^{[\ell]}Y_i\psi_{jk}^{[\ell]}(X_i),$$

for NR, for every $(j,k,l) : j_0 \le j \le j_1$, supp $\varphi_{j_0k}^{[0]} \bigcap$ supp $f \neq \emptyset$, supp $\psi_{jk}^{[\ell]} \bigcap$ supp $f = \emptyset$, where the functions $\varphi_{j_0k}^{[0]}, \psi_{jk}^{[\ell]}$ are introduced in Section 2, together with j_0, j_1, k and ℓ; $\tilde{\alpha}_{j_0k}^{[0]} = \hat{\alpha}_{j_0k}^{[0]}$, $\tilde{\alpha}_{j_0k(-i)}^{[0]} = \hat{\alpha}_{j_0k(-i)}^{[0]}$, $\tilde{\beta}_{jk}^{[\ell]} = \frac{\hat{\beta}_{jk}^{[\ell]}}{1+\nu 2^{2js}}$, $\tilde{\beta}_{jk(-i)}^{[\ell]} = \frac{\hat{\beta}_{jk(-i)}^{[\ell]}}{1+\nu 2^{2js}}$,

$$\hat{\alpha}_{j_0k}^{[0]} = \frac{1}{n}\sum_{i=1}^{n}\varphi_{j_0k}^{[0]}(X_i), \qquad \hat{\beta}_{jk}^{[\ell]} = \frac{1}{n}\sum_{i=1}^{n}\psi_{jk}^{[\ell]}(X_i),$$

$$\hat{\alpha}_{j_0k(-i)}^{[0]} = \frac{1}{n-1}\sum_{x=1,x\neq i}^{n}\varphi_{j_0k}^{[0]}(X_x), \qquad \hat{\beta}_{jk(-i)}^{[\ell]} = \frac{1}{n-1}\sum_{x=1,x\neq i}^{n}\psi_{jk}^{[\ell]}(X_x),$$

for DE,and

$$\hat{\alpha}_{j_0k}^{[0]} = \frac{1}{n}\sum_{i=1}^{n}Y_i\varphi_{j_0k}^{[0]}(X_i), \qquad \hat{\beta}_{jk}^{[\ell]} = \frac{1}{n}\sum_{i=1}^{n}Y_i\psi_{jk}^{[\ell]}(X_i),$$

$$\hat{\alpha}_{j_0k(-i)}^{[0]} = \frac{1}{n-1}\sum_{x=1,x\neq i}^{n}Y_x\varphi_{j_0k}^{[0]}(X_x), \qquad \hat{\beta}_{jk(-i)}^{[\ell]} = \frac{1}{n-1}\sum_{x=1,x\neq i}^{n}Y_x\psi_{jk}^{[\ell]}(X_x)$$

for NR. (Here and henceforth we assume for the NR model, with no loss of generality, that supp f is contained in the unit hypercube of $\mathbb{R}^d$). As indicated in [22], B1, this type of cross validation rules leads to the following respectively cross validation criteria:

$$\Phi_\sigma(\nu) = \sum_k \hat{\alpha}^{[0]}_{j_0 k} + \sum_{j=j_0}^{j_1} 2^{2j\sigma} \sum_k \sum_{\ell=1}^{2^d-1} \tilde{\beta}^{[\ell]}_{jk(-i)}(\nu)^2 - \frac{2}{n} \sum_{i=1}^{n} \left[\sum_k \hat{\alpha}^{[0]}_{j_0 k(-i)} \varphi^{[0]}_{j_0 k}(X_i) + \sum_{j=j_0}^{j_1} 2^{2j\sigma} \sum_k \sum_{\ell=1}^{2^d-1} \tilde{\beta}^{[\ell]}_{jk(-i)}(\nu) \psi^{[\ell]}_{jk}(X_i) \right], \quad (16)$$

for DE, and

$$\Phi_\sigma(\nu) = \sum_k \hat{\alpha}^{[0]}_{j_0 k} + \sum_{j=j_0}^{j_1} 2^{2j\sigma} \sum_k \sum_{\ell=1}^{2^d-1} \tilde{\beta}^{[\ell]}_{jk(-i)}(\nu)^2 - \frac{2}{n} \sum_{i=1}^{n} \left[\sum_k \hat{\alpha}^{[0]}_{j_0 k(-i)} Y_i \varphi^{[0]}_{j_0 k}(X_i) + \sum_{j=j_0}^{j_1} 2^{2j\sigma} \sum_k \sum_{\ell=1}^{2^d-1} \tilde{\beta}^{[\ell]}_{jk(-i)}(\nu) Y_i \psi^{[\ell]}_{jk}(X_i) \right], \quad (17)$$

for NR, which corresponds to B^σ_{22}-cross validation, $\sigma \geq 0$. For $\sigma = 0$, (1) corresponds to the L_2-cross validation criterion considered in [22], Subsection 6.2. As we shall see, cross validation with respect to the criteria ([10], [11]) (or B^σ_{22}-cross validation, for short) can be studied in a unified way. In fact, our main results (see Section 3) are formulated simultaneously for NR and DE, and it will be informative to trace the parallels, as well as the differences, in the proofs for the two different models.

We shall formulate the results in this section simultaneously, for both NR and DE. Detailed proofs will be given for the more technically involved NR case, but a parallel discussion of the analogous arguments for DE will also be included. We recall that throughout this section it is implicitly assumed that f is compactly supported, with supp f contained within the unit hypercube Ω in $\mathbb{R}^d$, centered at the origin, that in the case of NR $X_i, \quad i = 1, \cdots, n$, are uniformly distributed on Ω, and that the 1-periodized version of (3) is being considered.

Theorem 1 *(Sufficient condition for existence of the ν-minimizer.) Consider NR and DE. Let $0 \leq \sigma \leq s' < s < r$. Assume that the penalized model is via $K_2\left(\sqrt{\nu}, f; L_2(\mathbb{R}^d), \dot{B}^s_{22}(\mathbb{R}^d)\right)$, $f \in L_\infty(\mathbb{R}^d) \bigcap B^{s'}_{\infty\infty}(\mathbb{R}^d)$, and that $j_0 \leq j_1$, $j_1 \to \infty$, and $2^{j_1} = O(n^{\frac{1}{d}})$. Assume that either $j_0 = O(1)$ and $f \notin V_j$ for any $j < j_0$ or $j_0 \to \infty$, $f \notin V_j$ for any $j \in \mathbb{Z}$ and the index s' is completely sharp. Let $\{\lambda_j\}_{j=j_0}^\infty \in \ell_2$. If one of the three mutually exclusive cases holds:*

(i) $s < \frac{d+2\sigma}{4}$ and $2^{j_1 d - 2(2s-\sigma)(j_1 - j_0) - 2j_0(s-s')} = o(n)$;

(ii) $s = \frac{d+2\sigma}{4}$ *and* $2^{2j_0(s+s'-\sigma)(j_1-j_0+1)} = o(n)$*;*

(iii) $s > \frac{d+2\sigma}{4}$ *and* $2^{j_0[d-2(s-s')]} = o(n)$*;*

then, for sufficiently large n *there exists* $\tilde{\nu} : \Psi_\sigma(\tilde{\nu}) = \min_{\nu\geq 0} \Psi_\sigma(\nu)$*, with* $0 < \tilde{\nu} < C$*, where* $\Psi_\sigma(\nu) = E\|\tilde{f}_\nu - f\|^2_{B^\sigma_{22}}$*, for both NR and DE. Moreover, if*

(i') $s < \frac{d-2(s'-2\sigma)}{4}$ *and*
$j_0 = O(1), \quad 2^{j_1[d+2s'-4(s+s'-\sigma)]} = o(n)$*, or*
$j_0 \to \infty, \quad 2^{j_1(d+2s')-4(s+s'-\sigma)(j_1-j_0)} = O(n\lambda^2_{j_0})$*, or*

(ii') $s = \frac{d-2(s'-2\sigma)}{4}$ *and*
$j_0 = O(1)$*, or*
$j_0 \to \infty, \quad 2^{j_0(s+s'-\sigma)}(j_1 - j_0 + 1) = O(n^{\frac{1}{2}}\lambda_{j_0})$*, or*

(iii') $s > \frac{d-2(s'-2\sigma)}{4}$*, and*
$j_0 = O(1)$*, or*
$j_0 \to \infty, \quad 2^{j_0(d+2s')} = O(n\lambda^2_{j_0})$*,*

and if one of the following holds

(i") $s < \sigma + \frac{d}{2}$ *and* $2^{j_1(d+2s')-2(s+s'-\sigma)(j_1-j_0)} = o(n\lambda^2_{j_0})$*;*

(ii") $s = \sigma + \frac{d}{2}$ *and* $2^{j_0(s+s'-\sigma)}(j_1 - j_0 + 1) = o(n\lambda^2_{j_0})$*;*

(iii") $s > \sigma + \frac{d}{2}$ *and* $2^{j_0(d+2s')} = o(n\lambda^2_{j_0})$*;*

then, with probability tending to 1 as $n \to \infty$*, there exists* $\nu^\star : \Phi_\sigma(\nu^\star) = \min_\nu \Phi_\sigma(\nu)$*, with* $0 < \nu^\star < C$*. The constant* $C : 0 < C < \infty$ *depends on* f, φ, ψ *for DE and also on* δ *for NR.*

Remark 1 *In view of* $\sigma < s$*, the cases (i) and (ii) can be fulfilled only when* $s < \frac{d}{2}$*. Therefore, smooth functions are being estimated under the assumption (iii). If* $\sigma = s'$*, then conditions (i - iii) and (i' - iii') coincide.*

Theorem 2 *(consistency of cross validation in* B^σ_{22}*,* $\sigma \geq 0$*.) Consider both NR and DE. Assume that* $0 < s' \leq s < r$*,* $EW_1^4 = \Delta^2 < \infty$*, that the penalized model is via* $K_2\left(\sqrt{\nu}, f; L_2(\mathbb{R}^d), \dot{B}^s_{22}(\mathbb{R}^d)\right)$*, and that* $f \in L_\infty(\mathbb{R}^d) \bigcap B^{s'}_{22}(\mathbb{R}^d)$ *holds. Let* $0 \leq \sigma \leq \frac{s'}{2}$ *and* $\sigma < \frac{d}{2}$*. Let* $j_0 \leq j_1$*, with* $j_0 \to \infty$ *and* $2^{j_1} = O(n^{\frac{1}{d}})$*,*

$$2^{-j_0} = o\left(\frac{2^{-2j_1\sigma/d}}{(j_1 - j_0 + 1)^{\frac{1}{d}}}\right) \tag{18}$$

and

$$2^{-j_0} = O\left(n^{-\frac{1}{2(d+s'-2\sigma)}}\right) \tag{19}$$

Then, for any $\nu \geq 0$,

$$\frac{\Phi_\sigma(\nu) - \Psi_\sigma(\nu) + T_n}{\Psi_\sigma(\nu)} \xrightarrow[n\to\infty]{P} 0 \tag{20}$$

holds, where

$$\begin{aligned} T_n &= \sum_k {\alpha_{j_0 k}^{[0]}}^2 - \sum_{j=j_0}^{j_1} 2^{2j\sigma} \sum_k \sum_{\ell=1}^{2^d-1} {\beta_{jk}^{[\ell]}}^2 + \frac{2(n+1)}{n} \sum_k \sum_{\ell=1}^{2^d-1} \hat{\alpha}_{j_0 k}^{[0]} \alpha_{j_0 k}^{[0]} \\ &\quad - \frac{2(n+1)}{n} \sum_k \sum_{\ell=1}^{2^d-1} {\alpha_{j_0 k}^{[0]}}^2 \end{aligned}$$

is a quantity which does not depend on ν *and*

$$T_n - \sum_k \sum_{\ell=1}^{2^d-1} {\alpha_{j_0 k}^{[0]}}^2 - \sum_{j=j_0}^{\infty} 2^{2j\sigma} \sum_k \sum_{\ell=1}^{2^d-1} {\beta_{jk}^{[\ell]}}^2 \xrightarrow[n\to\infty]{P} 0$$

holds. If, moreover, ν *depends on the sample size* n *and*

$$\nu = \nu_n = o\left(\frac{2^{-j_1\sigma + (2s - \frac{d}{2} - \sigma)}}{\sqrt{j_1 - j_0 + 1}}\right), \tag{21}$$

then the conditions $\sigma < \frac{d}{2}$ *and (18) can be removed.*

Remark 2 *When* $\sigma \geq \frac{d}{2}$*, the range of admissible* σ *in Theorem 2 can be broadened considerably, if an upper asymptotic rate for* $\nu \to 0$ *is known (see (??), cf. Remark 3).*

Corollary 1 *(Consistency in the choice of the* ν*-minimizer.) Consider both NR and DE. Under the conditions of Theorem 1, in the case* $\sigma < \frac{d}{2}$*,*

$$\frac{\Psi_\sigma(\nu^\star)}{\Psi_\sigma(\tilde{\nu})} \xrightarrow[n\to\infty]{P} 1 \tag{22}$$

holds. If the constraint $\sigma < \frac{d}{2}$ *and (??) are replaced by (??), then*

$$\frac{\Psi_\sigma(\omega^\star)}{\Psi_\sigma(\tilde{\omega})} \xrightarrow[n\to\infty]{P} 1 \tag{23}$$

still holds true, where $\omega^\star = \min(\nu^\star, \omega)$, $\tilde{\omega} = \min(\tilde{\nu}, \omega)$, *with (cf. (??))*

$$\omega = \omega_n = o\left(\frac{2^{-j_1\sigma+(2s-\frac{d}{2}-\sigma)}}{\sqrt{j_1 - j_0 + 1}}\right). \tag{24}$$

Theorem 3 *(Consistency of estimation in* B^σ_{22}, $\sigma \geq 0$, *via* $\tilde{f}_{\tilde{\nu}}$*.) Consider both NR and DE. Assume that* $0 < s' \leq s < r$, $0 \leq \sigma \leq s'$, *the penalized model is via* $K_2\left(\sqrt{\nu}, f; L_2(\mathbb{R}^d), \dot{B}^s_{22}(\mathbb{R}^d)\right)$, $f \in L_\infty(\mathbb{R}^d) \bigcap B^{s'}_{22}(\mathbb{R}^d)$, *and that* $j_0 \leq j_1$, $j_1 \to \infty$. *Assume also that* $f \notin V_j$, *where* $j < j_0$ *if* $j_0 = O(1)$ *and* $j \in \mathbb{Z}$ *if* $j_0 \to \infty$. *If* $j_0 = O(1)$ *and* $\nu = \nu_n$ *is bounded, then the condition* $\nu \to 0$ *is necessary for* $\Psi_\sigma(\nu) \to 0$ *to hold as* $n \to \infty$. *Moreover, if* $j_0 = O(1)$, $\sigma < s'$, *and* $2^{j_1} = o\left(n^{\frac{1}{d+2\sigma}}\right)$ *holds, then* $\nu \to 0$ *is also a sufficient condition for* $\Psi_\sigma(\nu) \to 0$ *to hold as* $n \to \infty$. *If* $2^{j_1} = o\left(n^{\frac{1}{d+2\sigma}}\right)$, *with* $j_0 \to \infty$ *and* $\sigma \leq s'$, *then only boundedness of* $\nu = \nu_n$ *is already sufficient for* $\Psi_\sigma(\nu)$ *to hold as* $n \to \infty$. *In particular, if* s' *is sharp, then the above necessity claim is true for* $\nu = \tilde{\nu}_n$. *The above sufficiency claims are also true for* $\nu = \tilde{\nu}_n$.

Remark 3 *The constraint* $2^{j_1} = o\left(n^{\frac{1}{d+2\sigma}}\right)$ *in Theorem 3 can be relaxed considerably if a lower asymptotic rate for* $\nu \to 0$ *is known (cf. also Remark 2).*

Theorem 4 *(Asymptotic behaviour of* $\tilde{\nu}$*.) Consider both NR and DE. Let* $0 < s' < s < r$. *Assume that the penalized model is via* $K_2\left(\sqrt{\nu}, f;\ L_2(\mathbb{R}^d), \dot{B}^s_{22}(\mathbb{R}^d)\right)$, $f \in L_\infty(\mathbb{R}^d) \bigcap B^{s'}_{22}(\mathbb{R}^d)$ *and that* $j_0 \leq j_1$, $j_1 \to \infty$ *and* $2^{j_1} = o\left(n^{\frac{1}{d}}\right)$. *Assume also that either* $j_0 = O(1)$ *and* $f \notin V_j$ *for any* $j < j_0$, *or* $j_0 \to \infty$, $f \notin V_j$ *for any* $j \in \mathbb{Z}$ *and the index* s' *is completely sharp. Finally, let* $0 \leq \sigma < 2s$, *and assume that one of the three cases (i - iii) of Theorem 1 holds. Then,* $\tilde{\nu} \to 0$ *as* $n \to \infty$.

Corollary 2 *(Consistency in probability of estimation in* B^σ_{22}, $\sigma \geq 0$, *via* $\tilde{f}_{\nu^\star}$*.) Consider both NR and DE. Assume that* $\sigma \leq \frac{s'}{2}$, $2^{j_1} = o\left(n^{\frac{1}{d+2\sigma}}\right)$, *the conditions of Theorem 4 are fulfilled, and (??) holds. Then,*

$$\Psi_\sigma(\omega^\star) \xrightarrow[n\to\infty]{P} 0 \tag{25}$$

holds. Moreover, if $\sigma < \frac{d}{2}$ *and (??) is fulfilled, then also*

$$\Psi_\sigma(\nu^\star) \xrightarrow[n\to\infty]{P} 0 \tag{26}$$

holds.

Remark 4 *The condition $f \in L_\infty(\mathbb{R}^d)$ in all of the above results is essential only when $0 < s \leq \frac{d}{2}$, due to the Sobolev embedding within the scale of Besov spaces (see also [22], Remark* **??** *and B21).*

Remark 5 *In the partial case $\sigma = 0$ (L_2-cross validation), when $d = 1$, Theorems 1 - 4 and Corollaries 1, 2 yield essential improvements of Theorems 5 - 8 and Corollary 2 in [**?**], Theorem 3 in [38], as well as of Theorems 1 - 3 in [20].*

1.5 The software

1.5.1

The task is to develop a software system capable of applying the non-thresholding shrinkage algorithms for the non-parametric regression problem. Data size is always taken an exact power of 2 in order to use the fast discrete wavelet transform, and since the methods described above will produce results of practical use only when combined with thresholding techniques in adaptive estimators. For the wavelet transform I use gm-waves, and for visualization I use gm-lib combined with qt. Gm-waves and gm-lib are developed in Narvik University College and can be used freely for educational purposes. I have written the software completely in C++, using templates where needed. Number one priority was to make the source code as reusable as possible. Only the key moments in the source code are commented, instead I have preferred to write self-explanatory code, increasing its readability this way.

1.5.2 About gm-waves

Gm-waves is a new software system developed as a wavelet library. It consists of two parts: a computational library for wavelet algorithms and a visualization library. Both libraries are written in template-based C++. The visualization library is OpenGL-based and uses the display hierarchy of gm-lib - a geometric modelling library developed at Narvik University College. The user interface of gm-waves relies on QT, a complete C++ application development framework.

The current version of gm-waves features most of the wavelets and algorithms described in [34], including also the lifting scheme, fast lifted wavelet transforms, the Daubechies-Lagarias algorithm, wavelet packets and best basis selection algorithms, filter calculation functions, pursuit methods and others. Among the new features are a new algorithm for N-dimensional discrete wavelet transform (DWT) using recursive templates in C++, a 2-dimensional discrete wavelet transform for

processing of large data sets using a Graphics Processing Unit (GPU) shading algorithm etc.

1.6 Generating random numbers

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

John von Neumann

Performing experiments with noisy data I encountered the problem of generating noise or random numbers. There are two primary methods for generating random numbers: computer generated pseudo random numbers, and random numbers generated by some physical process like radio-active decay or the thermal phenomena. I shall introduce examples for both methods here. The prefix pseudo is used to emphasize on the fact that it's not a truly random sequence of numbers but a sequence behaving like one.

1.6.1 Computer generated pseudo random generators

Most of the computer generated pseudo random number generators (PRNGs) produce an uniformly distributed sequences. Since any PRNG runs on a deterministic computer, it is necessarily a deterministic algorithm. If the PRNG uses fixed amount of memory it will produce a periodic sequence. Given a sufficient number of iterations the generator will revisit a previous internal state, after which it will repeat forever. That is why the output of such generator will always have one property that a true random sequence can never have: guaranteed periodicity. The primary PRNGs classes are: linear congruential generators, lagged Fibonacci generators, linear feedback shift registers and generalized feedback shift registers.

Linear congruential generators

Linear congruential generators represent one of the oldest and best studied PRNGs. However their properties are far from ideal. Let R_i is the i^{th} random number. Then the sequence $\{R_i\}_0^\infty$ is generated by the following formula:

$$R_{i+1} = (A \times R_i + B) mod M$$

It is known that these random generators have a period at most M. It is also known that for good choice of A and B the linear congruential generator will have a period M if $M = 2^n$, the period will be $M - 1$ if M is prime and so on.

While the linear congruential generators are capable of producing decent pseudo random numbers in theory, in practice they are extremely sensitive to the choice of A, B and M. In "Numeric recipes in C" William Press, Saul Teukolsky, William Vetterling and Brian Flannery present a linear congruential generator with

$$A = 1664525, B = 1013904223, M = 2^{32}$$

although the authors say: *"These are not particularly good choices for A and B, though they are not gross embarrassments by themselves."*.

Linear congruential generators should never be used for cryptography purposes because of the well known cryptanalysis techniques. They should be used with caution in Monte Claro algorithms since the period of the generator may be lesser then the data size needed.

Lagged Fibonacci generator

A Lagged Fibonacci generator is another example of a pseudo random number generator. This class of random number generator is designed as an improvement on the "standard" linear congruential generator. These are based on a generalization of the Fibonacci sequence ($F_{n+2} = F_{n+1} + F_n$). The sequence of the Lagged Fibonacci is generated by the following formula:

$$R_i = (R_{i-j} \star R_{i-k}) mod M$$

where $0 < j < k$

M is usually a power of 2 $M = 2^n$, often 2^{32} or 2^{64}. The $\star$ operator denotes some binary operation. This may be either addition, subtraction, multiplication, or the bitwise arithmetic exclusive-or operator (XOR). The theory of this type of generator is rather complex, and it may not be sufficient simply to choose random values for j and k. These generators also tend to be very sensitive to initialization. Generators of this type store the last k values.

If the $\star$ operation is addition, then the generator is described as an Additive Lagged Fibonacci Generator, if it is multiplication, the generator is called a Multiplicative Lagged Fibonacci Generator, and if it is XOR, the generator is called a Two-tap Generalized Feedback Shift Register. The Mersenne twister algorithm is a variation on a Two-tap Generalized Feedback Shift Register.

Lagged Fibonacci generators have a maximum period of $(2^k - 1)\frac{m}{2}$ if the $\star$ operation is addition or subtraction, and $(2^k - 1)$ the $\star$ operation is XOR. Popular pairs of j and k are: $j = 7, k = 10$, $j = 5, k = 17$, $j = 24, k = 55$, $j = 65, k = 71$, $j = 128, k = 159$. A large list of possible values for j and k is listed in Donald Knuth's volume 2 of "The Art of Computer Programming".

Linear feedback shift register

A linear feedback shift register is a shift register whose input bit is a linear function of its current state. Since the only linear functions of single bits are exclusive-or (XOR) and inverse exclusive-or (inverse-XOR), the input bit is calculated by the XOR of some bits of the overall shift register value. A linear feedback shift register with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long period.

Linear feedback shift register works in the following way. It keeps n bits stored. At each step a XOR over several of the stored bit and the result is the feedback which is inserted in the beginning of the stored bit sequence. The last bit of the sequence is removed and used for an output.

Linear feedback shift register have long been used as a pseudo-random number generator for use in stream ciphers (especially in military cryptography), due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed outputs. However their output is completely linear, leading to fairly easy cryptanalysis.

Blum-Blum-Shub pseudo random number generator Blum Blum Shub is a pseudo random number generator proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub. The output sequence is produced by the function:

$$R_{i+1} = R_i^2 mod M$$

where $M = p \times q$ is the product of two large primes p and q. At each step of the algorithm, some output is derived from R_i. The output is commonly either the bit parity of R_i or one or more of the least significant bits of R_i.

The generator is not appropriate for use in simulations, only for cryptography, because it is not very fast. However, it has an unusually strong security proof, which relates the quality of the generator to the computational difficulty of integer factorization. If integer factorization is difficult (as is suspected) then Blum-Blum-Shub generator with large M will have an output free from any nonrandom patterns that can be discovered with any reasonable amount of calculation. This makes it as secure as other encryption technologies tied to the factorization problem, such as RSA encryption.

Mersenne twister

The Mersenne twister is a pseudo random number generator developed in 1997 by Makoto Matsumoto and Takuji Nishimura. It provides for fast generation of very high quality pseudo random numbers, having been designed specifically to

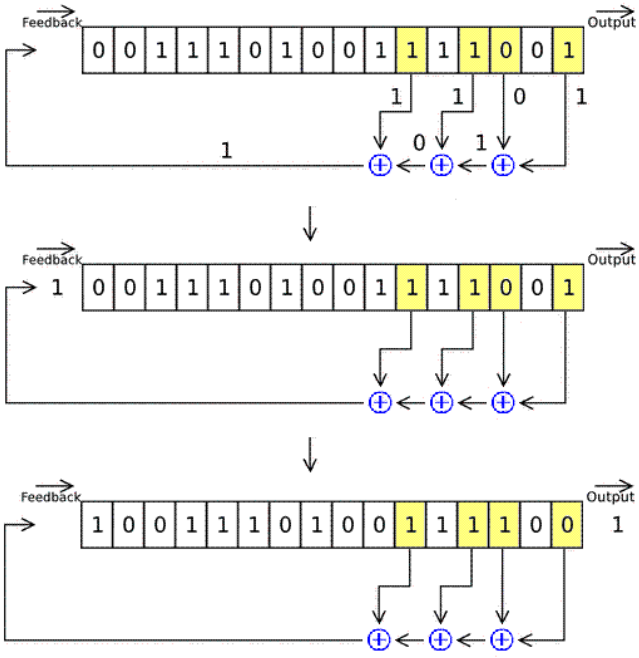


Figure 3: Linear Feedback Shift Register

rectify many of the flaws found in older algorithms. The newer and more commonly used algorithm is the Mersenne Twister MT 19937. MT 19937 has the following properties: a colossal period of $2^{19937} - 1$, faster than all but the most statistically unsound generators and passes numerous tests for statistical randomness, including the stringent Diehard tests. This period explains the origin of the name: it is a Mersenne prime, and some of the guarantees of the algorithm depend on internal use of Mersenne primes. In practice, there is little reason to use larger ones.

The algorithm itself is a twisted generalised feedback shift register. The "twist" is a transformation which assures equidistribution of the generated numbers in 623 dimensions. Unlike Blum Blum Shub, the algorithm in its native form is not suitable for cryptography. Observing a sufficient number of iterates allows one to predict all future iterates. Combining the Mersenne twister with a hash cures this problem but slows down generation. For many other applications, however, the Mersenne twister is fast becoming the random number generator of choice.

1.6.2 Hardware random generators

A hardware random number generator is an apparatus that generates random numbers from a physical process. Because the outcome of quantum-mechanical events cannot in principle be predicted, they are the gold standard for randomness. Some quantum phenomena used for random number generation include:

- A nuclear decay radiation source, detected by a Geiger counter attached to a PC.
- Photons traveling through a semi-transparent mirror. The mutually exclusive events (reflection - transmission) are detected and associated to "0" - "1" bit values.

Another source of randomness is the thermal phenomena. Thermal phenomena are easier to detect but they can be open to attack by lowering the temperature of the system, though most systems will stop operating at temperatures (150 K) low enough to reduce noise by a factor of two. Some thermal phenomena used include:

- Thermal noise from a resistor, amplified to provide a random voltage source.
- Avalanche noise generated from an avalanche diode or Zener breakdown noise from a reverse-biased zener diode.
- Atmospheric noise, detected by a radio receiver attached to a PC.

"Move the mouse to produce randomness"

In the absence of quantum or thermal noise, other phenomena that tend to be random can be used. With several such sources, combined carefully, enough entropy can be collected for the occasional creation of cryptographic keys. The advantage is that this needs no special hardware. The primary source of randomness used here is the precise timing of the interrupts caused by mechanical input/output devices, such as keyboards and disk drives, which explains the quote above. This last approach must be implemented carefully and may be subject to attack if it is not. For instance, the generator built into the Linux kernel may be vulnerable.

1.6.3 PRNG implementations

Random numbers in C

The ANSI C present a built in linear congruential generator which should be used with caution. In the header `<stdlib.h>` the following macros and function are defined:

```
#define RAND_MAX 0x7fff
int __cdecl rand(void);
void __cdecl srand(unsigned int _Seed);
```

In case a float pseudo random number in $[0, 1]$ is needed, it can be get by an expression like:

```
x = rand() / ( RAND_MAX + 1.0 );
```

In case an integer pseudo random number in $[1..max]$ is needed, it should be get by the expression:

```
x = 1 + (int) ( ( rand() * max ) / ( RAND_MAX + 1.0 ) );
```

rather than the most commonly used:

```
x = 1 + rand() % ( max + 1 );
```

which relies on low order bits and should never be used. Similarly the user should never split a number returned by `rand()` for to different porpoises, two different numbers should be used instead.

The main problem is the short period of the presented generator. For example if this generator is used in a Monte-Carlo algorithm and a set if 1000000000 points are examined, what actually happens is examination of the same 32767 points about 30000 times. However if the ANSI C PRNG is the choice it should be seeded with execution dependent variable in order to get different sequence in each execution, for example:

```
srand( (unsigned int) time( 0 ) );
```

which initializes the PRNG with a time dependent value.

Random numbers in C++

C++ inherits the ANSI C PRNG and it is the only standard compiler and platform independent generator that can be used. However there are a couple of compiler and/or platform dependent PRNG is available.

Random numbers in GNU C++

The GNU C++ Library defines a couple of PRNGs. The two classes `RNG` and `Random` are used together to generate a variety of random number distributions. A distinction must be made between random number generators, implemented by class `RNG`, and random number distributions. A random number generator produces a series of randomly ordered bits. These bits can be used directly, or cast to other representations, such as a floating point value. A random number generator should produce a uniform distribution. A random number distribution, on the other hand, uses the randomly generated bits of a generator to produce numbers from a distribution with specific properties. Each instance of `Random` uses an instance of class `RNG` to provide the raw, uniform distribution used to produce the specific distribution. Several instances of Random classes can share the same instance of RNG, or each instance can use its own copy. For more information see the User's Guide to the GNU C++ Library

Random numbers in Unix/Linux

There is a standard function `random()` in unix/linux distributions, more information available in manual pages (RANDOM(3)). The function is defined in `<stdlib.h>`

```
long int random(void);
```

The `random()` function uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to RAND_MAX. The period of this random number generator is very large, approximately $16 \times (2^{31} - 1)$ and is the better choice of Unix/Linux users.

GNU Scientific Library

The GNU Scientific Library is a free software available for download under GNU Public License. The library provides a large collection of random number generators which can be accessed through a uniform interface. Environment variables allow you to select different generators and seeds at runtime, so that you can easily switch between generators without needing to recompile your program. Each instance of a generator keeps track of its own state, allowing the generators to be used in multi-threaded programs. Additional functions are available for transforming uniform

random numbers into samples from continuous or discrete probability distributions such as the Gaussian, log-normal or Poisson distributions. For more information: The official GNU Scientific Library page.

Random numbers in Java

Java provides a toolkit for generating random numbers, in the class `java.util.Random`. An instance of this class is used to generate a stream of pseudo random numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. The algorithms implemented by class Random use a protected utility method that on each invocation can supply up to 32 pseudorandomly generated bits. Many applications will find the random method in class Math simpler to use. The `java.lang.Math` class provides a:

```
public static double random()
```

method that returns a double value in $[0.0, 1.0)$. Returned values are chosen pseudo randomly with uniform distribution from that range. For more information see Javadoc.

Random numbers in .NET platform

The .NET platform provides methods for generating pseudo random numbers in the `System.Random` class. The current implementation of the Random class is based on Donald E. Knuth's subtractive random number generator algorithm. To generate a cryptographically secure random number a class that is derived from `System.Security.Cryptography.RandomNumberGenerator` should be used such as `System.Security.Cryptography.RNGCryptoServiceProvider`. For more information see MSDN.

1.6.4 Generating i.i.d. random variables

Let ε_i are independent identically distributed random variables in $[a, b]$. Our task is to determine a and b so as

$$E\varepsilon_i = e$$

$$E(\varepsilon_i - E\varepsilon_i)^2 = \delta^2$$

where e and δ^2 are given constants.

We will generate random variables with $E\varepsilon_i = 0$, if we want to achieve $E\varepsilon_i = e \neq 0$ then we will use the sequence $\{\varepsilon_i + e\}$ which will consist of random numbers in $[a + e, b + e]$, where $[a, b]$ is the domain for ε_i.

If ε_i are identically distributed random variables in $[a, b]$ then

$$E\varepsilon_i = \frac{a+b}{2} = 0 \Rightarrow a = -b$$

First we will determine δ^2 with given $[a, b]$

$$\delta^2 = E(\varepsilon_i - E\varepsilon_i)^2 = E\varepsilon_i^2 = x^2 p(x)$$

where x is a point in $[a, b]$. So in order to compute δ^2 we have to compute the mean value of the function $f(x) = x^2$ in $[a, b] = [-b, b]$

$$\delta^2 = \frac{\int_a^b x^2 dx}{b-a} = \frac{2\int_0^b x^2 dx}{b-(-b)} = \frac{\int_0^b x^2 dx}{b} = \frac{x^3|_0^b}{3b} = \frac{b^3 - 0^3}{3b} = \frac{b^2}{3}$$

$$\delta^2 = \frac{b^2}{3} \Rightarrow b = \sqrt{3\delta^2}$$

In conclusion, in order to produce identically distributed random variables with given variance δ^2, the interval

$$[-\sqrt{3\delta^2}, \sqrt{3\delta^2}]$$

should be used. To produce identically distributed random variables with given variance δ^2 and expectation e, the interval

$$[-\sqrt{3\delta^2} + e, \sqrt{3\delta^2} + e]$$

should be used.

1.6.5 The rejection method.

We will present a powerful technique for generating random variables with a given distribution. The method rely on an simple geometric fact. Given the graph of the probability distribution ($p(x)$) the area below the graph in any range $[a, b]$ for x corresponds to the probability of generation x in that range. For the algorithm we choose another function $f(x)$, so that: $0 \leq p(x) \leq f(x)$, and $\int_{-\infty}^{\infty} f(x)dx < \infty$. The later is possible because $\int_{-\infty}^{\infty} p(x)dx = 1$. $f(x)$ is called comparison function. The rejection method is the following:

Choose an uniformly distributed point (X, Y), so as $0 \leq Y \leq f(x)$.

If Y is above $p(x)$ $(Y > p(x))$, reject the point.

Else accept the point and output X as a result.

It is obvious that the accepted points are uniform in the accepted area, so that their X values have the desired distribution. Another obvious fact is that the fraction of rejected points is exactly:

$$\frac{\int_{-\infty}^{\infty} p(x)dx}{\int_{-\infty}^{\infty} f(x)dx}$$

This method rises the problem of choosing an uniformly distributed point below $f(x)$. One possible solution is the transformation method that can be found in William Press, Saul Teukolsky, William Vetterling and Brian Flannery's "Numeric Recipes in C".

Another simple and worse performing solution is to choose:

$$f(x) = \max_{x \in [a,b]} p_1(x), x \in [a,b]$$

$$f(x) = 0, x \notin [a,b]$$

where $p_1(x)$ is the normalized $p(x)$, so as:

$$\int_a^b p_1(x)dx = 1$$

$[a,b]$ should be determined so as $p(x) < \varepsilon, x \notin [a,b]$, where $\varepsilon > 0$ is a insignificant probability for the current experiment. Determining an uniformly distributed point (X,Y) is simply determining uniformly distributed $X \in [a,b]$ and determining uniformly distributed $Y \in [0, \max_{x \in [a,b]} p_1(x)]$.

2 Main results

2.1 Decomposing the risk into approximation and variance term

2.1.1 1-dimensional case

Let $f(x) : [0,1] \to \mathbb{R} \in C_0$, $\varepsilon_i \quad i = 1..n$ are independent identically distributed i.i.d. random variables. $E\varepsilon_i = 0$ and $E\varepsilon_i^2 = \delta^2$. Let $g(x) : \mathbb{R} \to \mathbb{R}$ known Riemman integrable.

Let x_i are deterministic knots ($x_i = \frac{1}{2N} + (i-1)\frac{1}{N}$) and $\hat{X}_i$ are independent, uniformly distributed on $[0,1]$ random variables. I will discuss both cases (1) and (2) together and stress on the differences. X_i is either x_i or $\hat{X}_i$.

Let $\hat{Y}_i = f(X_i) + \varepsilon_i$.

$$F = \int_0^1 f(x)g(x)dx, \quad \hat{F_n} = \frac{1}{n}\sum_1^n \hat{Y}_i g(X_i) \tag{27}$$

$$Risk = E(F - \hat{F_n})^2 = E((F - E\hat{F_n}) + (E\hat{F_n} - \hat{F_n}))^2 =$$

$$E(F - E\hat{F_n})^2 + 2E((F - E\hat{F_n})(\hat{F_n} - \hat{F_n}) + E(\hat{F_n} - \hat{F_n})^2 =$$

$$(F - E\hat{F_n})^2 + 2((F - E\hat{F_n})E(E\hat{F_n} - \hat{F_n}) + E(E\hat{F_n} - \hat{F_n})^2 =$$

$$(F - E\hat{F_n})^2 + 2((F - E\hat{F_n})(E\hat{F_n} - E\hat{F_n}) + E(E\hat{F_n} - \hat{F_n})^2 =$$

$$(F - E\hat{F_n})^2 + E(E\hat{F_n} - \hat{F_n})^2$$

$(F - E\hat{F_n})^2$ - approximation term.

$E(E\hat{F_n} - \hat{F_n})^2$ - variance term.

$$E\hat{F_n} = E\frac{1}{n}\sum_1^n \hat{Y}_i g(X_i) =$$

$$E\frac{1}{n}\sum_1^n (f(X_i) + \varepsilon_i)g(X_i) =$$

$$\frac{1}{n}\sum_1^n Ef(X_i)g(X_i) + \frac{1}{n}\sum_1^n E\varepsilon_i g(X_i) =$$

$$\frac{1}{n}\sum_{1}^{n} Ef(X_i)g(X_i)$$

$$E\hat{F_n} = E\frac{1}{n}\sum_{1}^{n} f(X_i)g(X_i) \tag{28}$$

if $X_i = \hat{X}_i$ then $E\hat{F}_n = F$, since the expectation of the quadrature form above is exactly the value of the integral when the nodes are i.i.d random variables. From this fact follows that the approximation term is 0 in case of i.i.d nodes.

In the case of deterministic nodes ($x_i = \frac{1}{2N} + (i-1)\frac{1}{N}$) from (28) follows:

$$approximation \quad term = (F - E\hat{F_n})^2 =$$

$$\left(\int_0^1 f(x)g(x)dx - E\frac{1}{n}\sum_{1}^{n} f(X_i)g(X_i)\right)^2 =$$

$$\sum_{1}^{n}\left(\int_{X_i-\frac{1}{2n}}^{X_i+\frac{1}{2n}} f(x)g(x)dx - f(X_i)g(X_i)dx\right)^2 =$$

$$\sum_{1}^{n}\left|\int_{X_i-\frac{1}{2n}}^{X_i+\frac{1}{2n}} f(x)g(x)dx - f(X_i)g(X_i)dx\right|^2 \leq$$

$$\sum_{1}^{n}\left(\int_{X_i-\frac{1}{2n}}^{X_i+\frac{1}{2n}} |f(x)g(x)dx - f(X_i)g(X_i)|\,dx\right)^2 \leq \frac{const}{n^\beta}$$

$$approximation \quad term \leq \frac{const}{n^\beta} \tag{29}$$

assuming $|g(x_i)^2 - g(x)^2| \leq const|x - x_i|^\alpha$

$$variance \quad term = E(E\hat{F_n} - \hat{F_n})^2 =$$

$$E(\frac{1}{n}\sum_{1}^{n} f(X_i)g(X_i) + \frac{1}{n}\sum_{1}^{n}(f(X_i) + \varepsilon_i)g(X_i))^2 =$$

$$\frac{1}{n^2}E(\sum_{1}^{n}\varepsilon_i g(X_i))^2 = \frac{1}{n^2}E\sum_{1}^{n}(\varepsilon_i g(X_i))^2 =$$

$$\frac{1}{n^2}\sum_{1}^{n} E\varepsilon_i^2 g(X_i)^2 = \frac{\delta^2}{n}\frac{1}{n}\sum_{1}^{n} g(X_i)^2 =$$

for sufficiently large n:

$$variance \quad term \asymp \frac{\delta^2}{n}||g||_{L_2}^2 \tag{30}$$

from (29) and (30) follows that in case of deterministic nodes:

$$risk \asymp \frac{const}{n^{\beta}} + \frac{\delta^2}{n}||g||^2_{L_2} \tag{31}$$

2.1.2 1-dimensional case using wavelets

Using the assumptions in 2.1.1. Let φ be an univariate scaling function (father wavelet), and let ψ be the corresponding univariate wavelet (mother wavelet), obtained by multiresolution analysis (see, e.g., [16], [15]), so that for any $j_0 \in \mathbb{Z}$ the functions $\varphi_{j_0k_1}(x_1) = 2^{\frac{j_0}{2}}\varphi(2^{j_0}x_1 - k_1)$, $\psi_{jk_1}(x_1) = 2^{\frac{j}{2}}\psi(2^j x_1 - k_1)$, $x_1 \in \mathbb{R}$, $j = j_0, j_0+1, \cdots$, $k_1 \in \mathbb{Z}$, form an orthonormal basis of $L_2(\mathbb{R})$

Let

$$\int_{-\infty}^{\infty}\varphi_{j_0k}(x)dx = 1, \quad \int_{-\infty}^{\infty}\varphi_{j_0k}(x)^2dx = 1,$$
$$\int_{-\infty}^{\infty}\psi_{jk}(x)dx = 0 \quad and \quad \int_{-\infty}^{\infty}\psi_{jk}(x)^2dx = 1$$

The decomposition of f is:

$$f(x) = \sum_{k=-\infty}^{\infty}\alpha_{j_0k}\varphi_{j_0k}(x) + \sum_{j=j_0}^{j_1}\sum_{k=-\infty}^{\infty}\beta_{jk}\psi_{jk}(x) \tag{32}$$

ant the approximation is:

$$\hat{f_n}(x) = \sum_{k=-\infty}^{\infty}\hat{\alpha_{j_0k}}\varphi_{j_0k}(x) + \sum_{j=j_0}^{\infty}\sum_{k=-\infty}^{\infty}\hat{\beta_{jk}}\psi_{jk}(x) \tag{33}$$

$$risk = E\int_0^1 (f(x) - \hat{f_n}(x))^2dx =$$

$$E\left(\sum_{k=-\infty}^{\infty}(\alpha_{j_0k} - \hat{\alpha_{j_0k}})^2 + \sum_{j=j_0}^{j_1}\sum_{k=-\infty}^{\infty}(\beta_{jk} - \hat{\beta_{jk}})^2 + \sum_{j=j_1}^{\infty}\sum_{k=-\infty}^{\infty}\beta_{jk}^2\right) =$$

$$\sum_{k=-\infty}^{\infty}E(\alpha_{j_0k} - \hat{\alpha_{j_0k}})^2 + \sum_{j=j_0}^{j_1}\sum_{k=-\infty}^{\infty}E(\beta_{jk} - \hat{\beta_{jk}})^2 + \sum_{j=j_1}^{\infty}\sum_{k=-\infty}^{\infty}\beta_{jk}^2$$

$$risk = \sum_{k=-\infty}^{\infty}E(\alpha_{j_0k} - \hat{\alpha_{j_0k}})^2 + \sum_{j=j_0}^{j_1}\sum_{k=-\infty}^{\infty}E(\beta_{jk} - \hat{\beta_{jk}})^2 + \sum_{j=j_1}^{\infty}\sum_{k=-\infty}^{\infty}\beta_{jk}^2 \tag{34}$$

considering 2.1.1, let $g(x) = \varphi_{j_0K}$

$$< f(x), g(x) >= \int_{-\infty}^{\infty} f(x)g(x)dx =$$

$$\sum_{k=-\infty}^{\infty} \alpha_{j_0 k} \int_{-\infty}^{\infty} \varphi_{j_0 k} \varphi_{j_0 K} dx + \sum_{j=j_0}^{\infty} \sum_{k=-\infty}^{\infty} \beta_{jk} \int_{-\infty}^{\infty} \psi_{jk} \varphi_{j_0 K} dx = \alpha_{j_0 K}$$

and

$$< \hat{f(x)}, g(x) >= \int_{-\infty}^{\infty} \hat{f(x)} g(x) dx =$$

$$\sum_{k=-\infty}^{\infty} \hat{\alpha_{j_0 k}} \int_{-\infty}^{\infty} \varphi_{j_0 k} \varphi_{j_0 K} dx + \sum_{j=j_0}^{\infty} \sum_{k=-\infty}^{\infty} \hat{\beta_{jk}} \int_{-\infty}^{\infty} \psi_{jk} \varphi_{j_0 K} dx = \hat{\alpha_{j_0 K}}$$

from (31) follows:

$$E(< f(x), g(x) > - < \hat{f(x)}, g(x) >)^2 = (\alpha_{j_0 K} - \hat{\alpha_{j_0 K}})^2 \leq \frac{const}{n_{j_0 K}^{\sigma}} + \frac{\delta^2}{n} \qquad (35)$$

again considering 2.1.1, let $g(x) = \psi_{JK}$

$$< f(x), g(x) >= \int_{-\infty}^{\infty} f(x) g(x) dx =$$

$$\sum_{k=-\infty}^{\infty} \alpha_{j_0 k} \int_{-\infty}^{\infty} \varphi_{j_0 k} \psi_{JK} dx + \sum_{j=j_0}^{\infty} \sum_{k=-\infty}^{\infty} \beta_{jk} \int_{-\infty}^{\infty} \psi_{jk} \psi_{JK} dx = \beta_{JK}$$

and

$$< \hat{f(x)}, g(x) >= \int_{-\infty}^{\infty} \hat{f(x)} g(x) dx =$$

$$\sum_{k=-\infty}^{\infty} \hat{\alpha_{j_0 k}} \int_{-\infty}^{\infty} \varphi_{j_0 k} \psi_{jK} dx + \sum_{j=j_0}^{\infty} \sum_{k=-\infty}^{\infty} \hat{\beta_{jk}} \psi_{jk} \int_{-\infty}^{\infty} \psi_{JK} dx = \hat{\beta_{JK}}$$

from (31) follows:

$$E(< f(x), g(x) > - < \hat{f(x)}, g(x) >)^2 = (\beta_{JK} - \hat{\beta_{JK}})^2 \leq \frac{const}{n_{JK}^{\sigma}} + \frac{\delta^2}{n} \qquad (36)$$

from (34)-(36) follows:

$$risk = \sum_{k=-\infty}^{\infty} E(\frac{const}{n_{j_0 k}^{\sigma}} + \frac{\delta^2}{n})^2 + \sum_{j=j_0}^{j_1} \sum_{k=-\infty}^{\infty} (\frac{const}{n_{jk}^{\sigma}} + \frac{\delta^2}{n}) + \sum_{j=j_1}^{\infty} \sum_{k=-\infty}^{\infty} \beta_{jk}^2 \qquad (37)$$

The parameters j_0 and j_1 are selected here as in [24], j_0 must be such that$N^{1/(2s+d} \leq 2^{j_0} < 2N^(1/(2s+d))$ and j_1 must be such: $2^{dj_1} = o(N)$ (we choose $2^{dj_1} \sim 2N/ln \quad N$ as in [24]). There are other ways for determining j_0 and j_1 for example cross validation which we will not discuss here.

2.2 Minimizing the risk (levelwise)

Let us consider the model for non-thresholding shrinkage from [22], on page 314 for deterministic nodes (1), v is the regularization parameter ($vge0$):

$$E(\tilde{\beta_{jk}} - \beta_{jk})^2 =$$

$$\frac{1}{(1+v2^{2js})^2}\left[(\bar{\beta_{jk}} - \beta_{jk})^2 + \frac{\delta^2}{n}\frac{1}{n}\sum_{i=1} n\psi_{jk}(X_i)^2 - 2v2^{2js}\beta_{jk}(\bar{\beta_{jk}} - \beta_{jk}) + v^2 2^{4js}\beta_{jk}^2\right] \tag{38}$$

where $\bar{\beta_{jk}}$ are the values of the wavelet coefficients of f approximated by a quadrature form. The wavelet coefficients of the estimator are shrunk using the following formulae:

$$\tilde{\beta_{jk}} = \frac{\hat{\beta_{jk}}}{1+2^{2js}v} \tag{39}$$

If we consider (38) levelwise we have:

$$\sum_k E(\tilde{\beta_{jk}} - \beta_{jk})^2 =$$

$$\sum_k \frac{1}{(1+v_j 2^{2js})^2}\left[(\bar{\beta_{jk}} - \beta_{jk})^2 + \frac{\delta^2}{n}\frac{1}{n}\sum_{i=1}^{n} \psi_{jk}(X_i)^2 - 2v_j 2^{2js}\beta_{jk}(\bar{\beta_{jk}} - \beta_{jk}) + v_j^2 2^{4js}\beta_{jk}^2\right] \tag{40}$$

Now we will calculate the exact values for v_j in order to minimize $\sum_{k=-\infty}^{\infty} E(\tilde{\beta_{jk}} - \beta_{jk})^2$.

Lets

$$2^{2js} = a_j$$

$$\sum_{k=-\infty}^{\infty} (\bar{\beta_{jk}} - \beta_{jk})^2 = D_j^2$$

$$\frac{\delta^2}{n}\frac{1}{n}\sum_{k=-\infty}^{\infty}\sum_{i=1}^{n} \psi_{jk}(X_i)^2 = \Delta_j^2$$

$$\sum_{k=-\infty}^{\infty} \beta_{jk}(\bar{\beta_{jk}} - \beta_{jk}) = C_j$$

$$\sum_{k=-\infty}^{\infty} \beta_{jk}^2 = B_j^2$$

Lets consider $\sum_{k=-\infty}^{\infty} E(\tilde{\beta_{jk}} - \beta_{jk})^2$ as a function of v:

$$F_j(v) = \sum_{k=-\infty}^{\infty} E(\tilde{\beta_{jk}} - \beta_{jk})^2 = \frac{1}{(1+a_j v)^2}(D_j^2 + \Delta_j^2 - 2a_j v C_j + a_j^2 v^2 B_j^2) \quad (41)$$

In order to compute the minimum of $F_j(v)$ we have to solve the following equation:

$$F_j(v)' = 0$$

$$\left[\frac{1}{(1+a_j v)^2}(D_j^2 + \Delta_j^2 - 2a_j v C_j + a_j^2 v^2 B_j^2)\right]' = 0$$

$$\frac{1}{(1+a_j v)^4}[(-2a_j C_j + 2a_j^2 v B_j^2)(1+a_j v)^2 - 2a_j(1+a_j v)(D_j^2 + \Delta_j^2 - 2a_j v C_j + a_j^2 v^2 B_j^2)]' = 0$$

$$\frac{1}{(1+a_j v)^4}[(-2a_j C_j + 2a_j^2 v B_j^2)(1+a_j v)^2 - 2a_j(1+a_j v)(D_j^2 + \Delta_j^2 - 2a_j v C_j + a_j^2 v^2 B_j^2)] = 0$$

$$\frac{2a_j}{(1+a_j v)^3}[(-C_j + a_j v B_j^2)(1+a_j v) - D_j^2 - \Delta_j^2 + 2a_j v C_j + a_j^2 v^2 B_j^2)] = 0$$

$$\frac{2a_j}{(1+a_j v)^3}[-C_j + a_j v B_j^2 - a_j v C_j + a_j^2 v^2 B_j^2 - D_j^2 - \Delta_j^2 + 2a_j v C_j - a_j^2 v^2 B_j^2)] = 0$$

$$\frac{2a_j}{(1+a_j v)^3}[-C_j + a_j v B_j^2 - D_j^2 - \Delta_j^2 + a_j v C_j] = 0$$

$$\frac{2a_j}{(1+a_j v)^3}[a_j v(B_j^2 + C_j) - (C_j + D_j^2 + \Delta_j^2)] = 0$$

Since $a_j > 0$ and $v \geq 0 \quad \Rightarrow \quad \frac{2a_j}{(1+a_j v)^3} > 0 \quad \Rightarrow$

$$\frac{2a_j}{(1+a_j v)^3}[a_j v(B_j^2 + C_j) - (C_j + D_j^2 + \Delta_j^2)] = 0 \quad iff \quad v = \frac{(C_j + D_j^2 + \Delta_j^2)}{a_j(B_j^2 + C_j)} = v^* \Rightarrow$$

$F_j(v)$ has only one extremum and since $F_j(v^*)'' > 0$ $F_j(v)$ has a minimum in v^*, except $\pm\infty$, and since v^* may be negative $\Rightarrow$

$$v_j = \max(0, \frac{(C_j + D_j^2 + \Delta_j^2)}{a_j(B_j^2 + C_j)}) \quad (42)$$

$v_j = 0$ means that wavelet coefficients on level j should not be shrunk, a positive value of v_j means that corresponding wavelet coefficients are going to be shrunk. After implementing this model, we found out that the results are worse than expected for relatively smooth function (having few singularity points). The problem is that we rely on the fact that noise is usually situated on finer levels and we try to shrink them more with the parameter a_j but from (39) and (42) $\Rightarrow$

$$\tilde{\beta_{jk}} = \frac{\hat{\beta_{jk}}}{1 + 2^{2js}v} = \frac{\hat{\beta_{jk}}}{1 + \frac{C_j + D_j^2 + \Delta_j^2}{B_j^2 + C_j}}$$

which does not take into account neither the level index j, nor the smoothness index s of the Besov space B_{pq}^s. We achieve better smoothing if we choose of j_0 and j_1 as in previous section.

Let us consider the above model for i.i.d random nodes. The difference from the above is that the expectation of the inner product integral f approximation by a quadrature form is exact: $E\bar{\beta_{jk}} = \beta_{jk}$, there for,

$$D_j^2 = \sum_{k=-\infty}^{\infty} (\bar{\beta_{jk}} - \beta_{jk})^2 = 0$$

$$C_j = \sum_{k=-\infty}^{\infty} \beta_{jk}(\bar{\beta_{jk}} - \beta_{jk}) = 0$$

So the minimum of $F_j(v)$ will be at:

$$v^* = \frac{\Delta_j^2}{a_j B_j^2}$$

which is always non negative $\Rightarrow$

$$v_j = \frac{\Delta_j^2}{a_j B_j^2} \tag{43}$$

in the case of i.i.d. nodes.

One possible way for evaluating β_{jk} and $\bar{\beta_{jk}}$ is cross validation. For details see appendix ([23]).

2.3 Minimizing the risk (same regularization parameter for all leves)

Now the task is to compute a single regularization parameter for all wavelet levels. Since it is very difficult task to compute analytically the value of the regularization

parameter for which the risk is minimal we will approach this task in a computational way. Most of the cases the risk has only one extremum, which is minimum, so a simple way for computing it is to start from a positive value x and search in the interval $[0, 2x]$. At each step if the search interval is $a, b]$ we check the values of the risk in a, $\frac{a+b}{2}$ and b, if $risk(a) > risk(\frac{a+b}{2}) > risk(b)$ then we change the search interval to $[\frac{a+b}{2}, 2b]$. If $risk(a) < risk(\frac{a+b}{2}) < risk(b)$ then we change the search interval to $[a, \frac{a+b}{2}]$. In the last case $risk(a) > risk(\frac{a+b}{2}) < risk(b)$ we have several subcases: if $risk(a) > risk(\frac{a+b}{4}) > risk(\frac{a+b}{2}) < risk(\frac{3(a+b)}{4}) < risk(b)$ then we change the search interval to $[\frac{a+b}{4}, \frac{3(a+b)}{4}]$, if $risk(a) > risk(\frac{a+b}{4}) > risk(\frac{a+b}{2}) > risk(\frac{3(a+b)}{4}) < risk(b)$ then we change the search interval to $[\frac{a+b}{2}, b]$ and finally if $risk(a) > risk(\frac{a+b}{4}) < risk(\frac{a+b}{2}) < risk(\frac{3(a+b)}{4}) < risk(b)$ then we change the search interval to $[a, \frac{a+b}{2}]$. Note that only one step extends the search interval and it can be performed only finite number of times since the function has only one minimum in finite point. Moreover it can be shown that this minimum tends to 0 when the sample size $N \to \infty$.

Since regularization parameters have values less than 2 in almost all the cases, another solution is to choose some set of points in $[0, 2]$ and use a genetic algorithm for searching the minimal value of the risk, which will find the minimum (or a value very close to the minimum) for relatively small number of iterations.

2.4 Equalizing $||\tilde{f}||_{B^s_{pq}}$ to $||f||_{B^s_{pq}}$

The idea of this method is to equalize the norms of the exact function and the estimator in Besov space B^s_{pq}. For this method we need to have some knowledge for the Besov norm of the exact function. Both homogeneous and inhomogeneous norm can be used. For homogeneous Besov norm (6) we have:

$$||f||_{\overset{\bullet}{B}{}^{s}_{pq}(\mathbb{R}^d)} = \left\{ \sum_{j=-\infty}^{\infty} \left[2^{j[s+n(\frac{1}{2}-\frac{1}{p})]} \left(\sum_{k\in\mathbb{Z}^n} \sum_{l=1}^{2^n-1} |\beta_{jk}^{[l]}|^p \right)^{\frac{1}{p}} \right]^q \right\}^{\frac{1}{q}}.$$

In this case it can be shown that the dependence in the regularization parameter is monotone, which means that we can use again the technique described in the previous sections.

If we equalize the norms levelwize we don't achieve improvement because again the smoothing index of the Besov space is not taken into consideration.

2.5 Cross validation

For proofs of all theorems in 1.4 see [23].

3 The Software

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies."

The idea of the software system is to demonstrate the models discussed above. All the software developed by me lies in `mainwindow.ui.h` and `Random.h`. During the development process I fixed some performance issues in gm_lib and reported them to the gm_lib team.

During the development process I emphasized on producing decent output rather than ease of use. The output can be exported in vector formats such as pdf, encapsulated postscript (eps) and post script. Output formats with loss such as jpeg or gif were avoided on purpose.

The software design follows standard qt practice, and although it's a little less object oriented than a common graphical user interface application, in my opinion it's designed very well. I tried to make the source code as scalable as possible.

3.1 User Interface

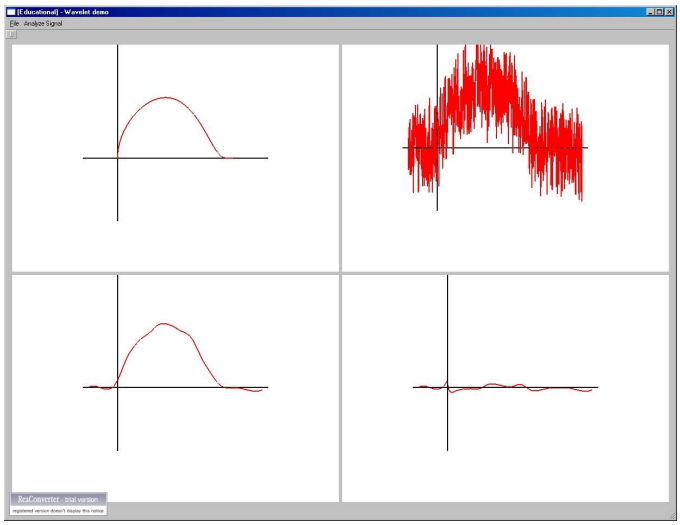


Figure 4: The Wavelet demo application

On figure (3.1) is shown the standard window of the application I developed. There are 4 separate windows containing:

- The exact function for the experiment on the top left.
- The noisy function (the exact function with added white noise) for the experiment on the top right.
- The estimated function after the experiment on the bottom left.
- The the error (the difference between exact and estimated functions) on the bottom right.

The application can print any of the windows into some vector graphics format from `file > print` menu. Some predefined function are build in as well as the possibility to load any data from a file. The file for 1d data starts with `data_size_1d`, followed by `long_data_size_1d`, and then by minimum and maximum value of the argument of the function. Next in the file there should be `long_data_size_1d` numbers representing the function values in each point (deterministic nodes are considered). All numbers should be separated by at least one white space. Analogously for 2d data the file starts with `data_size_1d` followed by `long_data_size_2d` and minimum and maximum values for both function parameters. Next in the file there should be `long_data_size_2d x long_data_size_2d` numbers representing the function values in each point (deterministic nodes are considered). All numbers should be separated by at least one white space. Only `data_size_1d` (or `data_size_2d x data_size_2d` are considered for approximating the function, long data sizes are used for computing an approximation of functions exact coefficients.

3.2 Mathematical model

The models discussed in the previous section are implemented in `Analyse1D()` and `Analyse2D()` function in `mainform` class. There is an enumeration `ExperimentType` defining which of the models will be used. Its possible values are:

- `Experiment_V` - computing one regularization parameter for all wavelet levels. The exact coefficients of the function ($\beta_j k$) are computed like the approximate coefficients ($\bar{\beta_j} k$) using DWT by using the `long_data_size_1d` or `long_data_size_2d` points respectively.
- `Experiment_V_j` - same as above but computing separate regularization parameter for each wavelet level.

- `Experiment_Norm` - computing one regularization parameter for all wavelet levels so as B^s_{pq} norm of the approximated function is equal to the B^s_{pq} norm of the exact function. One and the same data size is used for computing both norms.

- `Experiment_Norm_j` - same as above but computing separate regularization parameter for each wavelet level.

For all computations I use Daubechies wavelets of degree 6.0 or 7.0. Most of the experiments were preformed for the λ-tear function since it's known exactly in which Besov spaces it belongs to [22] and it is a very well studied function so I can compare my results with other researches.

3.3 mainwindow.ui.h

The file is used for declaring the functions that process user input via keyboard/mouse. The main functionality of this project is in the following functions:

```
void init()
```

Initialize all the variables used in the program.
effects: All internal variables are properly set.
returns: void

```
void SetupCamera( int dimension, double x_min, double x_max, double y_min,

double y_max, double z_min, double z_max, bool use_curve )
```

This function is used to set up the camera in each window. Camera is pointed at the center of the curve/surface. If the data is one dimensional the camera is positioned right above the center of the embracing rectangle. If the data is two dimensional the camera is positioned so as
argument: `int dimension` - dimension of the analyzed object.
argument: `double x_min` - minimal x value of the analyzed object.
argument: `double x_max` - maximum x value of the analyzed object.
argument: `double y_min` - minimal y value of the analyzed object.
argument: `double y_max` - maximum y value of the analyzed object.
argument: `double z_min` - minimal z value of the analyzed object.
argument: `double z_max` - maximum z value of the analyzed object.

argument: `bool use_curve` - defines whether to turn the camera towards the curve or the surface in each window.

requires: $1 \leq$ `dimension` ≤ 2

requires: `x_min` $\leq$ `x_max`

requires: `y_min` $\leq$ `y_max`

requires: `z_min` $\leq$ `z_max`

returns: void

`void Clear()`

The function clears all the objects inserted in the scenes.

effects: `mycurve[ i ] = NULL` for `i = 0..number_of_windows`.

effects: `mysurf[ i ] = NULL` for `i = 0..number_of_windows`.

returns: void

`void LoadSignal1D( func1d func, double t_min, double t_max )`

Loads and displays one dimensional data for analysis. Display the result of the analysis.

argument: `func1d func` - pointer to a function to be analyzed. `func1d` is defined as: `long double (*func1d )( long double )`

argument: `double t_min` - minimal value of function parameter.

argument: `double t_max` - maximal value of function parameter.

effects: Data is loaded in `data_1d`, `long_data_1d` and `noisy_data_1d`.

returns: void

`void Analyse1D()`

Analyzes one dimensional data.

requires: Data is loaded in `data_1d`, `long_data_1d` and `noisy_data_1d`. The result of the analysis is in `reconstructed_data_1d`.

effects: The result of the analysis is in `reconstructed_data_1d`.

`void LoadSignal2D( func2d func, double u_min, double u_max, double v_min, double v_max )`

Loads and displays two dimensional data for analysis. Display the result of the analysis.

argument: `func2d func` - pointer to a function to be analyzed. `func1d\end` is defined as: `long double ( *func2d )( long double, long double )`

argument: `double u_min` - minimal value of function first parameter.

argument: `double u_max` - maximal value of function first parameter.
argument: `double v_min` - minimal value of function second parameter.
argument: `double v_max` - maximal value of function second parameter.
effects: Data is loaded in `data_2d`, `long_data_2d` and `noisy_data_2d`.
returns: void

`void Analyse2D( bool update_points = true )`

Analyzes two dimensional data.
requires: Data is loaded in `data_2d`, `long_data_2d` and `noisy_data_2d`. The result of the analysis is in `reconstructed_data_2d`.
effects: The result of the analysis is in `reconstructed_data_2d`.

`void LoadImage( std::string filename )`

Loads and displays image for analysis. Display the images returned by the analysis.
argument: `std::string filename` - filename to be loaded.
requires: `filename` exists.
effects: The image is loaded in `}image[ i ]` for `i = 0..number_of_windows`.
returns: void

`void AnalyzeImage()`

Analyzes an image loaded in `image[ 0 ]`. The function is called by `LoadImage`.
requires: `image[ i ].width = image[ 0 ].width` and `image[ i ].height = image[ 0 ].height` for `i = 0..number_of_windows`
requires: `image[ i ].width = image[ 0 ].height = 2^{n}` for some n
effects: `image[ 1 ]` contains the noisy image.
effects: `image[ 2 ]` contains the denoised image.
effects: `image[ 3 ]` contains the difference between `image[ 0 ]` and `image[ 2 ]`.
returns: void

`void Redraw()`

Forces redraw of all the windows.
returns: void

`void ResizeToFullscreen()`

Resizes the current window to full screen or restores normal state if the window is already in full screen. The function is activated by double click.
returns: void

3.4 Random.h

`class Random`

The class defines a Linear congruential psudo-random number generator. The values for A, B and M are taken from "Numeric recipes in C" by William Press, Saul Teukolsky, William Vetterling and Brian Flannery. This method is chosen because of its quickness, simplicity and long enough period, the later is the main reason I don't rely on the random generator build in C. Since the class is derived from the abstract class `AbstractRandomGenerator`, it's very easy to define other random generators and use them if needed. The `Random\end` class has the following public methods:

```
Random( const long double& variance = 0.0, const long double& expectation = 0.0 );
```

A default constructor creating an generator that has certain variance and expectation. The minimum and maximum are calculated as in [...]. The constructor seeds the random generator with a execution dependent variable. If a generator producing one and the same sequence is needed, it should be seeded with a reasonable value after creation.

argument: `const long double& variance` variance of the generated random numbers. Default value is `0.0`.

argument: `const long double& expectation` expectation of the generated random numbers. Default value is `0.0`.

effects: The variance and expectation are set to the given values and minimum and maximum values of the interval in which the random numbers will be generated are computed.

```
Random( const long double& minimum, const long double& maximum, int dummy )
```

A constructor creating an generator that has certain minimum and maximum. The variance and expectation are calculated as in [...]. The constructor seeds the random generator with a execution dependent variable. If a generator producing one and the same sequence is needed, it should be seeded with a reasonable value after creation.

argument: `const long double& minimum` left boundary of the interval in which the random numbers will be generated.

argument: `const long double& maximum` right boundary of the interval in which the random numbers will be generated.

argument: `int dummy` no effect argument. It is necessary to have a dummy value for distinguishing this constructor from the default one, which can also take 2 `const long double&` arguments.

requires: $minimum \leq maximum$.

effects: The interval is set to $[minimum, maximum]$ and variance and expectation are computed.

`virtual ~Random()`

A destructor. Has no effect.

`Random( const Random& right )`

A copy constructor.

effects: Internal state is copied.

`Random& operator=( const Random& right )`

Assignment operator.

effects: Internal state is copied.

returns: Copied value.

`virtual long double operator()()`

The method is derived from `AbstractRandomGenerator`.

effects: Next random number is generated.

returns: Floating point random number in $[minimum, maximum]$.

`virtual long long Rand()`

The method is derived from `AbstractRandomGenerator`.

effects: Next random number is generated.

returns: Integer random number in $[0, 2^{32})$.

`virtual long long Rand( long long max )`

The method is derived from `AbstractRandomGenerator`.

effects: Next random number is generated.

returns: Integer random number in $[0, max)$.

`virtual void Seed( long long seed )`

The method is derived from from `AbstractRandomGenerator`.
effects: The generator is seeded with *seed*.
requires: $seed \geq 0$.
returns: void.

`virtual long double Expectation() const`

The method is derived from `AbstractRandomGenerator`.
guarantees: Internal state is not changed.
returns: The expectation of the generated floating point numbers.

`virtual long double Variance() const`

The method is derived from `AbstractRandomGenerator`.
guarantees: Internal state is not changed.
returns: The variance of the generated floating point numbers.

`virtual long double Minimum() const`

The method is derived from `AbstractRandomGenerator`.
guarantees: Internal state is not changed.
returns: The minimum possible value of the generated floating point numbers.

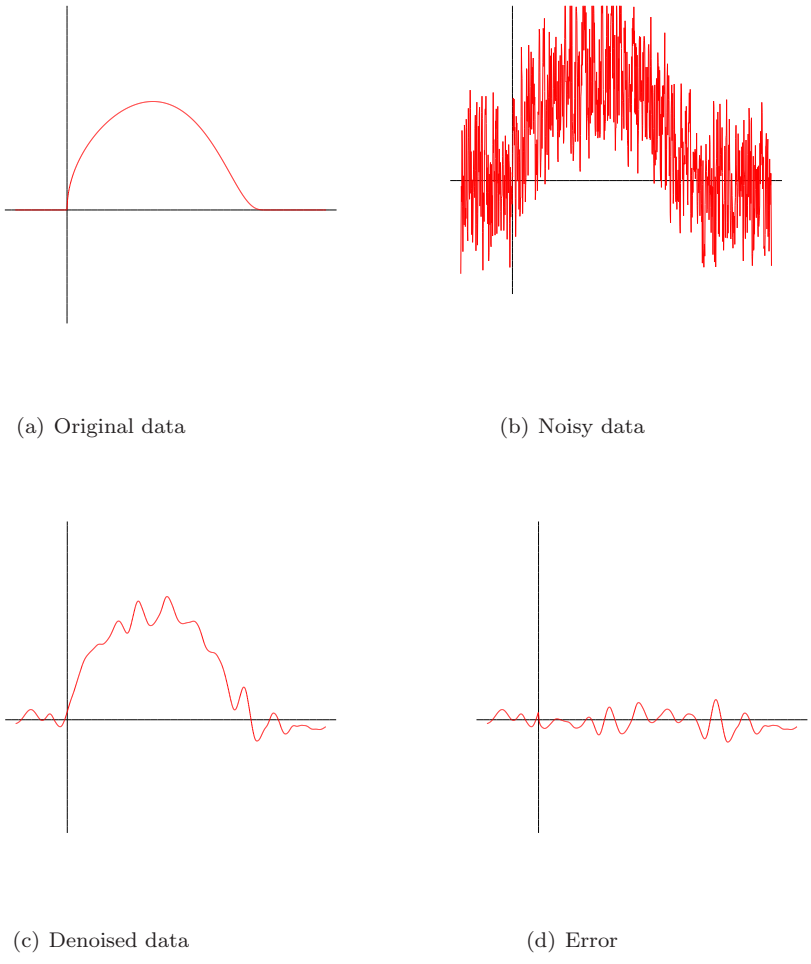
`virtual long double Maximum() const`

The method is derived from `AbstractRandomGenerator`.
guarantees: Internal state is not changed.
returns: The maximum possible value of the generated floating point numbers.

4 Numerical and graphical results

4.1 Minimizing the risk

4.1.1 $\lambda = 0.5$, $\delta^2 = \frac{1}{12}$ (corresponding to white noise in $[-0.5, 0.5]$)



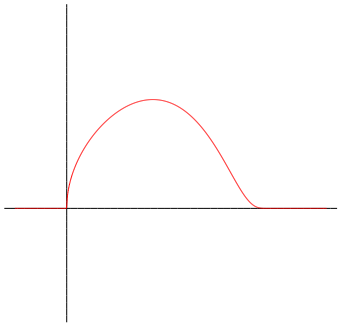
(a) Original data

(b) Noisy data

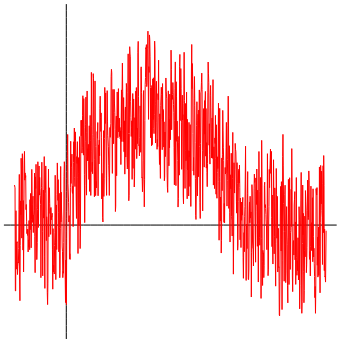
(c) Denoised data

(d) Error

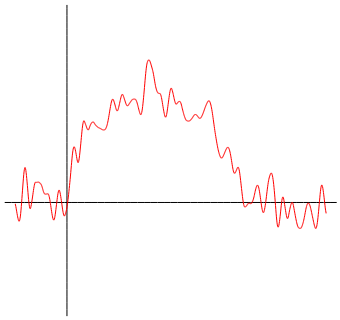
Figure 5: Data size = 1024, s = 10, $j_0 = 3$, $j_1 = 10$



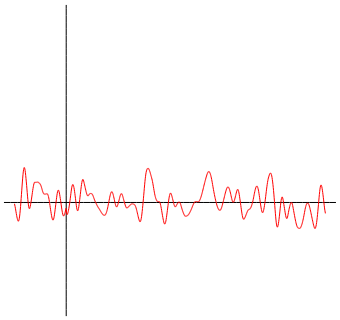
(a) Original data



(b) Noisy data

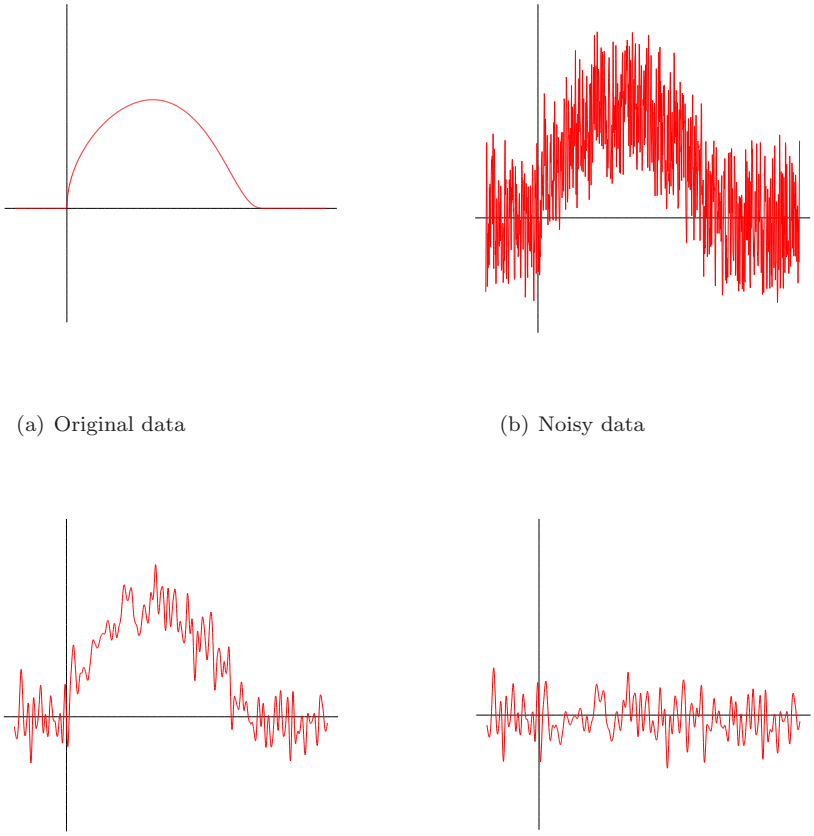


(c) Denoised data



(d) Error

Figure 6: Data size = 1024, s = 8, $j_0 = 3$, $j_1 = 10$



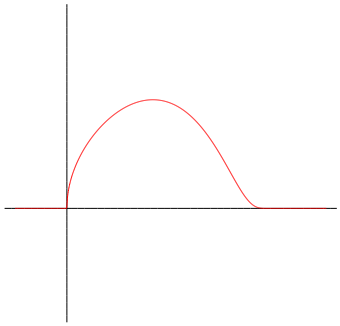
(a) Original data

(b) Noisy data

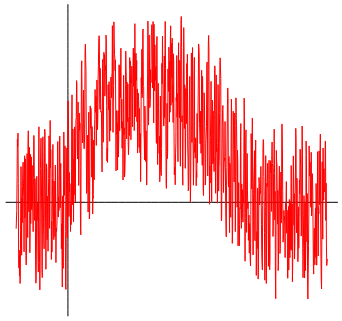
(c) Denoised data

(d) Error

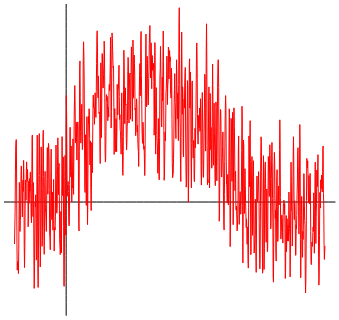
Figure 7: Data size = 1024, s = 5, $j_0 = 3$, $j_1 = 10$



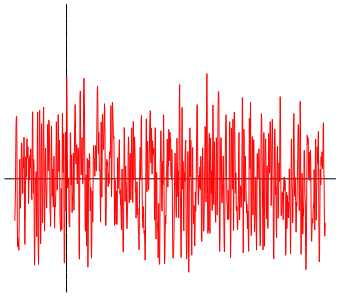
(a) Original data



(b) Noisy data



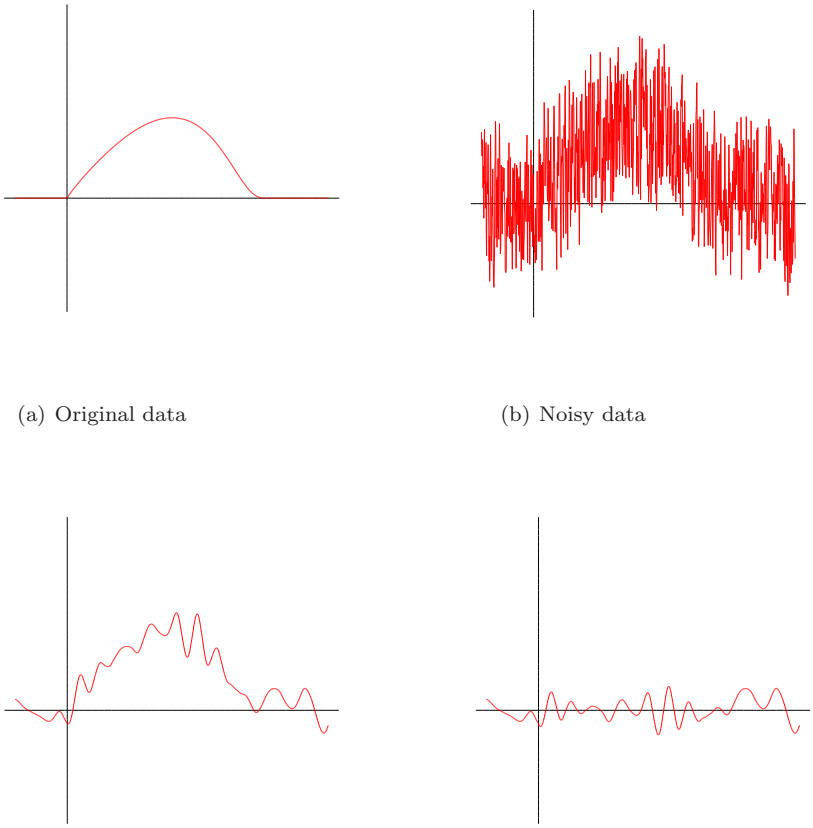
(c) Denoised data



(d) Error

Figure 8: Data size = 1024, s = 3, $j_0 = 3$, $j_1 = 10$

4.1.2 $\lambda = 0.9$, $\delta^2 = \frac{1}{12}$ (corresponding to white noise in $[-0.5, 0.5]$)



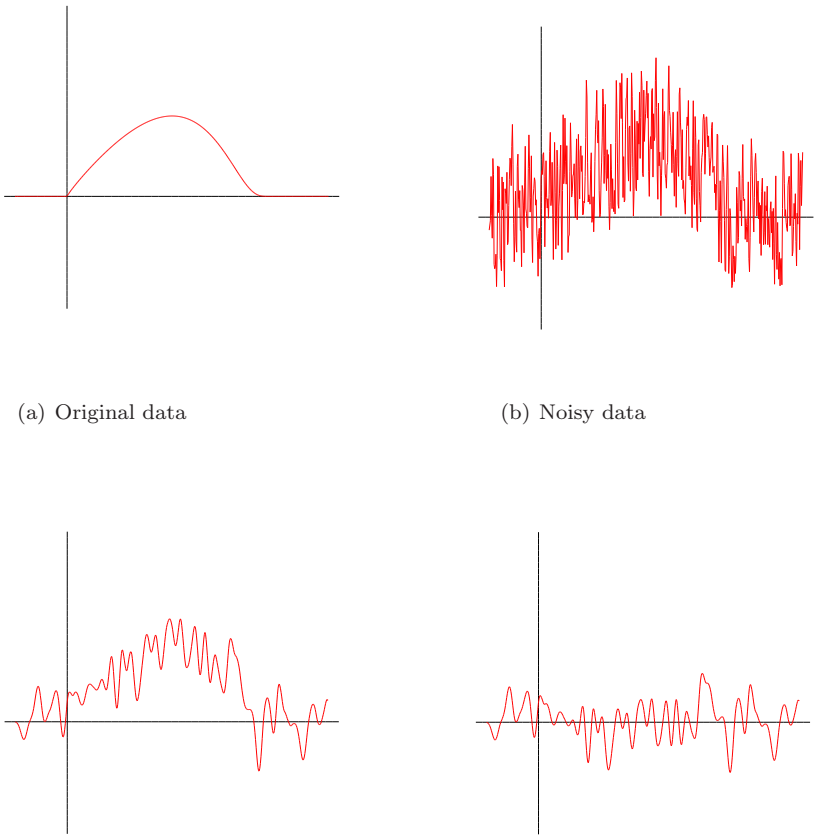
(a) Original data

(b) Noisy data

(c) Denoised data

(d) Error

Figure 9: Data size = 1024, s = 10, $j_0 = 3$, $j_1 = 10$



(a) Original data

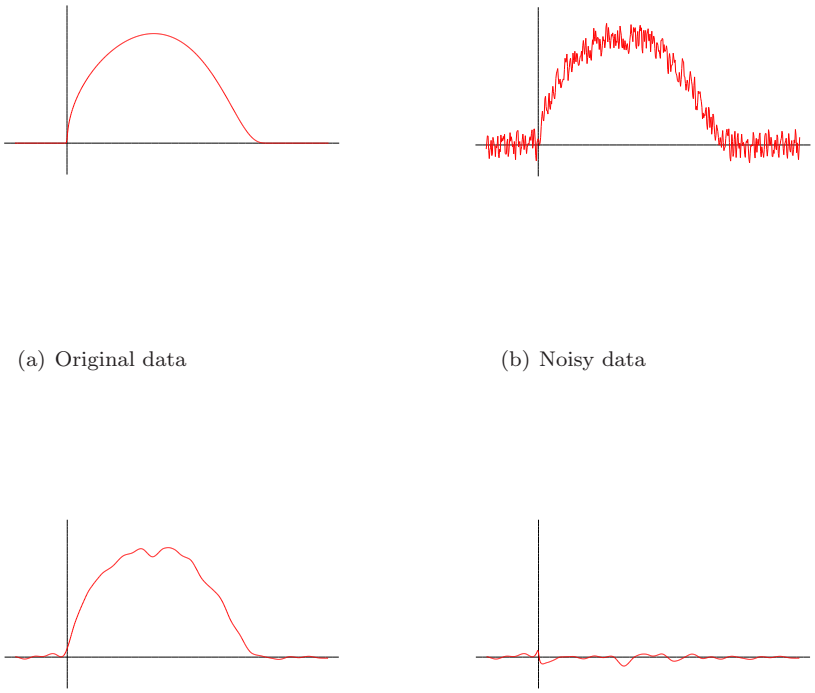
(b) Noisy data

(c) Denoised data

(d) Error

Figure 10: Data size = 512, s = 8, $j_0 = 2$, $j_1 = 9$

4.1.3 $\lambda = 0.5$, $\delta^2 = \frac{1}{300}$ (corresponding to white noise in $[-0.1, 0.1]$)



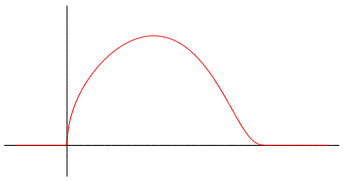
(a) Original data

(b) Noisy data

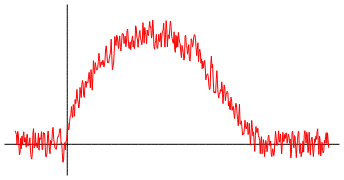
(c) Denoised data

(d) Error

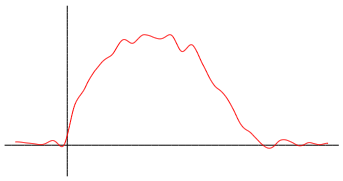
Figure 11: Data size = 512, s = 15, $j_0 = 2$, $j_1 = 9$



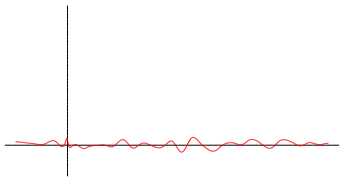
(a) Original data



(b) Noisy data

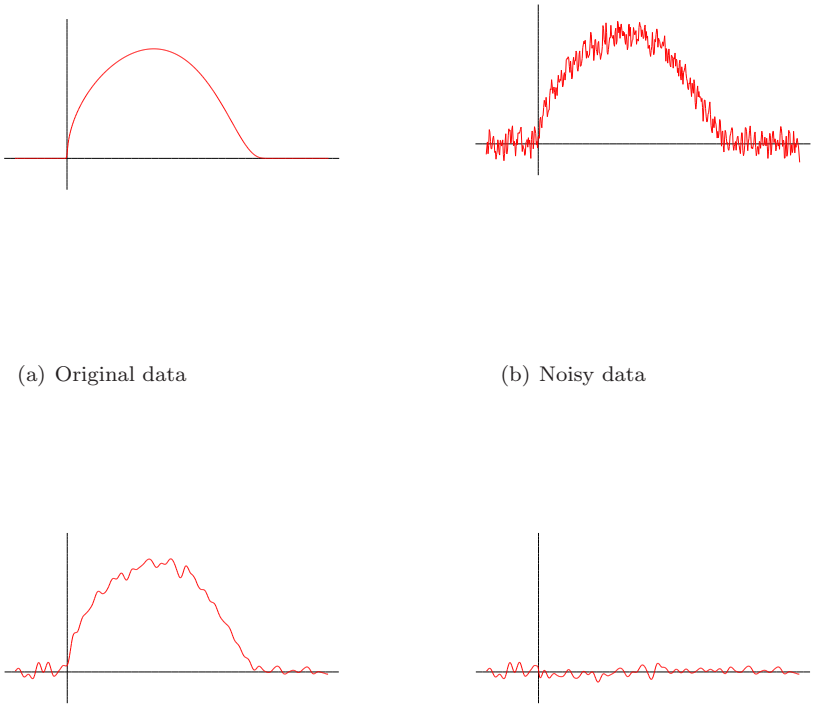


(c) Denoised data



(d) Error

Figure 12: Data size = 512, s = 10, $j_0 = 2$, $j_1 = 9$



(a) Original data

(b) Noisy data

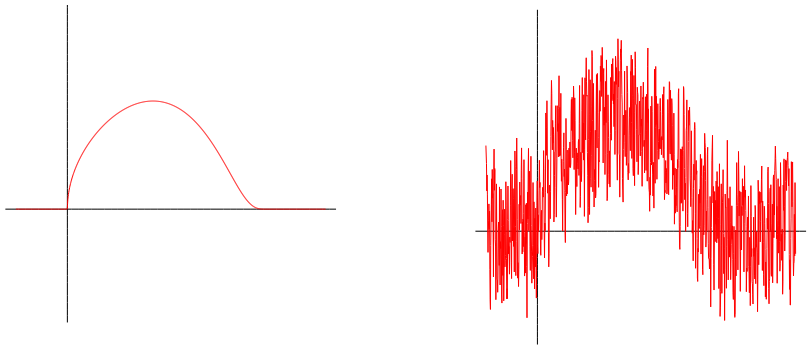
(c) Denoised data

(d) Error

Figure 13: Data size = 512, s = 8, $j_0 = 2$, $j_1 = 9$

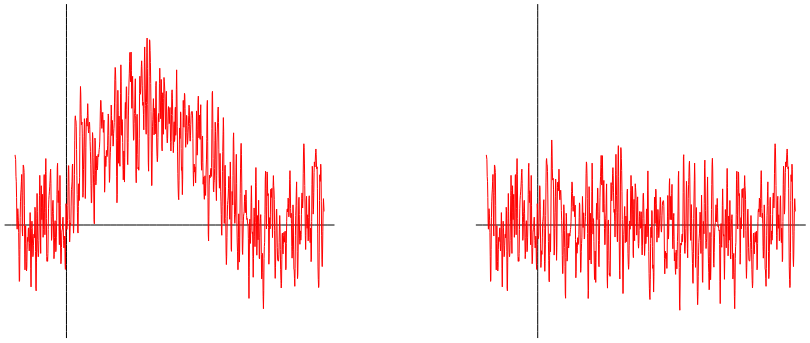
4.2 Minimizing the risk at each level

4.2.1 $\lambda = 0.5$, $\delta^2 = \frac{1}{12}$ (corresponding to white noise in $[-0.5, 0.5]$)



(a) Original data

(b) Noisy data

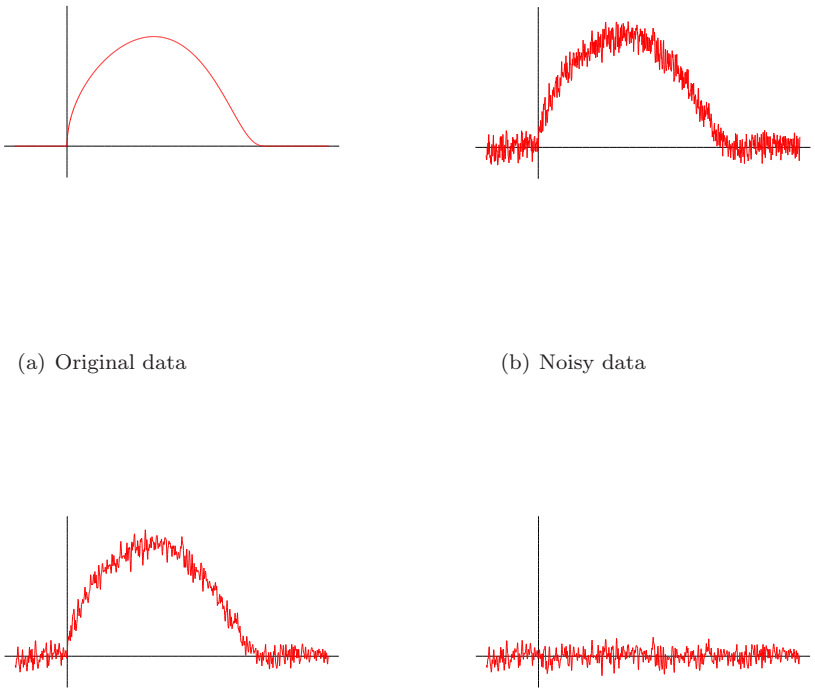


(c) Denoised data

(d) Error

Figure 14: Data size = 1024, $j_0 = 3$, $j_1 = 10$

4.2.2 $\lambda = 0.5$, $\delta^2 = \frac{1}{300}$ (corresponding to white noise in $[-0.1, 0.1]$)



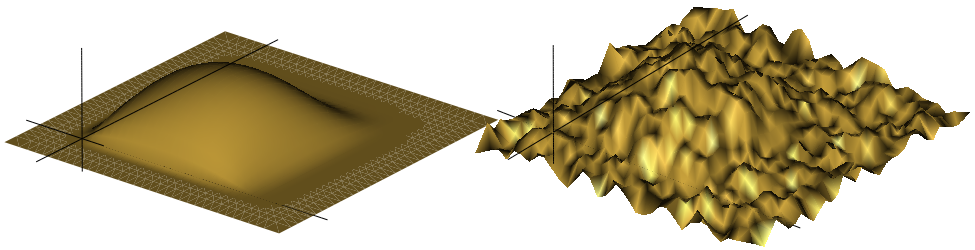
(a) Original data

(b) Noisy data

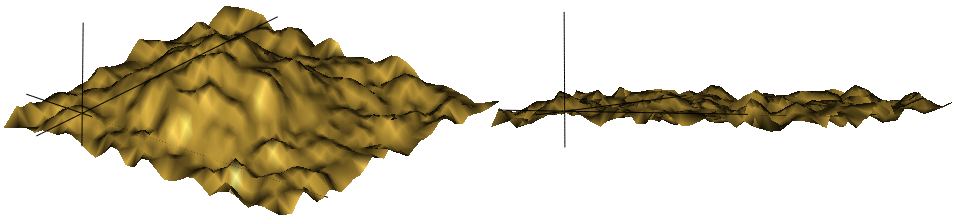
(c) Denoised data

(d) Error

Figure 15: Data size = 1024, $j_0 = 3$, $j_1 = 10$



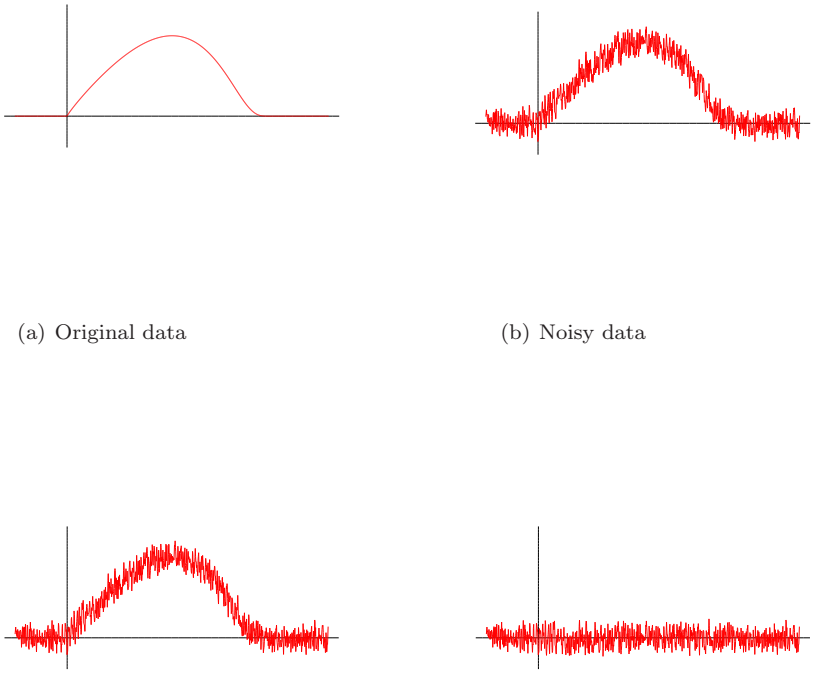
(a) Original data (b) Noisy data



(c) Denoised data (d) Error

Figure 16: Data size = 32 x 32, $j_0 = 1$, $j_1 = 5$

4.2.3 $\lambda = 0.9$, $\delta^2 = \frac{1}{300}$ (corresponding to white noise in $[-0.1, 0.1]$)



(a) Original data

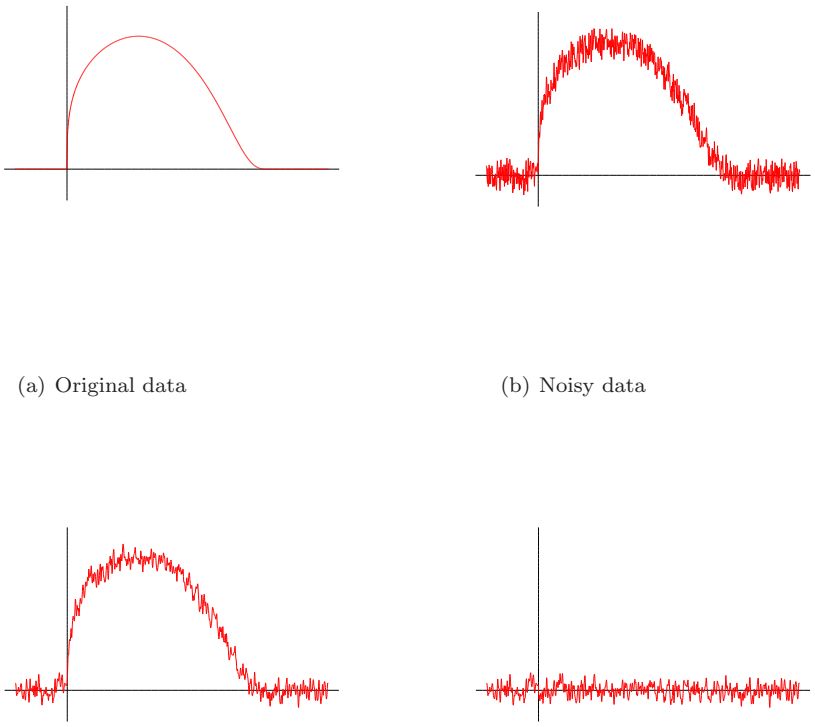
(b) Noisy data

(c) Denoised data

(d) Error

Figure 17: Data size = 1024, $j_0 = 3$, $j_1 = 10$

4.2.4 $\lambda = 0.3$, $\delta^2 = \frac{1}{300}$ (corresponding to white noise in $[-0.1, 0.1]$)



(a) Original data

(b) Noisy data

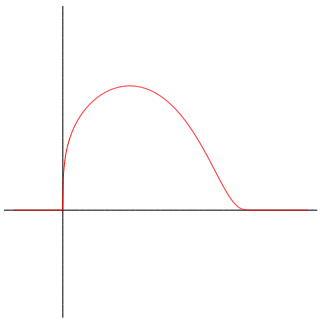
(c) Denoised data

(d) Error

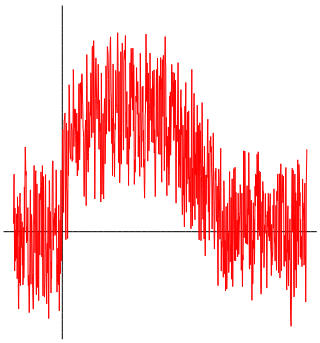
Figure 18: Data size = 1024, $j_0 = 3$, $j_1 = 10$

4.3 Equalizing $||\tilde{f}||_{B^s_{pq}}$ to $||f||_{B^s_{pq}}$ at each wavelet level

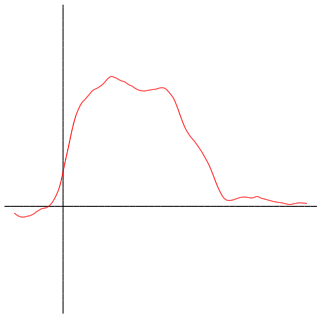
4.3.1 $\delta^2 = \frac{1}{12}$(corresponding to white noise in $[-0.5, 0.5]$)



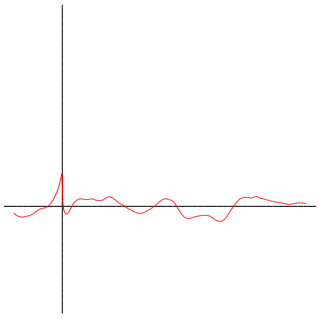
(a) Original data



(b) Noisy data

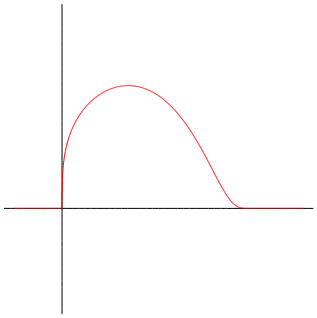


(c) Denoised data

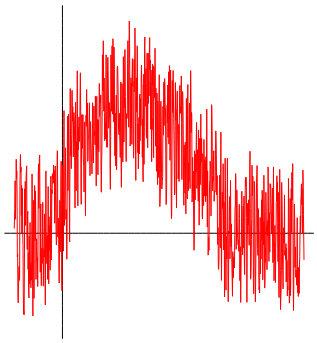


(d) Error

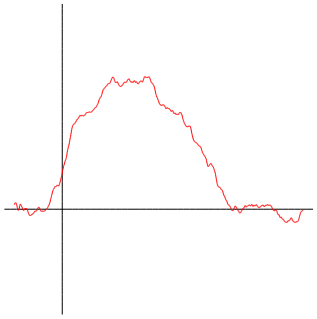
Figure 19: Data size = 1024, $||\tilde{f}||_{B^s_{0.50.5}} = ||f||_{B^s_{0.50.5}}$, $j_0 = 3$, $j_1 = 10$



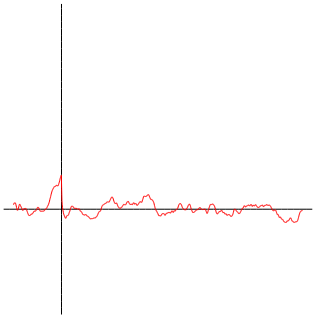
(a) Original data



(b) Noisy data

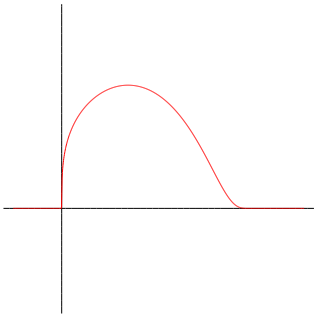


(c) Denoised data

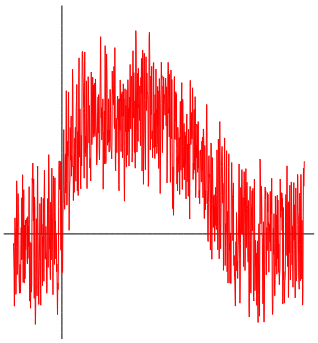


(d) Error

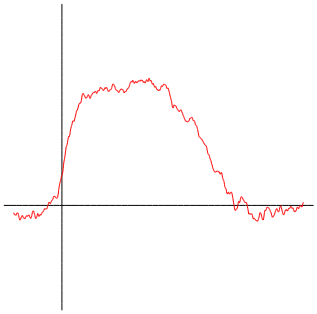
Figure 20: Data size = 1024, $||\tilde{f}||_{B^s_{1.51.5}} = ||f||_{B^s_{1.51.5}}$, $j_0 = 3$, $j_1 = 10$



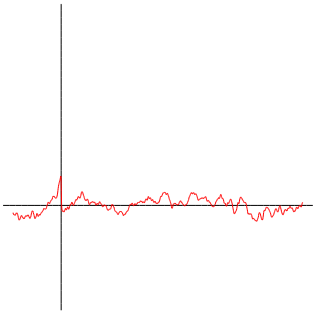
(a) Original data



(b) Noisy data

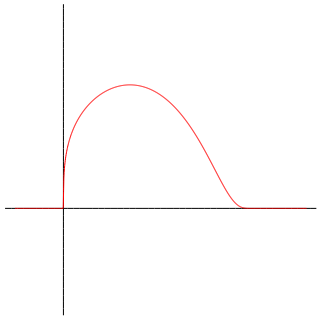


(c) Denoised data

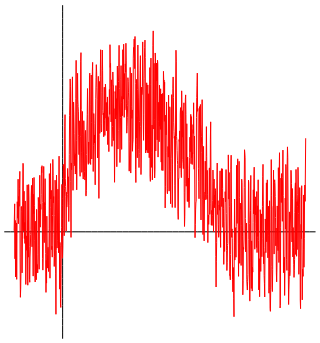


(d) Error

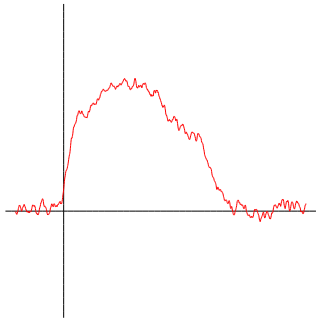
Figure 21: Data size = 1024, $||\tilde{f}||_{B^s_{22}} = ||f||_{B^s_{22}}$, $j_0 = 3$, $j_1 = 10$



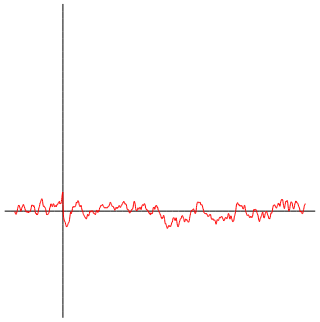
(a) Original data



(b) Noisy data

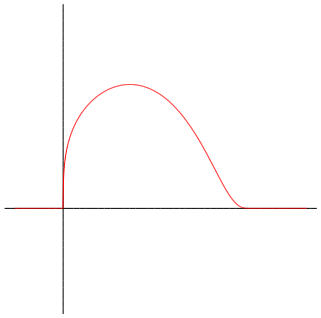


(c) Denoised data

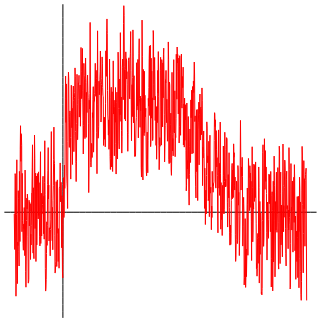


(d) Error

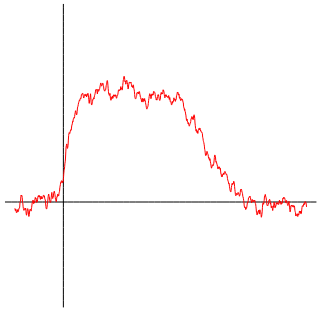
Figure 22: Data size = 1024, $||\tilde{f}||_{B^s_{2.52.5}} = ||f||_{B^s_{0.50.5}}$, $j_0 = 3$, $j_1 = 10$



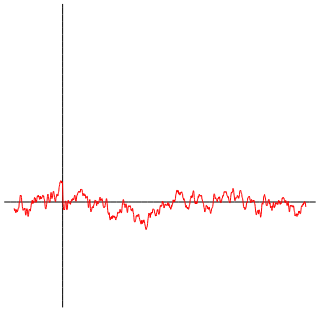
(a) Original data



(b) Noisy data



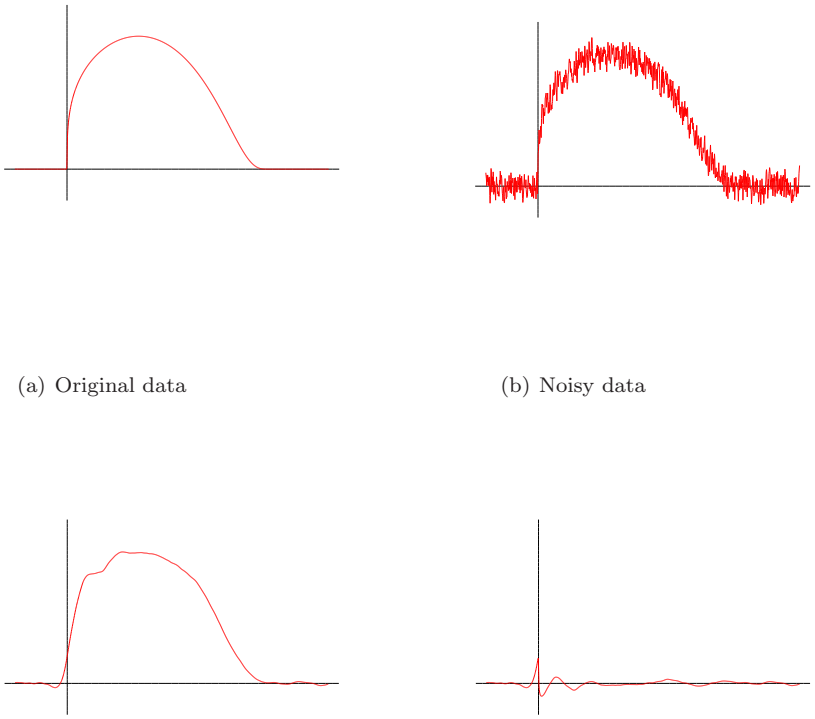
(c) Denoised data



(d) Error

Figure 23: Data size = 1024, $||\tilde{f}||_{B^s_{55}} = ||f||_{B^s_{55}}$, $j_0 = 3$, $j_1 = 10$

4.3.2 $\delta^2 = \frac{1}{300}$(corresponding to white noise in $[-0.1, 0.1]$)



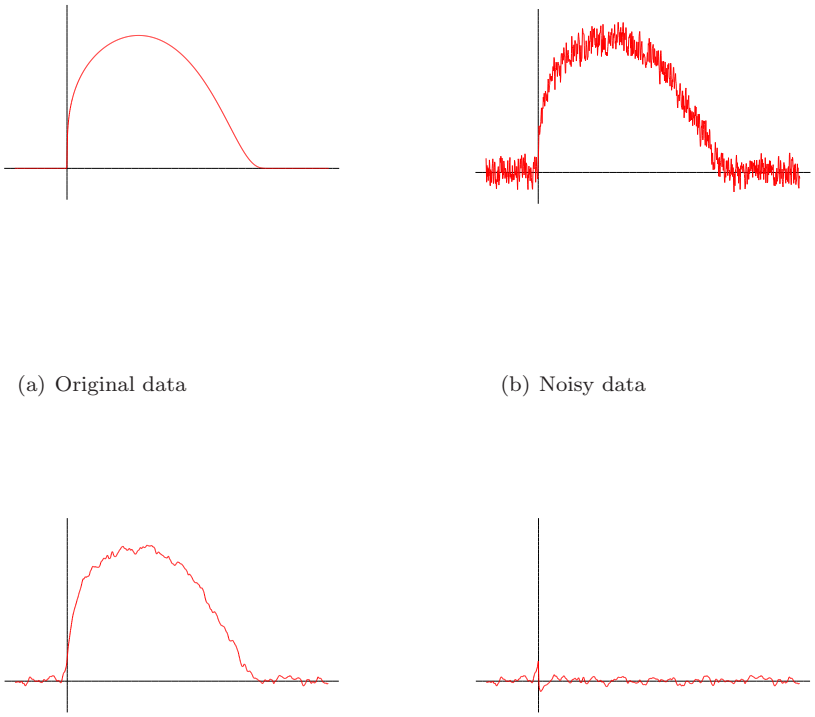
(a) Original data

(b) Noisy data

(c) Denoised data

(d) Error

Figure 24: Data size = 1024, $||\tilde{f}||_{B^s_{0.50.5}} = ||f||_{B^s_{0.50.5}}$, $j_0 = 3$, $j_1 = 10$



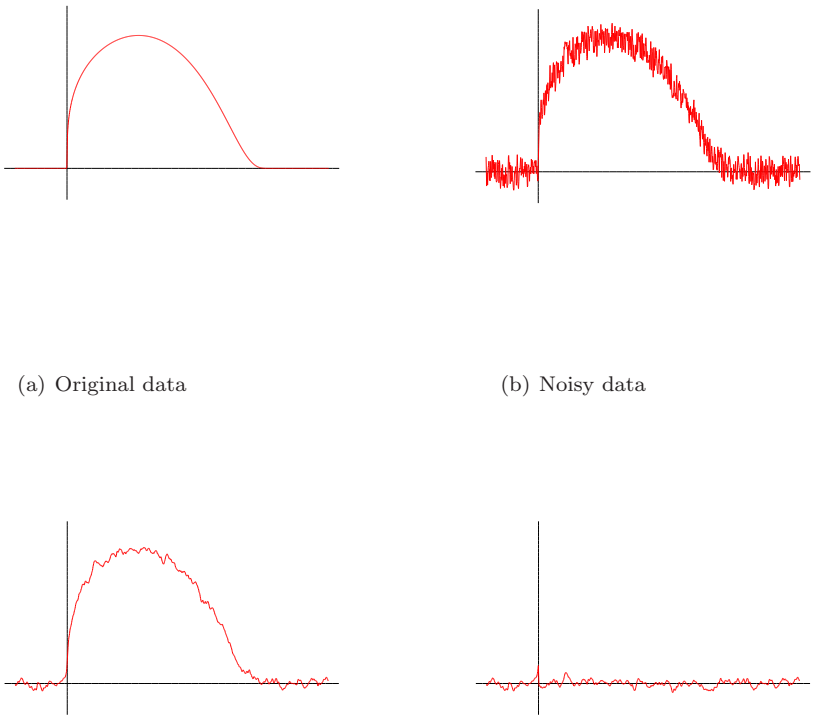
(a) Original data

(b) Noisy data

(c) Denoised data

(d) Error

Figure 25: Data size = 1024, $||\tilde{f}||_{B^s_{1.51.5}} = ||f||_{B^s_{1.51.5}}$, $j_0 = 3$, $j_1 = 10$



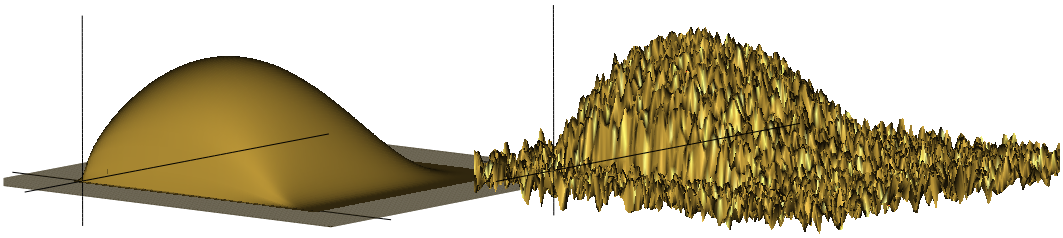
(a) Original data

(b) Noisy data

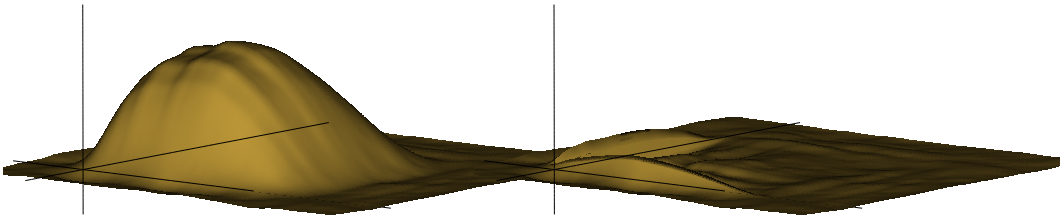
(c) Denoised data

(d) Error

Figure 26: Data size = 1024, $||\tilde{f}||_{B^s_{22}} = ||f||_{B^s_{22}}$, $j_0 = 3$, $j_1 = 10$

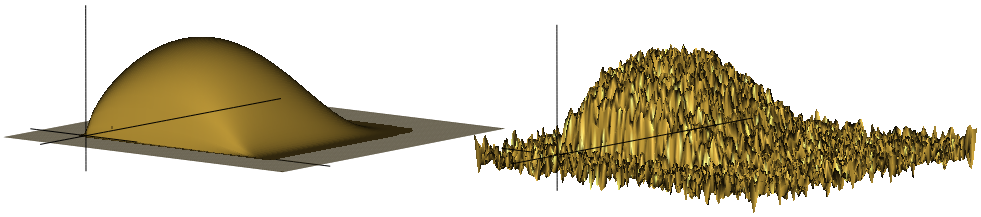


(a) Original data (b) Noisy data

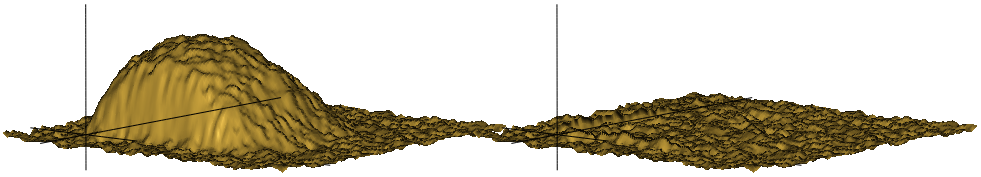


(c) Denoised data (d) Error

Figure 27: Data size = 128 x 128, $||\tilde{f}||_{B^s_{0.50.5}} = ||f||_{B^s_{0.50.5}}$, $j_0 = 3$, $j_1 = 10$

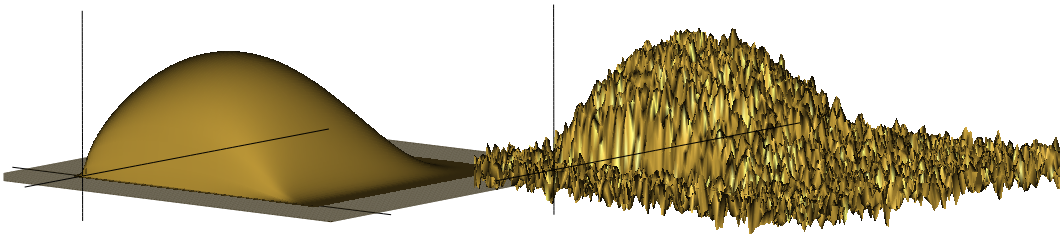


(a) Original data (b) Noisy data

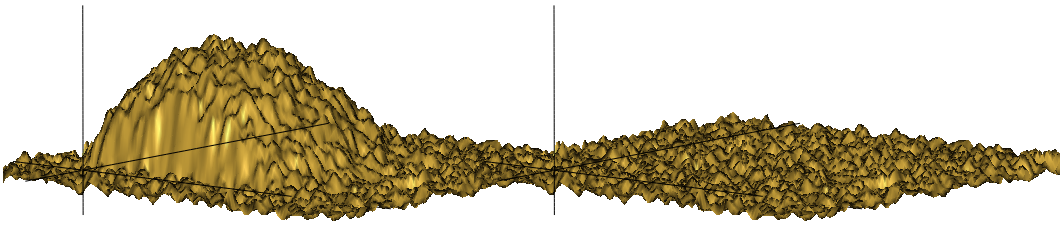


(c) Denoised data (d) Error

Figure 28: Data size = 128 x 128, $||\tilde{f}||_{B^s_{22}} = ||f||_{B^s_{22}}$, $j_0 = 3$, $j_1 = 10$



(a) Original data (b) Noisy data



(c) Denoised data (d) Error

Figure 29: Data size = 128 x 128, $||\tilde{f}||_{B^s_{55}} = ||f||_{B^s_{55}}$, $j_0 = 3$, $j_1 = 10$

References

[1] U. Amato and D.T. Vuza. Besov regularization, thresholding and wavelets for smoothing data. *Numer. Funct. Anal and Optimiz.*, 18(5&6) (1997), 461-493.

[2] U. Amato and D.T. Vuza. *Wavelet regularization for smoothing data*. Techn.Report CNR 108/1994, Instituto per Applicazioni della Matematica (1994).

[3] A. Antoniadis. Smoothing noisy data with coiflets. *Statistica Sinica*, 4 (1994), 651-678.

[4] A. Antoniadis. Smoothing noisy data with tapered coiflet series. *Scand. J. Statist.*, 23 (1996), 313-330.

[5] P.M. Anselone and P.J. Laurent. A general method for the construction of interpolating or smoothing spline-function. *Numerische Math.*, 12, (1968), 66-82.

[6] A. Barron, L. Birgé and P. Massart. Risk bounds for model selection via penalization. Preprint. (1995).

[7] L. Birgé and P. Massart. From model selection to adaptive estimation. *Festschr. for Lucien Le Cam*. Springer: New York (1997), 55-87.

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

[9] Breiman, L., Better subset regression using the nonnegative garrote, *Technometrics* **37(4)** (1995), 373–384.

[10] A. Cohen, W. Dahmen, R.A. Devore. Multiscale decompositions on bounded domains. *Trans. Amer. Math. Soc.*, (To appear.)

[11] A. Cohen, I. Daubechies, P. Vial. Wavelets and fast wavelet transforms on the interval. *Appl. Comput. Harmon. Anal.*, 1, (1994), 54-81.

[12] Cohen, A., Wavelet methods in numerical analysis, in *Handbook of Numerical Analysis*, vol. VII, P. G. Ciarlet and J. L. Lions (eds.), Elsevier, Amsterdam, 2000.

[13] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numer. Math.*, 31 (1979), 377-403.

[14] D.D. Cox. Approximation of method of regularization estimators. *Annals of Statistics*, 16, (1988), 694-713.

[15] W. Dahmen. Wavelet and multiscale methods for operator equations. *Acta Numerica*, (1997), 55-228.

[16] I. Daubechies. *Ten Lectures on Wavelets*. SIAM : Philadelphia, 1992.

[17] Dechevsky L. T., Atomic decomposition of function spaces and fractional integral and differential operators, *Fractional Calculus & Applied Analysis*, **2(4)** (1999), 367–381.

[18] L. T. Dechevsky, S. I. Penev. On shape-preserving probabilistic wavelet approximators. Stochast. Anal. and Appl., 15 (2), (1997), 187-215.

[19] L. T. Dechevsky, S. I. Penev. On shape-preserving wavelet estimators of cumulative distribution functions and densities. Stochast. Anal. and Appl., 16 (3), (1998), 428-469.

[20] L.T. Dechevsky and S.I. Penev. Weak penalized wavelet estimation. Research Report S99-1, Department of Statistics, School of Mathematics, University of New South Wales, Sydney, 1999.

[21] Dechevsky, L. T., and J. Gundersen, Isometric conversion between dimension and resolution, submitted.

[22] Dechevsky L. T., J. O. Ramsay, and S. I. Penev, Penalized wavelet estimation with Besov regularity constriants, *Math. Balkanica (N. S.)*, **13(3-4)** (1999), 257–376.

[23] Dechevsky L. T., MacGibbon B., Dobrev K. *Cross validation in Besov spaces, for multivariate density estimation and nonparametric regression with random design*, in preparation

[24] Delyon B. and Juditsky A. On Minimax Wavelet Estimators *Applied and computational harmonic analysis 3*, 215-228 (1996).

[25] R.A. DeVore, G. Kyriazis, D. Leviatan and V.M. Tikhomirov. Compression and nonlinear n-widths. *J. Adv. Comp. Math.*, 1, (1993), 197-214.

[26] R.A. DeVore and B.J. Lucier. Fast wavelet techniques for near-optimal image processing. In *Proc. IEEE Military Communications Conf. IEEE Communications Society*: New York (1992).

[27] Donoho, D. L. and I. M. Johnstone, Ideal spatial adaptation via wavelet shrinkage, *Biometrika*, **81(3)** (1994), 425–455.

[28] Donoho, D. L. and I. M. Johnstone, Minimax estimation via wavelet shrinkage, *Ann. Statist.*, **26(3)** (1998), 879–921.

[29] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard, Wavelet shrinkage: asymptopia? (with discussion), J. Roy. Statist. Soc. Ser. B, **57(2)** (1995), 301–369.

[30] Gao, H.-Y., and A. G. Bruce, WaveShrink with firm shrinkage, *Statist. Sinica*, **7(4)** (1997), 855–874.

[31] T. Gasser and H.-J. Müller. Kernel estimation of regression functions. In Gasser, T. and Rosenblatt, M. (Eds.). Smoothing techniques for curve estimation. *Lecture Notes in Math.*, 757 Springer : Heisenberg, (1979), 23-68.

[32] James, W., and C. Stein, Estimation with quadratic loss, Math. Statist. Probab. **1**, (1961), 311–319.

[33] M. Jansen, M. Malfait and A. Bultheel. Generalized cross validation for wavelet thresholding. *Signal Processing*, 56, (1997), 33-44.

[34] Mallat, S. G., *A Wavelet Tour of Signal Processing*, 2nd ed., Academic Press, London, 2001.

[35] G. Nason. Wavelet shrinkage using cross validation. *J. Royal Statist. Soc., Ser B*, 58(2), (1996), 463-479.

[36] M. Rudemo, Empirical choice of histograms and kernel density estimators, *Scand. J. Statist.*, 9, (1982), 65-78.

[37] Stein, C., Estimation of the mean of multivariate normal distribution, Ann. Statist., **9**, (1981), 1135–1151.

[38] K. Tribouley. Practical estimation of multivariate densities using wavelet methods. *Statist. Neerlandica*, 49 (1995), 41-62.

[39] Vidakovic, B., *Statistical Modeling by Wavelets*, Wiley, New York, 1999.

[40] Wahba, G., *Spline Models for Observational Data*, SIAM, Philadelphia, 1990.

[41] F.D. Utreras. Cross-validation techniques for smoothing spline-functions in one or two dimensions. In Gasser, T. and Rosenblatt, M. (Eds.). *Smoothing Techniques for Curve Estimation*. Lecture Notes in Math., 757. Springer : heidelberg, (1979), 196-231.

List of Figures