Robuste nicht-parametrische Regression mit Wavelets

Diplomarbeit

Arne Kovac
Universität-Gesamthochschule Essen

Juni 1996

U. L.

Inhaltsverzeichnis

1 Wavelets

Wavelets haben in den letzten Jahren in vielen Wissenschaftszweigen Einzug gehalten. Das Spektrum reicht dabei von mathematischen Gebieten wie der Funktionalanalysis, den gewöhnlichen oder partiellen Differentialgleichungen, der Numerik und der Statistik über Naturwissenschaften wie der Physik und Chemie bis hin zu den Wirtschaftswissenschaften. Zu den Anwendungen gehören die Kompression von Bilddaten (wie sie das amerikanische FBI zur digitalen Speicherung von Fingerabdrücken verwendet), Bildverarbeitung (Erkennung von Ecken und Kanten in Bildern), Vorhersage von Erdbeben, Spracherkennung und die synthetische Generierung von Klängen.

In diesem Kapitel sollen zunächst einmal die theoretischen Grundlagen der Wavelet-Theorie geklärt werden. Es wird sich herausstellen, daß die Wavelets ψ_{jk} eine Orthonormalbasis des $L^2(\mathbb{R})$ bilden, so daß sich jede Funktion $f \in L^2(\mathbb{R})$ eindeutig als

$$f = \sum_{j,k} < f, \psi_{jk} > \psi_{jk}$$

darstellen läßt, wobei $< \cdot, \cdot >$ das übliche Skalarprodukt im $L^2(\mathbb{R})$ bezeichnet. Ein wesentlicher Unterschied zu der Fourier-Transformation liegt jedoch darin, daß nicht nur in der Frequenz, sondern auch in der Zeit lokalisiert wird.

Wir werfen im ersten Abschnitt zunächst einen kurzen Blick auf Fourier-Reihen, um dann im zweiten Abschnitt nach der Definition von Wavelets zu sehen, welche Gemeinsamkeiten, aber auch welche Unterschiede die Wavelet-Transformation mit sich bringt.

Im dritten Teil dieses Kapitels wird gezeigt, wie man eine Wavelet-Basis konstruieren kann. Dazu wird zunächst die Haarbasis untersucht, die in der Mathematik schon seit Beginn dieses Jahrhunderts bekannt ist, und es wird skizziert, wie man die dort gewonnenen Ergebnisse verallgemeinern kann. Zum Schluß wird eine Familie von Wavelet-Basen vorgestellt, die Ingrid Daubechies 1988 entdeckt hat und die für die Anwendungen wichtige Merkmale aufweist, die dazu geführt haben, daß die Daubechies-Wavelets zur Zeit die wohl am meisten genutzten Wavelets sind.

Wir schließen dieses Kapitel, indem wir in Abschnitt 1.4 die Diskrete Wavelet-Transformation (DWT) vorstellen, und schlagen damit die Brücke zwischen der Wavelet-Theorie und den Anwendungen. Bei der DWT werden nicht Funktionen, sondern Vektoren des $\mathbb{R}^n$ transformiert – diese Vektoren werden dann später verrauschte Daten sein.

1.1 Fourier-Reihen

Wir wollen uns zunächst an Fourier-Reihen erinnern. Bekanntlich läßt sich jede L^2-integrierbare Funktion f über $[0, 2\pi]$ in eine Fourier-Reihe entwickeln, das heißt, daß wir f wie folgt darstellen können:

$$f = \sum_{-\infty}^{\infty} c_n \exp(in\cdot),$$

wobei die Konvergenz bezüglich der L^2-Norm stattfindet. Die Fourier-Koeffizienten c_n berechnen sich dabei als

$$c_n = (2\pi)^{-1} \int_0^{2\pi} f(x) \exp(-imx) dx.$$

Wegen $\exp(inx) = \cos(nx) + i\sin(nx)$ kann man die Fourier-Reihe auch betrachten als eine Entwicklung von f in Abhängigkeit von den reellen Sinus- und Kosinus-Funktionen

$$1, \frac{1}{\sqrt{\pi}}\cos(\cdot), \frac{1}{\sqrt{\pi}}\sin(\cdot), \frac{1}{\sqrt{\pi}}\cos(2\cdot), \frac{1}{\sqrt{\pi}}\sin(2\cdot), \frac{1}{\sqrt{\pi}}\cos(3\cdot), \ldots,$$

die ein Orthonormalsystem bilden. Wir betrachten die Fourier-Transformation als eine Abbildung, die eine Funktion f in eine Menge von Koeffizienten c_n abbildet.

Angenommen, wir wollen eine gegebene Funktion f durch Terme einer Fourierreihe approximieren. Dann ist klar, daß wir daran interessiert sind, daß möglichst wenige Koeffizienten deutlich von 0 verschieden sind. Bestimmte glatte Funktionen haben tatsächlich so eine „ökonomische" Fourier-Entwicklung. Es gibt zum Beispiel einen Satz, der aussagt, daß die Fourier-Koeffizienten einer beliebig oft differenzierbaren Funktion mit $k \to \infty$ schneller abklingen als jede Potenz $1/k^r$.

Im allgemeinen darf man dies jedoch leider nicht erwarten. Dies liegt im wesentlichen daran, daß die Fourier-Transformation nicht in der Zeit lokalisiert – jeder Koeffizient hängt von den Funktionswerten der zu transformierenden Funktion auf ganz $[0, 2\pi]$ ab. Dies hat zur Folge, daß auch Unstetigkeiten in f alle Koeffizienten beeinflußen. Typischerweise führen solche Sprungstellen zu Fourier-Koeffizienten der Größenordnung $1/k$ für $k \to \infty$. Man benötigt somit eine große Zahl an Termen, um eine unstetige Funktion mit genügend großer Genauigkeit approximieren zu können.

Oft kennt man in der Praxis nicht die ganze Funktion f, sondern nur einige Stichproben. Für diesen Zweck gibt es einen Algorithmus (FFT, fast fourier transform), der, falls $f(t)$ auf einem Gitter mit $n = 2^m$ Punkten spezifiziert ist, die eng-verwandte diskrete Fourier-Transformation der gegebenen Werte berechnet. Dies geschieht mit einem Rechenaufwand von $O(n \log_2 n)$ Operationen.

1.2 Die Wavelet-Transformation

Die Wavelet-Transformation ähnelt in vielen Punkten der Fourier-Transformation. Auch hier wollen wir eine gegebene Funktion f durch Reihen über Funktionen aus einer Orthonormalbasis darstellen. Hierfür wählen wir nun aber nicht die trigonometrische Basis, sondern eine andere, sogenannte Waveletbasis.

Die Elemente hieraus nennen wir *Wavelets* und wollen wir im folgenden mit ψ_{jk} bezeichnen. Man beachte, daß wir hier zwei Indizes haben – dies liegt daran, daß jedes Basiselement durch Strecken und Verschieben einer einzigen Funktion ψ generiert wird, die wir *Mutterwavelet* nennen. Das Wavelet ψ_{jk} erhält man von dem Mutterwavelet ψ durch eine horizontale Stauchung um den Faktor 2^j und eine Verschiebung um $2^{-j}k$:

$$\psi_{jk}(t) = 2^{j/2}\psi(2^j t - k) \tag{1.1}$$

Der Faktor $2^{j/2}$ ist eine Normalisierungskonstante, die gewährleistet, daß $||\psi_{jk}||_2 = ||\psi||_2 = 1$.

Definition 1.1. *Eine Funktion $\psi : \mathbb{R} \to \mathbb{R}$ heißt Mutterwavelet der Ordnung m, falls die folgenden vier Eigenschaften erfüllt sind.* [1]

[1] Es gibt Anwendungen, für die nicht alle diese vier Eigenschaften von Wichtigkeit sind. Daher wird in manchen Publikationen der Begriff eines Mutterwavelets etwas weiter gefaßt. Dies begründet sich auch darin, daß in

($\psi 1$) *Es ist* $\psi \in L^\infty(\mathbb{R})$. *Falls* $m > 0$, *so gilt zusätzlich, daß* $\psi \in C^{m-1}(\mathbb{R})$ *und für alle* $j \in \{1, \ldots, m-1\}$ *ist* $\psi^{(j)} \in L^\infty(\mathbb{R})$.

($\psi 2$) ψ *und alle Ableitungen bis zur Ordnung* $m-1$ *sind im Unendlichen schnell fallend.*

($\psi 3$) *Für alle* $j \in \{0, \ldots m\}$ *ist* $\int x^j \psi(x) dx = 0$.

($\psi 4$) *Die Menge* $\{\psi_{jk}\}_{j,k\in\mathbb{Z}}$ *ist eine Orthonormalbasis des* $L^2(\mathbb{R})$.

Wir können jetzt jede quadratintegrierbare Funktion $f \in L^2(\mathbb{R})$ [2] bzgl. der L^2-Norm entwickeln als

$$f = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} f_{jk} \psi_{jk}. \tag{1.2}$$

Die *Wavelet-Koeffizienten* berechnen sich dabei wie üblich als

$$\begin{aligned} f_{jk} &= \int_{-\infty}^{\infty} f(x) \overline{\psi}_{jk}(x) dx \\ &= < f, \psi_{jk} >, \end{aligned} \tag{1.3}$$

wobei $< \cdot, \cdot >$ das innere Produkt bezeichnet.

Ein einfaches Beispiel für eine Waveletbasis ist die Haarbasis, die von dem Mutterwavelet der Ordnung 0 (siehe Abbildung 1)

$$\psi(x) = \begin{cases} 1 & 0 \le x < \frac{1}{2}, \\ -1 & \frac{1}{2} \le x < 1 \\ 0 & \text{sonst} \end{cases} \tag{1.4}$$

generiert wird. ψ erfüllt offensichtlich die Eigenschaften $(\psi 1)$, $(\psi 2)$ und $(\psi 3)$. Außerdem kann man sich durch die folgenden Beobachtungen davon überzeugen, daß die ψ_{jk} ein Orthonormalsystem bilden.

1. Die Träger von zwei verschiedenen Wavelets auf der gleichen Skala schneiden sich nicht, also gilt
$$< \psi_{jk_1}, \psi_{jk_2} >= 0$$
für alle $j \in \mathbb{Z}$ sowie $k_1, k_2 \in \mathbb{Z}$ mit $k_1 \neq k_2$.

2. Wavelets auf zwei verschiedenen Skalen haben entweder ebenfalls sich nicht-schneidende Träger, oder wenn sie es doch tun, dann liegt der Träger eines Wavelets komplett in einer Menge, auf der das andere konstant ist. In jedem Fall gilt
$$< \psi_{j_1 k_1}, \psi_{j_2 k_2} >= 0$$
für alle $j_1, j_2 \in \mathbb{Z}$ sowie $k_1, k_2 \in \mathbb{Z}$ mit $j_1 \neq j_2$.

---

der Mathematik schon lange vor der Entdeckung von Waveletbasen Funktionenfamilien eine Rolle spielten, deren Elemente über eine zu (1.1) analoge Beziehung gewonnen wurden.

Für unsere Zwecke genügen aber die (in der Praxis mit Abstand meist genutzten) Daubechies-Wavelets, auf die wir später noch zu sprechen kommen. Diese erfüllen insbesondere alle Eigenschaften $(\psi 1)$ bis $(\psi 4)$.

[2] Ohne großen Aufwand hätten wir auch Wavelets im $\mathbb{R}^n$ betrachten können. Da wir uns aber bei der nichtparametrischen Regression später ohnehin nur auf den Fall $n = 1$ konzentrieren, verzichten wir auf diese Verallgemeinerung.

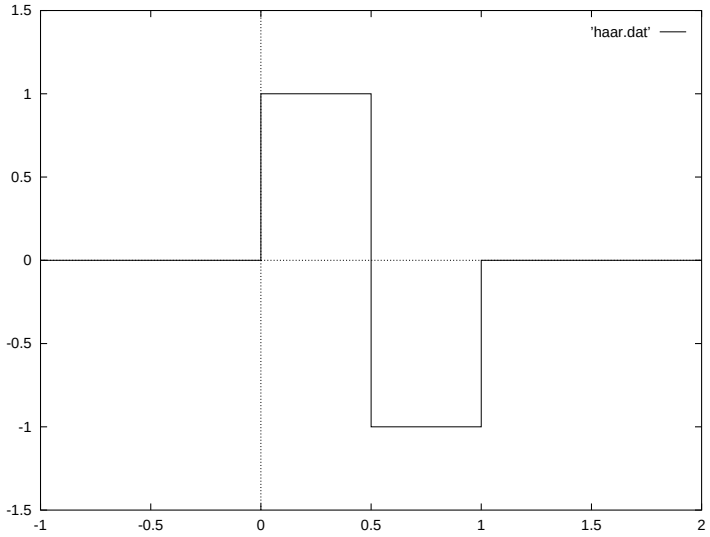

Abbildung 1: Das Mutterwavelet der Haarbasis

3. Schließlich gilt noch für alle $j, k \in \mathbb{Z}$, daß

$$< \psi_{jk}, \psi_{jk} >= 2^j \cdot \int \psi^2(2^j x - k)dx = 2^j \cdot \int_{2^{-j}k}^{2^{-j}(k+1)} 1dx = 1$$

Es bleibt nun noch zu zeigen, daß $\overline{\text{span}\{\psi_{jk}\}} = L^2(\mathbb{R})$. Dies werden wir aber erst im nächsten Abschnitt beweisen.

Die Haarbasis ist schon lange bekannt. A. Haar erwähnte sie erstmals 1910 in einem Anhang zu einer Arbeit ([22]). Levy benutzte sie in den 30er Jahren, als er die Brownsche Bewegung untersuchte. Den wesentlichen Antrieb für die Wavelettheorie gab allerdings erst in den 80er Jahren die Beobachtung, daß es noch viele andere Familien von Wavelets gibt, die von glatteren Mutterwavelets stammen. J.O. Strömberg war 1985 der erste, der für beliebiges $m \in \mathbb{N}$ eine Orthonormalbasis für den $L^2(\mathbb{R})$ aus Wavelets konstruierte, die m-mal differenzierbar sind. Yves Meyer fand eine Basis aus unendlich oft differenzierbaren Wavelets und Ingrid Daubechies konstruierte 1988 ihre (wohl mit Abstand am meisten verwendeten) *Daubechies-Wavelets,* die sich dadurch auszeichnen, daß sie neben m-maliger Differenzierbarkeit einen kompakten Träger besitzen. Auf diese Wavelets wird im nächsten Abschnitt eingegangen.

Wir kommen nun wieder zurück auf die Waveletentwicklung von Funktionen. Wir haben in $(\psi 4)$ bislang vorausgesetzt, daß die ψ_{jk} eine Orthonormalbasis des $L^2(\mathbb{R})$ bilden. Eine fundamentale Entdeckung über Wavelets ist aber, daß die Waveletbasen eine „universelle" Basis darstellen, nämlich eine unbedingte Basis [3] für die L^p-Räume ($1 < p < \infty$), Besov-Räume,

[3] Eine abzählbare Famile $\{e_i\}_{i\in\mathbb{N}}$ eines Banach-Raumes B heißt *Schauder-Basis,* falls jeder Vektor $x \in B$ geschrieben werden kann als

$$x = \sum_{i\in\mathbb{N}} \alpha_i e_i,$$

wobei die α_i skalarwertig sind und die Konvergenz durch

$$\lim_{n\to\infty} ||x - \sum_{i=1}^{n} \alpha_i e_i|| = 0$$

Triebel-Räume sowie die als Spezialfälle darin enthaltenen Hölder- und Sobolev-Räume (siehe auch Kapitel 2.2). Eine detaillierte Darstellung dieses Themas kann in [23] gefunden werden.

Im $L^1(\mathbb{R})$ und im $L^\infty(\mathbb{R})$ gelten diese Entwicklungen im allgemeinen nicht: Wählt man zum Beispiel f konstant 1, so würden alle Koeffizienten verschwinden, und es bliebe über $1 = 0$. Falls f andererseits eine C^∞-Funktion mit kompaktem Träger und Integral 1 ist, kann die Reihe in (1.2) nicht in der der L^1-Norm konvergieren, denn angenommen doch, dann kann man auf beiden Seiten integrieren, und zwar auf der linken Seite wegen der L^1-Konvergenz gliedweise, so daß man 0 erhält. Rechts aber steht 1, so daß sich wiederum ein Widerspruch ergibt.

Wir definieren daher:

Definition 1.2. *Eine Funktion $\phi : \mathbb{R} \to \mathbb{R}$ heißt Vaterwavelet der Ordnung m, falls die folgenden vier Eigenschaften erfüllt sind.*

($\phi 1$) *Es ist $\phi \in L^\infty(\mathbb{R})$. Falls $m > 0$, so gilt zusätzlich, daß $\phi \in C^{m-1}(\mathbb{R})$ und für alle $j \in \{1, \ldots, m-1\}$ ist $\phi^{(j)} \in L^\infty(\mathbb{R})$.*

($\phi 2$) *ϕ und alle Ableitungen bis zur Ordnung $m-1$ sind im Unendlichen schnell fallend.*

($\phi 3$) *$\int \phi(x)dx = 1$*

($\phi 4$) *Die Menge $\{\psi_{jk}, \phi_k\}_{j\in\mathbb{N},k\in\mathbb{Z}}$ ist eine Orthonormalbasis des $L^2(\mathbb{R})$.*

Es gilt dann für $f \in L^2(\mathbb{R})$, daß

$$f = \sum_{k=-\infty}^{\infty} \beta_k \phi_k + \sum_{j=0}^{\infty} \sum_{k=-\infty}^{\infty} f_{jk} \psi_{jk}, \tag{1.5}$$

wobei die Konvergenz in $L^2(\mathbb{R})$ stattfindet und sich die f_{jk} wie oben in (1.3) berechnen und die β_k als $\beta_k =< f, \phi_k >$.

Meyer zeigt in [23], daß die oben angesprochenen Probleme verschwinden, wenn man diese Darstellung benutzt. Wählt man zum Beispiel wieder f konstant 1, so ist $\beta_k = 1$ für alle $k \in \mathbb{Z}$ und es ergibt sich die Identität $1 = \sum_k \phi_k(x)$, die überraschenderweise stets erfüllt ist. Es gilt sogar allgemeiner, daß für jedes Polynom P vom Grad kleiner oder gleich m die Reihe konvergiert. Meyer betrachtet dazu in [23] den Untervektorraum V_0 des $L^2(\mathbb{R})$ mit

$$V_0 := \overline{\operatorname{span}\{\phi_k\}}$$

sowie die orthogonale Projektion P_0 auf V_0 und zeigt, daß $P_0(P) = P$ gilt. Der Titel des entsprechenden Abschnitts lautet vollkommen richtig: „A remarkable Identity"

Als nächstes betrachten wir die Waveletkoeffizienten noch einmal genauer: Jeder Koeffizient f_{jk} hängt nur *lokal* von f ab, denn

$$f_{jk} = \int_{-\infty}^{\infty} f(x)\overline{\psi_{jk}(x)}dx,$$

und das Wavelet ψ_{jk} hat endlichen (oder zumindestens so gut wie endlichen) Träger. f_{jk} enthält also Information über die Skala 2^{-j} nahe der Position $2^{-j}k$. Dieses Verhalten kann man sich

definiert ist. Eine Schauder-Basis heißt *unbedingte Basis*, falls für alle $x \in B$ die obige Reihe *unbedingt* zu x konvergiert, d.h. daß für jede Permutation $\sigma : \mathbb{N} \to \mathbb{N}$ die Reihe $\sum_0^\infty x_{\sigma(i)}$ in der Norm gegen x konvergiert.

gut wie ein Mikroskop vorstellen, bei der das Sichtfenster immer die gleiche Größe hat, man allerdings den Grad der Vergrößerung ebenso wie den zu betrachtenden Ausschnitt einstellen kann.

Diese Eigenschaft ist der Schlüssel zu der Wavelettheorie und hat nachhaltige Konsequenzen. Zum Beispiel beeinflussen Unstetigkeiten in f nur eine verhältnismäßig kleine Zahl an Koeffizienten. Da sich aber andererseits Funktionen, die zwischen zwei Unstetigkeitsstellen glatt sind, dort lokal gut durch Polynome approximieren lassen und diese durch Bedingung $(\psi 3)$ bei Verwendung eines hinreichend glatten Wavelets verschwinden, folgt, daß derartige Funktionen ökonomische Wavelet-Darstellungen besitzen – fast alle Koeffizienten können auf Null gesetzt werden, ohne daß man einen großen Fehler begeht. [4] Dadurch empfiehlt sich der Einsatz von Wavelets für Kompressionstechniken.

Das amerikanische FBI verwendet zum Beispiel Wavelets dazu, Fingerabdrücke elektronisch zu archivieren. Das Scannen einer einzelnen Karteikarte führt zu ungefähr 10 MByte Daten, und das FBI besitzt mehr als 200 Millionen solcher Karten . . .

1.3 Multiresolution Analysis und die Daubechies-Wavelets

In diesem Abschnitt soll gezeigt werden, wie man zu anderen Wavelet-Basen als der im vorangegangenen Abschnitt vorgestellten Haarbasis (insbesondere zu den Daubechies-Basen) gelangt. Die hier zusammengetragenen Ergebnisse sind aber zugleich grundlegend für das Verstehen der diskreten Wavelet-Transformation, die im nächsten Kapitel vorgestellt wird.

Definition 1.3. *Eine Familie $(V_j)_{j\in\mathbb{Z}}$ von abgeschlossenen Teilräumen des $L^2(\mathbb{R})$ heißt Multiresolution Analysis, falls die folgenden fünf Eigenschaften erfüllt sind:*

[M1] $V_j \subset V_{j+1}$

[M2] $v(x) \in V_j \Leftrightarrow v(2x) \in V_{j+1}$

[M3] $v(x) \in V_0 \Leftrightarrow v(x+1) \in V_0$

[M4] $\bigcup_{j\in\mathbb{Z}} V_j$ *liegt dicht in* $L^2(\mathbb{R})$ *und* $\bigcap_{j\in\mathbb{Z}} V_j = \{0\}$

[M5] Es existiert eine Funktion $\phi \in V_0$, so daß die Familie $\{\phi(x-k)|k \in \mathbb{Z}\}$ eine Orthonormalbasis des V_0 ist.

Wir benutzen die folgenden Sprechweisen: Die Teilräume wollen wir *Level* nennen, und ein Level ist *gröber* (bzw. *feiner*) als ein anderer, falls der Index des dazugehörigen Teilraums kleiner ist (bzw. größer). Die Funktion ϕ aus [M5] heißt *Skalierungsfunktion*. [5]

Beispiel 1.1. Betrachte

$$V_j := \{f \in L^2(\mathbb{R}) : f \text{ konstant auf } [2^{-j}k, 2^{-j}(k+1)) \text{ für alle } k \in \mathbb{Z}\}. \tag{1.6}$$

Die Eigenschaften [M1] bis [M4] sind offenbar alle erfüllt. Die orthogonale Projektion nach V_j ist gegeben durch

$$P_j(f)|_{[2^{-j}k,2^j(k+1))} = 2^j \int_{2^{-j}k}^{2^{-j}(k+1)} f(x)dx.$$

[4] Eine detaillierte Analyse dieser Eigenschaft findet man in [8] und [23].

[5] ϕ übernimmt bei der Generierung von Waveletbasen den Part des Vaterwavelets.

Für festes $f \in L^2(\mathbb{R})$ sind die $P_j f$ mit wachsendem j eine Näherung an f auf immer feinerer Skala. Schließlich wählen wir noch ϕ als die charakteristische Funktion auf $[0,1]$.

$$\phi(x) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{sonst} \end{cases}$$

Damit gilt aber offensichlich auch [M5]. Die durch (1.6) definierten Räume V_j bilden also tatsächlich eine Multiresolution Analysis.

Wir definieren nun (wie im vorangegangenen Kapitel bei der Einführung von Mutterwavelets)

$$\phi_{jk}(t) = 2^{j/2}\phi(2^j t - k). \tag{1.7}$$

Dann gilt

$$< f, \phi_{jk} >= 2^{j/2} \int_{2^{-j}k}^{2^{-j}(k+1)} f(x)dx$$

und somit

$$P_j f = \sum_{k\in\mathbb{Z}} < f, \phi_{jk} > \phi_{jk}.$$

Als nächstes wollen wir die Differenz zwischen $P_j f$ und der nächst feineren Näherung $P_{j+1} f$ betrachten. Man überprüft zunächst leicht, daß

$$\phi_{jk} = \frac{1}{\sqrt{2}}(\phi_{j+1,2k} + \phi_{j+1,2k+1})$$

und

$$< f, \phi_{jk}, f >= \frac{1}{\sqrt{2}} \left(< f, \phi_{j+1,2k} > + < f, \phi_{j+1,2k+1} >\right).$$

Es ergibt sich also

$$\begin{aligned} P_{j+1}f - P_j f &= \sum_{k\in\mathbb{Z}} < f, \phi_{j+1,k} > \phi_{j+1,k} - \sum_{k\in\mathbb{Z}} < f, \phi_{jk} > \phi_{jk} \\ &= \sum_{k\in\mathbb{Z}} < f, \phi_{j+1,k} > \phi_{j+1,k} \\ &\quad - \sum_{k\in\mathbb{Z}} \frac{1}{2} \left(< f, \phi_{j+1,2k} > + < f, \phi_{j+1,2k+1} >\right) \cdot (\phi_{j+1,2k} + \phi_{j+1,2k+1}) \\ &= \frac{1}{2} \sum_{k\in\mathbb{Z}} \left(< f, \phi_{j+1,2k} > - < f, \phi_{j+1,2k+1} >\right) \cdot (\phi_{j+1,2k} - \phi_{j+1,2k+1}). \end{aligned}$$

Es sei nun ψ das in (1.4) definierte Mutterwavelet der Haarbasis. Dann gilt offensichtlich

$$\psi(x) = \phi(2x) - \phi(2x-1)$$

und damit

$$\begin{aligned} \psi_{jk}(x) &= 2^{j/2}\psi(2^j x - k) \\ &= 2^{j/2}\left(\phi(2^{j+1}x - 2k) - \phi(2^{j+1}x - 2k - 1)\right) \\ &= \frac{1}{\sqrt{2}}(\phi_{j+1,2k} - \phi_{j+1,2k+1}), \end{aligned}$$

also

$$
\begin{aligned}
Q_j f &:= P_{j+1} f - P_j f \\
&= \sum_{k \in \mathbb{Z}} < f, \psi_{jk} > \psi_{jk}.
\end{aligned} \tag{1.8}
$$

Wir wollen nun aus den vorangegangenen Umformungen einige Schlußfolgerungen ziehen, die uns insbesondere zeigen, daß ψ tatsächlich ein Mutterwavelet der Ordnung 0 ist. Wir hatten im letzten Abschnitt bereits gesehen, daß die ψ_{jk} orthonormal sind. In (1.8) wird also $Q_j f$ bezüglich eines Orthonormalsystems entwickelt. Dabei ist $Q_j f$ die orthogonale Projektion von f auf $W_j := P_{j+1} L^2 - P_j L^2$, insbesondere auf das orthogonale Komplement von V_j in V_{j+1}. Aus den ersten beiden Eigenschaften einer Multiresolution Analysis, [M1] und [M2], zusammen mit $W_j \perp V_j$ und $V_{j+1} = V_j \oplus W_j$ folgt, daß die W_j paarweise orthogonal sind und ihre direkte Summe der $L^2(\mathbb{R})$ ist. Weil für jedes j die Familie $\{\psi_{jk}\}_{k \in \mathbb{Z}}$ eine Orthonormalbasis für W_j bildet, folgt, daß die gesamte Familie $\{\psi_{jk}\}_{j,k \in \mathbb{Z}}$ eine Orthonormalbasis des $L^2(\mathbb{R})$ ist.

In dem vorangegangenen Beispiel haben wir gesehen, wie eine Wavelet-Basis aus einer Multiresolution Analysis gewonnen wurde. Wir versuchen nun, dieses Beispiel zu verallgemeinern. Dazu sei $\{V_j\}_{j \in \mathbb{Z}}$ eine Multiresolution Analysis. Wie eben definieren wir W_j als das orthogonale Komplement von V_j in V_{j+1}:

$$V_{j+1} = V_j \oplus W_j \qquad W_j \perp V_j$$

Es folgt sofort, daß alle Räume W_j skalierte Versionen von W_0 sind,

$$f \in W_j \Leftrightarrow f(2^{-j} \cdot) \in W_0,$$

und daß die W_j orthogonale Räume sind, deren Summe der $L^2(\mathbb{R})$ ist,

$$L^2(\mathbb{R}) = \oplus_{j \in \mathbb{Z}} W_j$$

Um aus der Multiresolution Analysis eine Wavelet-Basis zu gewinnen, müssen wir somit eine Funktion $\psi \in W_0$ finden, deren ganzzahlige Verschiebungen eine Orthonormalbasis von W_0 darstellen. Weil $\phi \in V_0 \subset V_1 = \overline{\operatorname{span}\{\phi(2 \cdot - k)\}}$, gibt es $c_k \in l_2$ so, daß

$$\phi(x) = \sum_{k \in \mathbb{Z}} c_k \phi(2x - k). \tag{1.9}$$

Definiert man dann

$$\psi(x) = \sum_{k \in \mathbb{Z}} (-1)^k \overline{c}_{k+1} \phi(2x + k), \tag{1.10}$$

so stellt sich heraus, daß die dazugehörigen ψ_{0k} eine Orthonormalbasis von W_0 bilden (siehe zum Beispiel [23], [8], [32], [7] oder [5]). [6] Yves Meyer merkt in diesem Zusammenhang ganz richtig an, daß das Mutterwavelet somit eine Tochter des Vaterwavelets ist . . .

Insgesamt ist jetzt skizziert, wie man aus einer Multiresolution Analysis eine Wavelet-Basis gewinnt. Offen bleibt aber noch die Frage, welche Funktionen $\phi \in L^2(\mathbb{R})$ überhaupt eine Multiresolution Analysis erzeugen. Eine Wahl von ϕ haben wir bereits im Beispiel kennengelernt,

[6] Diese wichtige Beziehung zwischen ϕ und ψ wird grundlegend sein, wenn im nächsten Kapitel die Diskrete Wavelet-Transformation definiert wird.

nämlich die charakteristische Funktion über dem Intervall $[0,1)$, deren zugehörige Wavelet-Basis die Haarbasis ist. Wir haben jedoch bereits angedeutet, wofür wir Wavelets später benutzen möchten, nämlich um Funktionen durch endliche Summen von Wavelets zu approximieren. Daher sind wir daran interessiert, Wavelets von höherer Ordnung zu finden.

Eine bekannte Familie von Skalierungsfunktionen stellen die B-Splines dar. Das B-Spline der Ordnung 1 ist wieder die charakteristische Funktion über $[0,1)$, $N_1(x) = \chi_{[0,1)}(x)$. Für $m > 1$ definiert man rekursiv das B-Spline N_m über eine Faltung:

$$N_m = N_{m-1} * N_1.$$

Zum Beispiel ergibt sich für $m = 2$, daß

$$N_2(x) = \begin{cases} x, & 0 \leq x \leq 1 \\ 2 - x, & 1 \leq x \leq 2 \\ 0, & \text{sonst.} \end{cases}$$

Wählt man $\phi = N_m$ für ein $m > 1$, so sind zwar die Translationen ϕ_{0k} noch nicht orthogonal, doch in [23] wird gezeigt, daß man im Falle, daß die ϕ_{0k} eine Riesz-Basis von V_0 darstellen, durch

$$(\tilde{\phi})\hat{}\,(\xi) = C\hat{\phi}(\xi)\left(\sum_{k\in\mathbb{Z}} |\hat{\phi}(\xi + 2k\pi)|^2\right)^{-1/2}$$

eine Funktion $\tilde{\phi}$ definieren kann, so daß die $\tilde{\phi}_{0k}$ eine Orthonormalbasis von V_0 darstellen. [7] Auf diese Weise erhält man Wavelet-Basen beliebiger Ordnung – die sogenannten Battle-Lemarie-Basen. Wenngleich jedoch die Skalierungsfunktionen kompakten Träger haben, stellt sich heraus, daß die dazugehörigen Wavelets diese Eigenschaft nicht aufweisen. Wie wir aber noch sehen werden, ist ein kompakter Träger für unsere Zwecke wichtig.

Ingrid Daubechies zeigte 1988 in [7], wie man orthonormale Wavelets beliebiger Ordnung mit kompaktem Träger konstruieren kann. Startpunkt ihrer Überlegungen ist dabei nicht die Funktion ϕ selbst, sondern sind die Koeffizienten ϕ_k in (1.9). In Abbildung 2 sind die Koeffizienten für die Daubechies-Wavelets der Ordnungen 1 bis 8 aufgelistet. [8] Mit ihrer Kenntnis kann man die dazugehörige Skalierungsfunktion und das Wavelet berechnen. Der dazu erforderliche Algorithmus ist jedoch letztlich nichts anderes als die inverse Diskrete Wavelet-Transformation, die Gegenstand des nächsten Kapitels sein wird, so daß wir für den Moment nur in den Abbildungen 3 und 4 zeigen, wie die Daubechies-Wavelets für $m = 1$ und $m = 4$ aussehen, wobei m die Ordnung des Wavelets bezeichnen soll. [9]

[7] Bei einer Reihe von Anwendungen kann auf Orthonormalität verzichtet werden. Man verlangt dann von der Wavelet-Basis lediglich, eine Riesz-Basis des $L^2(\mathbb{R})$ zu bilden, d.h. jedes $f \in L^2(\mathbb{R})$ läßt sich eindeutig darstellen als $f = \sum_k c_k f_k$ und es existieren Konstanten $A, B > 0$, so daß

$$A||f||^2 \leq \sum_k |c_k|^2 \leq B||f||^2.$$

Solche Wavelets heißen *semi-orthogonal*. Wenn man nur an derartigen Wavelet-Basen interessiert ist, genügen die B-Splines, da sie semi-orthogonale Wavelets mit kompaktem Träger generieren.

[8] Wie die Koeffizienten berechnet werden, kann zum Beispiel in [7] oder [8] nachgelesen werden.

[9] I. Daubechies stellt noch eine zweite Familie von orthogonalen Wavelets mit kompaktem Träger vor, die zugunsten einer weniger asymmetrischen Form einen etwas breiteren Träger besitzt. Wenn wir in dieser Arbeit von „Daubechies-Wavelets" sprechen, meinen wir die Wavelet-Basis, deren Filter-Koeffizienten durch Abbildung 2 gegeben sind.

m=1	c_0	0.482962913145
	c_1	0.836516303738
	c_2	0.224143868042
	c_3	-0.129409522551
m=2	c_0	0.332670552950
	c_1	0.806891509311
	c_2	0.459877502118
	c_3	-0.135011020010
	c_4	-0.085441273882
	c_5	0.035226291882
m=3	c_0	0.230377813309
	c_1	0.714846570553
	c_2	0.630880767930
	c_3	-0.027983769417
	c_4	-0.187034811719
	c_5	0.030841381836
	c_6	0.032883011667
	c_7	-0.010597401785
m=4	c_0	0.160102397974
	c_1	0.603829269797
	c_2	0.724308528438
	c_3	0.138428145901
	c_4	-0.242294887066
	c_5	-0.032244869585
	c_6	0.077571493840
	c_7	-0.006241490213
	c_8	-0.012580751999
	c_9	0.003335725285
m=5	c_0	0.111540743350
	c_1	0.494623890398
	c_2	0.751133908021
	c_3	0.315250351709
	c_4	-0.226264693965
	c_5	-0.129766867567
	c_6	0.097501605587
	c_7	0.027522865530
	c_8	-0.031582039318
	c_9	0.000553842201
	c_{10}	0.004777257511
	c_{11}	-0.001077301085
m=6	c_0	0.077852054085
	c_1	0.396539319482
	c_2	0.729132090846
	c_3	0.469782287405
	c_4	-0.143906003929
	c_5	-0.224036184994
	c_6	0.071309219267
	c_7	0.080612609151
	c_8	-0.038029936935
	c_9	-0.016574541631
	c_{10}	0.012550998556
	c_{11}	0.000429577973
	c_{12}	-0.001801640704
	c_{13}	0.000353713800
m=7	c_0	0.054415842243
	c_1	0.312871590914
	c_2	0.675630736297
	c_3	0.585354683654
	c_4	-0.015829105256
	c_5	-0.284015542962
	c_6	0.000472484574
	c_7	0.128747426620
	c_8	-0.017369301002
	c_9	-0.044088253931
	c_{10}	0.013981027917
	c_{11}	0.008746094047
	c_{12}	-0.004870352993
	c_{13}	-0.000391740373
	c_{14}	0.000675449406
	c_{15}	-0.000117476784
m=8	c_0	0.038077947364
	c_1	0.243834674613
	c_2	0.604823123690
	c_3	0.657288078051
	c_4	0.133197385825
	c_5	-0.293273783279
	c_6	-0.096840783223
	c_7	0.148540749338
	c_8	0.030725681479
	c_9	-0.067632829061
	c_{10}	0.000250947115
	c_{11}	0.022361662124
	c_{12}	-0.004723204758
	c_{13}	-0.004281503682
	c_{14}	0.001847646883
	c_{15}	0.000230385764
	c_{16}	-0.000251963189
	c_{17}	0.000039347320

Abbildung 2: Die Koeffizienten der Daubechies-Wavelets

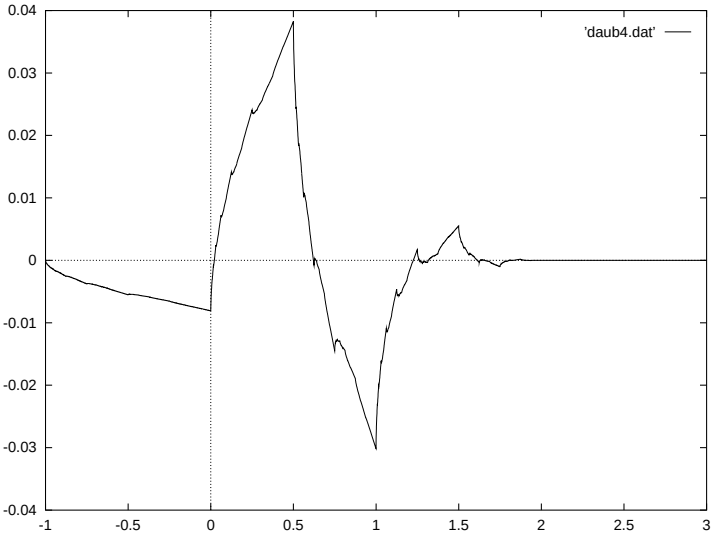

Abbildung 3: Das Daubechies-Wavelet der Ordnung 1

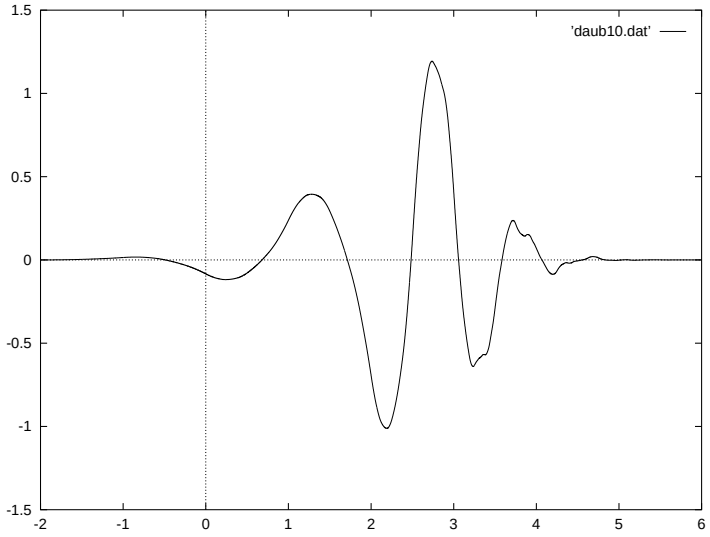

Abbildung 4: Das Daubechies-Wavelet der Ordnung 4

1.4 Die diskrete Wavelet-Transformation

Wir betrachten nun den Fall, daß nicht die gesamte Funktion f, sondern lediglich endlich viele Funktionswerte $y_1, \ldots, y_n$ an den Stellen $x_i = i/n$ bekannt sind. Die diskrete Wavelet-Transformation (DWT), die wir im folgenden definieren werden, ist eine diskretisierte Form von (1.3) und kann als lineare Transformation geschrieben werden, wobei eine orthogonale Matrix $\mathcal{W}$ verwendet wird, die sich aus dem Mutterwavelet ableitet. Die DWT von Daten $y = (y_1, \ldots, y_n)$ ist dann definiert als

$$w = \mathcal{W}y,$$

wobei die gewonnenen Wavelet-Koeffizienten w_i auch oft als empirische Koeffizienten bezeichnet werden. Würde man die DWT genauso implementieren wie geschrieben, so würde man einen Rechenaufwand von $O(n^2)$ benötigen. Für den Fall, daß $n = 2^J$ ist und die $x_1, \ldots, x_n$ gleichmäßig verteilt sind, existiert aber (wie bei der Fourier-Transformation) ein erstaunlich schneller Pyramiden-Algorithmus von Mallat (1989), der eine Komplexität von $O(n)$ aufweist.

Wir setzen nun voraus, daß man ein Vaterwavelet ϕ der Ordnung m zur Verfügung hat, das man für $h_i \in l_2$ darstellen kann als

$$\phi(x) = \sum_k h_k \phi(2x - k), \tag{1.11}$$

und daß das dazugehörige Mutterwavelet durch

$$\psi(x) = \sum_k g_k \phi(2x - k) \tag{1.12}$$

definiert ist, wobei $g_k = (-1)^k \overline{h}_{1-k}$. Diese Forderungen sind in der Praxis meistens erfüllt, insbesondere wenn die Wavelet-Basis gemäß der Konstruktion des letzten Abschnittes generiert ist.

Die folgende Idee steht hinter der Diskreten Wavelet-Transformation: ϕ_{jk} sei wieder wie in (1.7) definiert und $f \in L^2(\mathbb{R})$. Dann gilt

$$< f, \phi_{jk} >= \sum \overline{h}_k < f, \phi_{j+1,k} >$$

und

$$< f, \psi_{jk} >= \sum \overline{g}_k < f, \phi_{j+1,k} > .$$

Dies bedeutet aber, daß man, sobald alle Koeffizienten $< f, \phi_{Jk} >$ für einen festen Level $J \in \mathbb{Z}$ bekannt sind, die Koeffizienten $< f, \phi_{jk} >$ und $< f, \psi_{jk} >$ für alle $j < J$ und $k \in \mathbb{Z}$ ohne weitere Integralauswertungen bestimmen kann. Andererseits ist ein Koeffizient $< f, \phi_{jk} >$ ein gewichtetes und mit dem Faktor $2^{j/2}$ skaliertes Mittel von f in einer (bei großem j) kleinen Umgebung von $2^{-j}k$. Wenn wir also an f Glattheits- oder zumindestens Stetigkeitsbedingungen knüpfen, können wir diesen Koeffizienten durch den (skalierten) Funktionswert ersetzen, ohne daß die so entstandene Diskrepanz zwischen den tatsächlichen und den neu definierten Koeffizienten „allzu" groß wird. [10]

Es sei hier darauf hingewiesen, daß die DWT für die Samples von f natürlich in der Regel (d.h. wenn f nicht stückweise konstant auf $(k/n, (k+1)/n)$ ist) andere Koeffizienten berechnet als die stetige Wavelet-Transformation für die ursprüngliche Funktion f. Donoho merkt in

[10] Untersuchungen über diese Diskrepanz finden sich in der Doktorarbeit von Wim Swelden ([32]). Hier werden auch verschiedene andere Ideen diskutiert, um zu Startwerten für die diskrete Wavelet-Transformation zu gelangen, wie verschiedene Quadratur-Formeln.

[12] an, daß es manchmal verwirrend ist, daß ein und dasselbe Wort „Wavelet-Transformation" in zwei deutlich voneinander verschiedenen Wegen benutzt wird. In dem gleichen Artikel stellt er alternative Wavelet-Transformationen vor, die der gewohnten in vielen Punkten ähnlich sind (insbesondere auch auf einer Orthonormalbasis basieren, so daß man für Funktionen f die Darstellung (1.5) hat), aber zusätzlich die Eigenschaft aufweisen, daß die ersten n Koeffizienten der Wavelet-Entwicklung mit den empirischen Koeffizienten übereinstimmen.

Wir wollen die Idee der DWT konkretisieren und definieren zunächst, was wir unter einem linearen Filter verstehen wollen.

Definition 1.4. *Ein linearer Filter $\mathcal{F}$ ist eine stetige Abbildung $\mathcal{F} : l^2(\mathbb{Z}) \to l^2(\mathbb{Z})$, dessen Wirkungsweise auf eine doppelt unendliche Folge $(x_i)_{i\in\mathbb{Z}} \in l^2(\mathbb{R})$ beschrieben wird durch*

$$(\mathcal{F}x)_k = \sum_i f_{i-k}x_i, \tag{1.13}$$

wobei $(f_i)_{i\in\mathbb{Z}} \in l^2(\mathbb{Z})$.

Für Vektoren $x \in \mathbb{R}^n$ ist die Definition von $\mathcal{F}x$ von der Behandlung der Ränder abhängig. Üblich sind insbesondere periodische und symmetrische Randbedingungen, mit denen die Folge doppelt unendlich fortgesetzt wird.

Die diskrete Wavelet-Transformation basiert nun auf zwei Filtern $\mathcal{G}$ und $\mathcal{H}$ sowie einem „binären Dezimierungsoperator" $\mathcal{D}_0$. $\mathcal{G}$ sei durch die $\{g_k\}$ definiert, $\mathcal{H}$ durch die $\{h_k\}$. An dieser Stelle zeigt sich der Vorteil der Daubechies-Wavelets: Wenn Wavelets kompakten Träger haben, sind nur endlich viele der g_k und h_k von Null verschieden (bei den Daubechies-Wavelets: $2(m+1)$ Filterkoeffizienten, wobei m die Ordnung des Wavelets bezeichnet). Dies hat zur Folge, daß in (1.13) eine endliche Summe steht.

Definition 1.5. *Zwei Filter $\mathcal{F}^1, \mathcal{F}^2$, definiert über Folgen $(f_k^1)_{k\in\mathbb{Z}}, (f_k^2)_{k\in\mathbb{Z}} \in l^2(\mathbb{Z})$ heißen Quadrature Mirror Filter, wenn für beide die inneren Orthogonalitätsbeziehungen*

$$\sum_k f_k^1 f_{k+2l}^1 = 0 = \sum_k f_k^2 f_{k+2l}^2 \tag{1.14}$$

und

$$\sum_k (f_k^1)^2 = 1 = \sum_k (f_k^2)^2. \tag{1.15}$$

für alle $l \neq 0$ gelten und eine weitere gemeinsame Beziehung

$$\sum_k f_k^1 f_{k+2l}^2 = 0. \tag{1.16}$$

für alle $l \in \mathbb{Z}$ gilt.

Aus $(\phi 4)$ und $(\psi 4)$ folgt leicht, daß die Filter $\mathcal{G}$ und $\mathcal{H}$ Quadrature Mirror Filters sind. Gleichung (1.14) folgt zum Beispiel für $l \in \mathbb{Z} \setminus \{0\}$ wegen.

$$\begin{aligned} 0 = \int \psi(x)\psi(x+l)dx &= \int \left(\sum_{k_1\in\mathbb{Z}} g_{k_1}\phi(2x-k_1)\right) \cdot \left(\sum_{k_2\in\mathbb{Z}} g_{k_2}\phi(2(x+l)-k_2)\right) dx \\ &= \sum_{k_1\in\mathbb{Z}}\sum_{k_2\in\mathbb{Z}} g_{k_1}g_{k_2}\phi(2x-k1)\phi(2x-k_2+2l)dx \\ &= \sum_{k_1} g_{k_1}g_{k_1+l} \end{aligned}$$

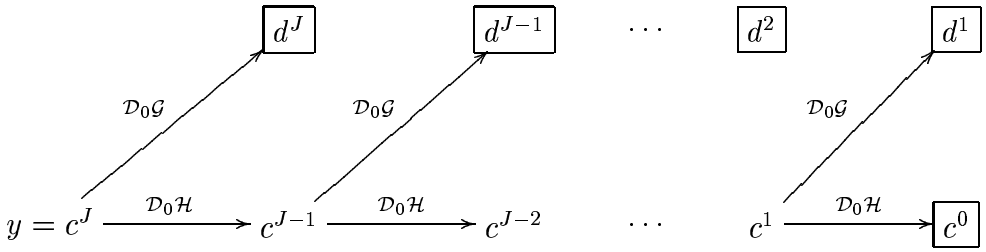

Abbildung 5: Die diskrete Wavelet-Transformation – Schema

Der *binäre Dezimierungsoperator* $\mathcal{D}_0$ wählt aus einer Folge jedes zweite Element aus:

$$(\mathcal{D}_0 x)_j = x_{2j}.$$

Aus den Orthogonalitätsbeziehungen (1.14), (1.15) und (1.16) folgt, daß die Abbildung einer Folge (x_n) auf das Folgenpaar $\{\mathcal{D}_0\mathcal{G}x, \mathcal{D}_0\mathcal{H}x\}$ eine orthogonale Abbildung ist. Wenn x ein Vektor der Länge 2^m ist und wir periodische Randbedingungen verwenden, bilden $\mathcal{D}_0\mathcal{G}$ und $\mathcal{D}_0\mathcal{H}$ den Vektor x jeweils auf einen Vektor der Länge 2^{m-1} ab.

Der Klarheit wegen setzen wir im folgenden periodische Randbedingungen voraus. Symmetrische Randbedingungen würden im wesentlichen nur die Notation erschweren. Es sei weiterhin $c^J := y$. Wir definieren nun rekursiv für $j = J, J-1, \ldots, 1$

$$d^j := \mathcal{D}_0\mathcal{G}c^j \quad \text{und} \quad c^{j-1} := \mathcal{D}_0\mathcal{H}c^j. \tag{1.17}$$

c^j heißt dabei der *Smooth beim Level* j, d^j die *Details beim Level* j. Aus den Filtereigenschaften folgt, daß die Abbildung $(\mathcal{D}_0\mathcal{G}, \mathcal{D}_0\mathcal{H})$ eine orthogonale Transformation ist. Man beachte, daß sowohl c^j wie d^{j+1} Vektoren der Länge 2^j sind. Desweiteren kann man in (1.17) ablesen, daß der Smooth beim Level j an den nächst tieferen Level übergeben wird, um damit den Smooth und die Details dieses Levels zu berechnen.

Ebenso leicht folgt aus der Orthogonalität der Abbildung $\{\mathcal{D}_0\mathcal{G}, \mathcal{D}_0\mathcal{H}\}$, wie man Gleichung (1.17) invertieren kann: Schreibe die Abbildung als eine Matrix und nehme die Transponierte.

Schließlich überlegen wir uns noch den Rechenaufwand, den dieser Schritt von Level j nach Level $j-1$ erfordert. Wenn die Filtersequenz h_n aus N hintereinander angeordneten Koeffizienten ungleich Null besteht, dann kann die Matrix der entsprechenden Abbildung höchstens $2^j N$ von Null verschiedene Einträge haben. Man kommt daher mit $2^j N$ Multiplikationen aus.

Die übliche diskrete Wavelet-Transformation ist nun die Abbildung, die dem Datenvektor y die Koeffizienten $d^J, d^{J-1}, \ldots, d^1, c^0$ zuordnet (siehe auch Abbildung 5). Man benötigt $O(n)$ Rechenoperationen, um für einen Datenvektor die Transformierte oder umgekehrt zu berechnen, wie Aufsummieren der $2^j N$ sofort zeigt.

Mit Induktion kann man zeigen, daß ein Detail-Koeffizient d^j_k von $N \cdot 2^{J-j} + (J-j)(N-2)$ Daten der doppelt unendlich fortgesetzten Folge abhängt, falls N der h_i von Null verschieden sind. Diese Zahl ist im Falle $N > 2$ und hinreichend kleinem j größer als $n = 2^J$, was zur Folge hat, daß bei periodischen oder symmetrischen Randbedingungen sowie $N > 2$ bei der Berechnung der Waveletkoeffizienten auf grober Skala einige Datenwerte mehrfach berücksichtigt werden. Wir definieren

$$j_0 := \min\{j \in \mathbb{N} : 2^{J-j} + (J-j)(N-2) \leq n\}. \tag{1.18}$$

Als nächstes zeigen wir, daß die Eigenschaft der verschwindenden Momente $(\psi 3)$ im wesentlichen nicht verlorengeht und beweisen das folgende Lemma.

Lemma 1.1. *Für die Filterkoeffizienten $(g_k)_{k\in\mathbb{Z}}$ einer Wavelet-Basis der Ordnung m und für alle $l \in \{0,\dots,m\}$ gilt, daß*

$$\sum_k k^l g_k = 0. \tag{1.19}$$

Beweis. Aus $(\psi 3)$ folgt für alle $0 \le l \le m$, daß

$$\begin{aligned} 0 &= \int x^l \psi(x) dx \\ &= \int x^l \sum_k g_k \phi(2x-k) dx \\ &= \sum_k g_k \int x^l \phi(2x-k) dx \\ &= \sum_k g_k \int \frac{1}{2^{l+1}} (y+k)^l \phi(y) dy \end{aligned} \tag{1.20}$$

Falls $l = 0$, so steht hier wegen $(\phi 3)$ bereits die Behauptung. Für $l \in \{1,\dots,m\}$ schließen wir induktiv und nehmen an, (1.20) wäre schon gezeigt für $l-1$, dann formen wir weiter um und erhalten:

$$\begin{aligned} 0 &= \frac{1}{2^{l+1}} \sum_k g_k \int \sum_i \binom{l}{i} y^i k^{l-i} \phi(y) dy \\ &= \frac{1}{2^{l+1}} \sum_i \binom{l}{i} \left(\int y^i \phi(y) dy \right) \sum_k k^{l-i} g_k \end{aligned} \tag{1.21}$$

Die Summanden für $i > 0$ fallen alle nach Induktionsvoraussetzung weg und die Behauptung für l folgt. □

Es folgt direkt, daß die Wavelet-Koeffizienten der DWT mit feinster Skala $w_{J-1,k}$ verschwinden, falls die dazugehörigen Funktionswerte von f auf einem Polynom vom Grad kleiner oder gleich m liegen. Die gleiche Aussage kann man aber leicht für Koeffizienten w_{jk} mit $j_0 \le j < J$ verallgemeinern, wenn man sich klar macht, daß der Filter $\mathcal{H}$ aus Polynomen l-ten Grades wieder Polynome l-ten Grades macht.

Zum Schluß betrachten wir zwei Beispiele. Wir beginnen mit der Funktion $f_1(x) = \sin\left(\frac{\pi}{x}\right)$, bei der die Frequenz in der Zeit variiert (Nason und Silvermann nennen diese Funktion „Chirp"). Die Abbildung 6 zeigt einerseits die Funktion selbst, andererseits die Koeffizienten einer diskreten Wavelettransformation für diese Funktion. [11] Hierbei wird jeder Koeffizient durch einen senkrechten Balken dargestellt, dessen Länge von der Größe des entsprechenden Koeffizienten abhängt. Unten stehen die Koeffizienten mit hoher Frequenz, nach oben hin wird die Frequenz immer kleiner.

Aus dem Ergebnis der Transformation ist deutlich erkennbar, daß hohe Frequenzen in der Mitte des betrachteten Zeitintervalles und kleine Frequenzen am Rand auftreten. Dies ist gerade die Eigenschaft der Wavelet-Transformation, in Zeit *und* Frequenz zu lokalisieren. Bei einer Fourier-Transformation hätte man als Ergebnis nur gewonnen, *welche* Frequenzen auftreten, aber nicht *wann*.

[11] Die Funktion wurde dazu zunächst an den 1024 Punkten $x_i = i/512 - 1$ für $i = 1,\dots,1024$ gesamplet. Für die DWT wurde das Daubechies-Wavelet der Ordnung 3 verwendet.

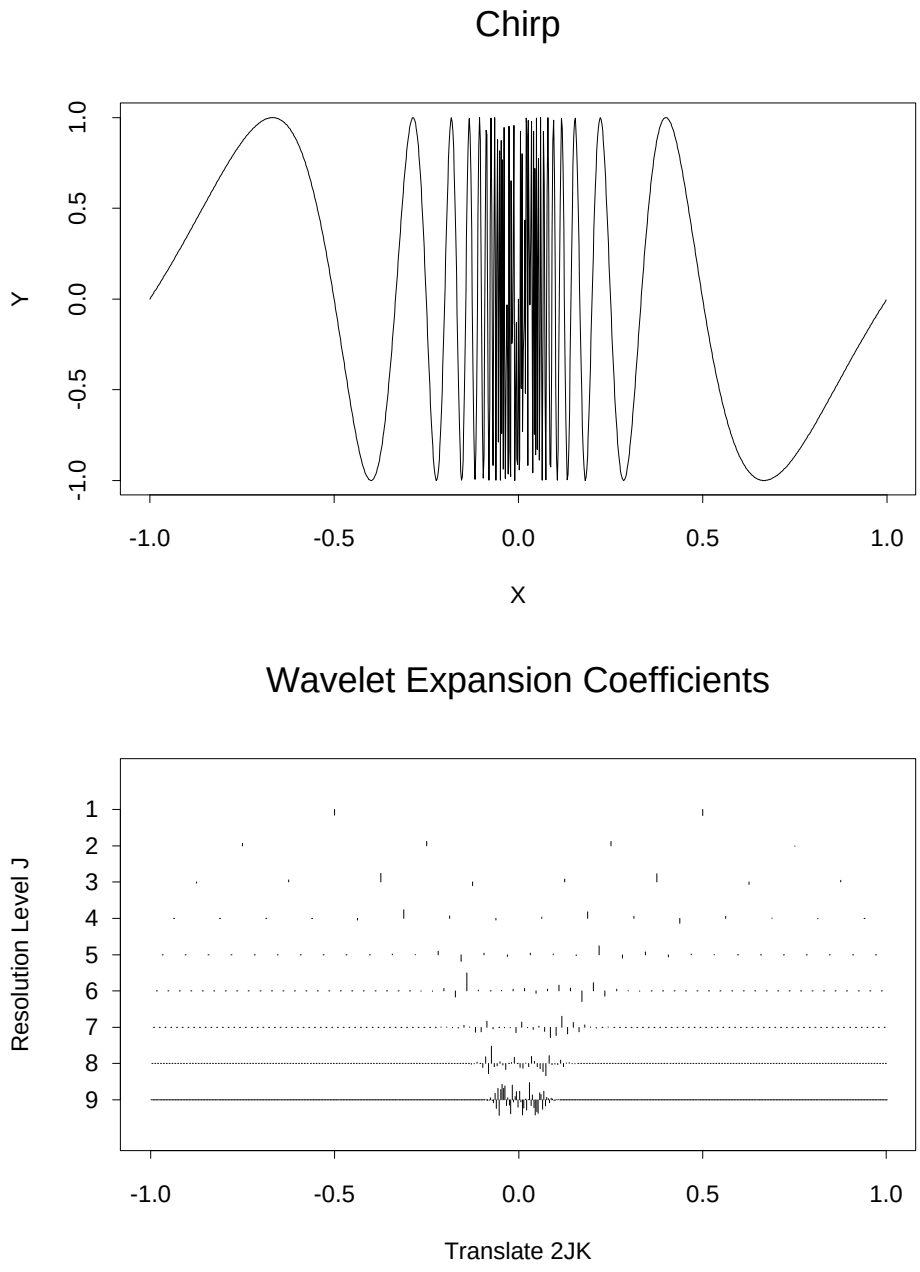

Abbildung 6: Die Funktion „Chirp” und ihre Wavelet-Koeffizienten

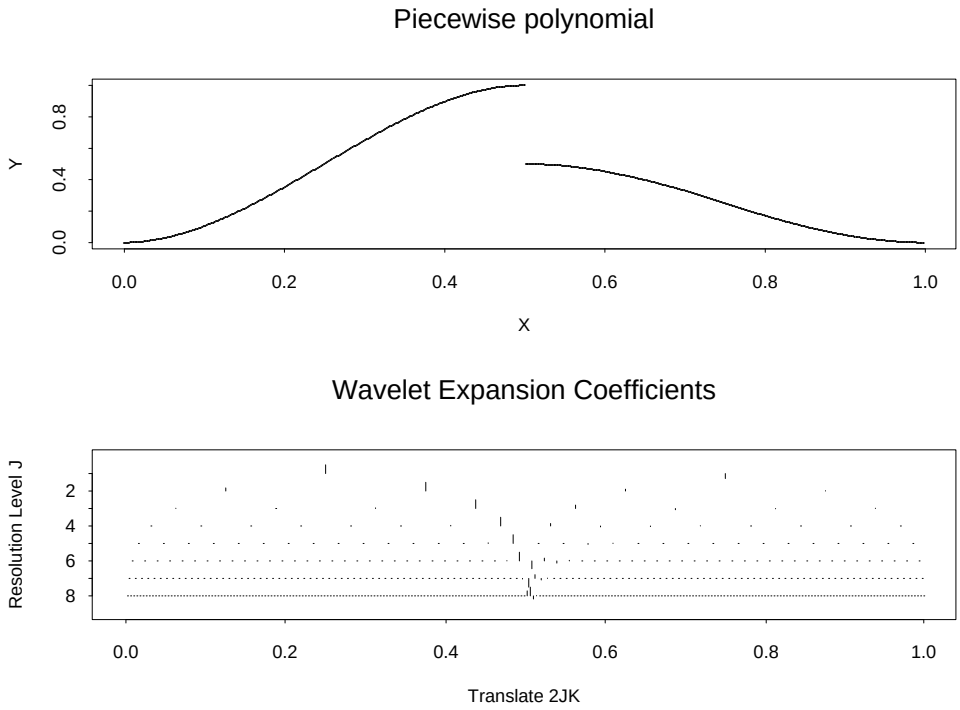

Abbildung 7: Ein stückweises Polynom dritten Grades und ihre Wavelet-Koeffizienten

Das zweite Beispiel zeigt in Abbildung 7 eine Funktion, die sich stückweise aus drei Polynomen dritten Grades zusammensetzt, nämlich

$$f_2(x) = \begin{cases} 4x^2(3-4x), & x \in [0, \frac{1}{2}] \\ \frac{4}{3}x(4x^2-10x+7) - \frac{3}{2}, & x \in [\frac{1}{2}, \frac{3}{4}] \\ \frac{16}{3}x(x-1)^2, & x \in [\frac{3}{4}, 1] \end{cases} \tag{1.22}$$

Die folgende Tabelle zeigt, wieviele Terme aus der Fourier-Entwicklung von f_2 nötig sind, um f_2 genauso gut zu approximieren, wie 5, 10, 15, 20 und 50 Wavelet-Terme. Das „genauso gut" wurde dabei durch die l^2-Norm der Residuen gemessen.

Anzahl Wavelet-Koeffizienten	Anzahl Fourier Koeffizienten	l^2-Norm der Residuen
5	12	1.60
10	43	0.78
15	191	0.38
20	1149	0.19
50	Alle	0.01

Eine ähnliche Untersuchung haben auch Nason und Silverman in [24] durchgeführt und sind qualitativ zu dem gleichen Ergebnis gekommen. Man sieht deutlich, daß die Wavelet-Transformation wesentlich ökonomischer arbeitet, das heißt, die gleiche Information wird durch weniger Koeffizienten repräsentiert.

2 Wavelet Shrinkage

Ein klassisches Problem der Statistik ist die nicht-parametrische Regression. Dabei betrachtet man Daten y_i, von denen man annimmt, daß man sie modellieren kann als

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

wobei f die Funktion ist, die wir schätzen wollen, und ε_i Rauschen. Anwendungsgebiete umfassen unter anderem die Bild- und Signalverarbeitung sowie die Erkennung von Sprache.

Wir stellen zunächst drei klassische Verfahren vor, nämlich Kernschätzer, Spline-Smoother und Fourier-Schätzer. Allen gemeinsam ist ihr Nachteil, daß Glattheit jeweils *global* justiert wird. Das heißt, daß das Ausmaß, wie stark geglättet wird, an jeder Stelle gleich ist. Dies führt bei der Anwendung oft zu dem Problem, daß entweder durch eine zu große Wahl des Parameters lokale Extrema weggeglättet werden oder aber durch eine kleinere Wahl die Funktion an manchen Stellen oszilliert.

Im folgenden Abschnitt wird dann Wavelet Shrinkage als eine Alternative vorgestellt. Hier wird auf die Daten die Diskrete Wavelet-Transformation angewendet, kleine Wavelet-Koeffizienten werden auf 0 gesetzt und anschließend wird auf die geänderten Daten wieder die inverse DWT angewendet. Es werden verschiedene Möglichkeiten gezeigt, wie man die Koeffizienten im zweiten Schritt modifizieren kann, einige Beispiele präsentiert und eine Reihe von theoretischen Aussagen über solche Wavelet-Schätzer zitiert.

Das dritte Kapitel dieses Teils beschäftigt sich schließlich mit den Auswirkungen von Ausreißern auf Wavelet-Schätzer. Es zeigt sich, daß Wavelet Shrinkage sehr empfindlich reagiert, was durch zwei Lemmata präzisiert wird.

2.1 Nicht-parametrische Regression

In Zusammenhang mit einer Untersuchung über Motorradhelme, die das Institut für Rechtsmedizin der Universität Heidelberg 1980 durchführte, wurden mit Crashtests Motorradunfälle simuliert. Abbildung 8 zeigt Daten, die bei einem Experiment anfielen, nämlich die Beschleunigung, der der Kopf eines Motorradfahrers beim Aufprall ausgesetzt ist. Ein Aspekt bei der Auswertung dieses Versuches bestand darin, Aussagen über den Zusammenhang zwischen Zeit und Beschleunigung zu machen, insbesondere eine Kurve anzugeben, die die Beschleunigung in Abhängigkeit von der Zeit darstellt.

Bevor wir im nächsten Abschnitt zeigen, wie man mit Hilfe der Diskreten Wavelet-Transformation zu solchen Kurven gelangen kann, skizzieren wir kurz klassische Verfahren und die Probleme, die sich bei deren Anwendung auf Datensätze ergeben.

Dazu seien Daten $y_1, \ldots, y_n$ gegeben, von denen wir annehmen, daß wir sie modellieren können als

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

wobei f die Funktion ist, die wir schätzen wollen, und ε_i unkorreliertes Rauschen mit $\mathbb{E}(\varepsilon_j) = 0$.

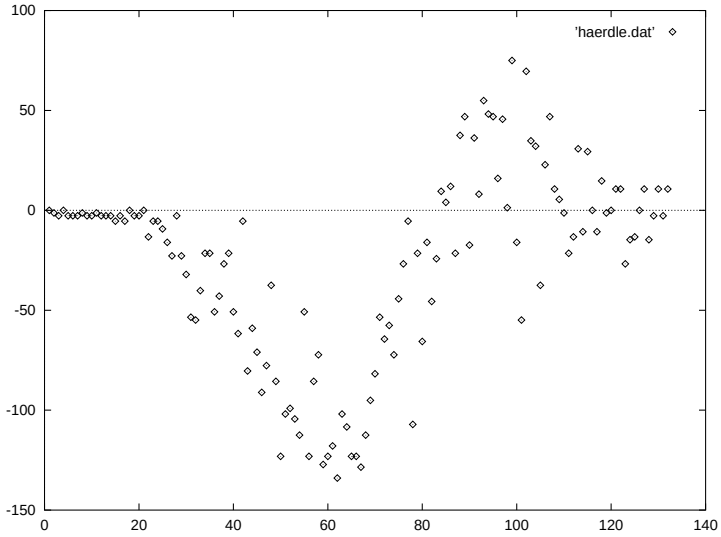

Abbildung 8: Die Motorrad-Daten

2.1.1 Kernschätzer

Wir betrachten zunächst *Kernschätzer*. Dies sind besonders einfache Schätzer der Form [12]

$$\hat{f}(t) = \frac{\sum_{i=1}^{n} y_i K(\frac{t-t_i}{\lambda})}{\sum_{i=1}^{n} K(\frac{t-t_i}{\lambda})}$$

wobei $K : \mathbb{R} \to \mathbb{R}$ die folgenden Eigenschaften aufweisen soll (siehe zum Beispiel [18]):

1. $\text{supp}K \subset [-1; 1]$,
2. $\max_{t\in[-1;1]} K(t) = K(0)$,
3. $\int_{-1}^{1} K(u)du = 1$,
4. $\int_{-1}^{1} uK(u)du = 0$,
5. $\int_{-1}^{1} u^2K(u)du = \alpha \neq 0$,
6. $\int_{-1}^{1} K^2(u)du < \infty$.

Der Parameter $\lambda > 0$ ist ein Glattheitsparameter und wird als *Bandbreite* bezeichnet. Für kleines λ ist $\hat{f}(t)$ stark abhängig von den Daten nahe t, so daß der Kernschätzer eine rauhe Kurve produziert. Läßt man $\lambda \to 0$ konvergieren, so konvergiert $\hat{f}(t_i) \to t_i$ sowie $\hat{f}(t) \to 0$ für $t \neq t_i$. Bei großem Lambda hingegen werden mehr Daten gemittelt und die Kurve ist glatter. Für $\lambda \to \infty$ konvergiert $\hat{f}(t)$ gegen den Mittelwert der Daten

In der Literatur existieren unzählige Vorschläge für die Wahl von K und λ, wobei jede den Kernschätzer bezüglich eines anderen Kriteriums „optimiert". Wir wollen auf diese verschiedenen Möglichkeiten erst gar nicht eingehen, denn allen gemeinsam ist auch ein Problem, das

[12] Eubank stellt in [18] noch vier weitere Varianten von Kernschätzern vor, die aber nach eigener Aussage ähnliche Eigenschaften aufweisen. Wir wollen auf diese Feinheiten nicht näher eingehen.

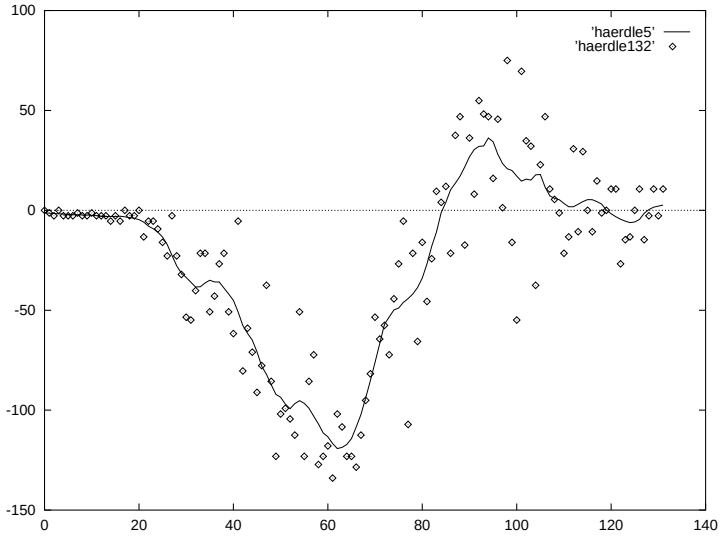

Abbildung 9: Kernschätzer angewandt auf die Motorrad-Daten mit $\lambda = 5$

durch die **globale** Wahl des Glattheitsparameters entsteht: Bei Datensätzen mit deutlich ausgeprägten lokalen Extrema ist entweder die Schätzfunktion nicht mehr glatt oder aber die lokalen Extrema werden abgeschnitten. Wir illustrieren dieses Problem, indem wir auf die Motorrad-Daten aus Abbildung 8 einen Kern-Schätzer anwenden. Der Kern wurde dabei als

$$K(u) = \frac{3}{4}(1 - u^2)\{|u| \leq 1\}$$

gewählt und wurde 1979 von Gasser und Müller ([21]) vorgeschlagen. Abbildung 9 zeigt das Resultat mit $\lambda = 5$: Die Schätzfunktion ist sehr rauh, obwohl das Minimum bei 60 noch immer knapp überschätzt wird. Wählt man auf der anderen Seite $\lambda = 9$, so wird das Minimum bereits stark überschätzt, obwohl die Kurve im Intervall $[80; 132]$ noch immer nicht glatt ist (siehe Abbildung 10).

Wir illustrieren dieses Problem noch mit einem zweiten Beispiel. Abbildung 11 zeigt den Datensatz „Doppler". [13] Es handelt sich um die Funktion

$$f(t) = 21 \cdot \sqrt{t(1-t)} \sin\left(2\pi \frac{1 + .05}{t + 0.05}\right),$$

die an den Punkten $t_i = i/2048$ mit standardnormalverteiltem Rauschen gestört wurde.

Das Problem bei diesem Datensatz besteht darin, daß die verschiedenen Frequenzen der Funktion f erfordern, daß rechts mehr geglättet werden muß als links. Dies wird durch die globale Bandbreite λ jedoch verhindert. Abbildung 12 zeigt den Kernschätzer mit $\lambda = 40$. Um eine glatte Kurve zu erhalten, darf man λ keinesfalls viel kleiner wählen, denn im Intervall $[1500; 2000]$ zeigen sich bereits erste Unregelmäßigkeiten. Andererseits wird bereits das lokale Extremum bei 680 deutlich überschätzt. Zum Vergleich zeigt Abbildung 13 das Ergebnis mit $\lambda = 8$.

[13] Dieser Datensatz dient auch in einer Reihe von Artikeln über nicht-parametrische Regression mit Wavelets von D. L. Donoho als Beispiel (siehe zum Beispiel [17] oder [13]).

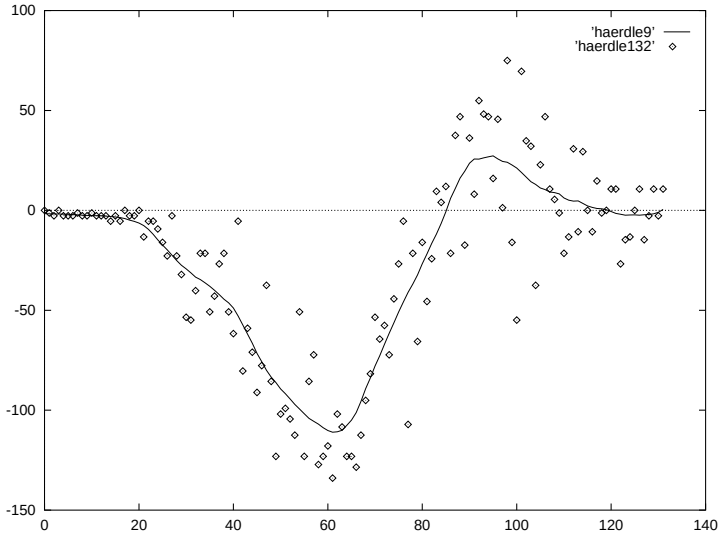

Abbildung 10: Kernschätzer angewandt auf die Motorrad-Daten mit $\lambda = 9$

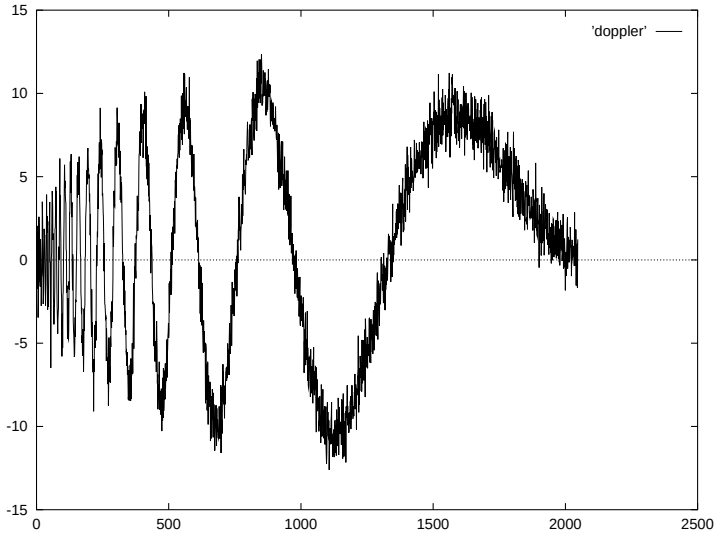

Abbildung 11: Der Datensatz „Doppler”

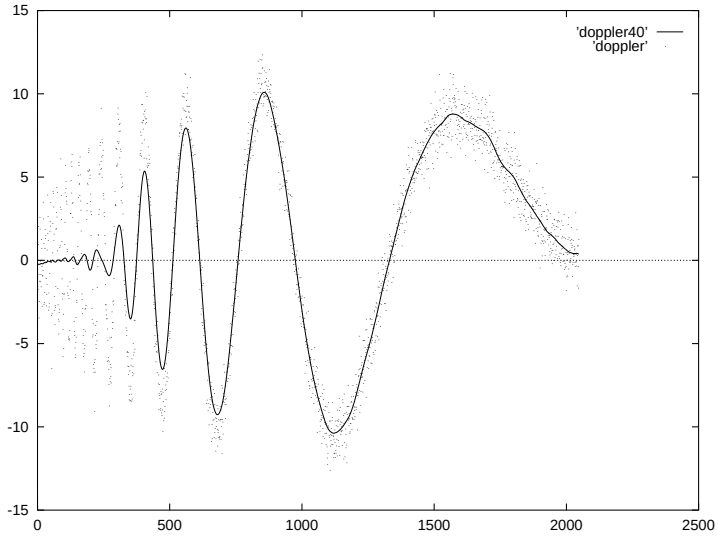

Abbildung 12: Kernschätzer angewandt auf den Datensatz „Doppler” mit $\lambda = 40$

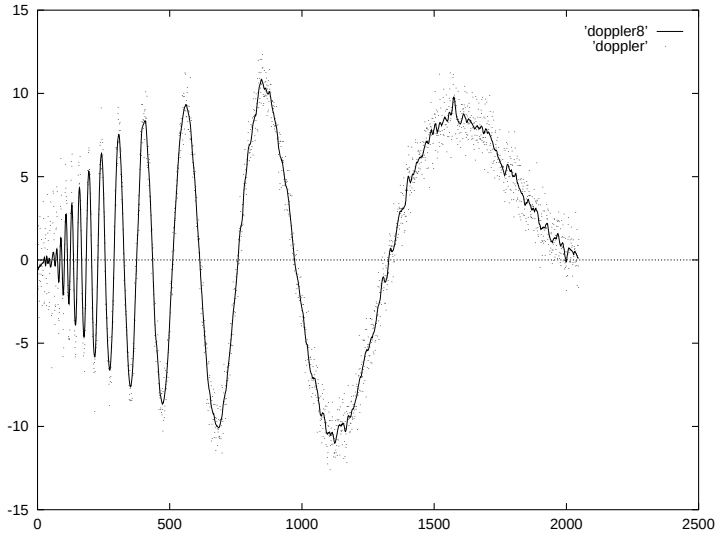

Abbildung 13: Kernschätzer angewandt auf den Datensatz „Doppler” mit $\lambda = 8$

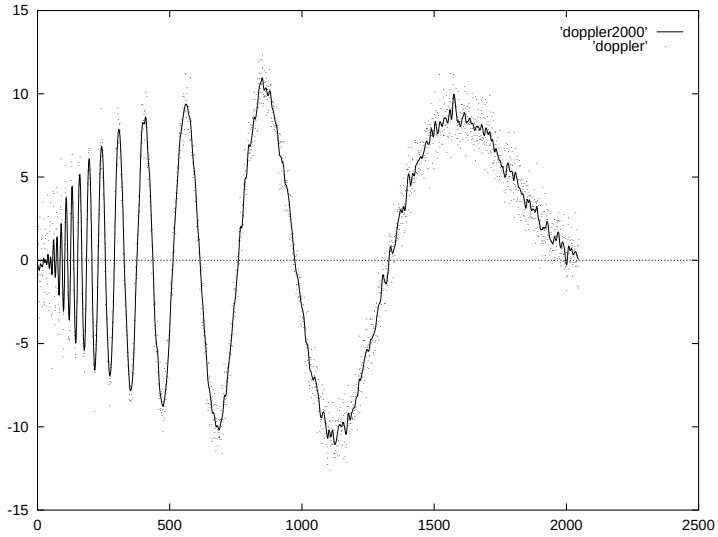

Abbildung 14: Spline Smoothing angewandt auf den Datensatz „Doppler" mit $\lambda = 43$

2.1.2 Spline-Smoothing

Als nächstes betrachten wir *Spline-Smoothing*. Die Idee besteht darin, daß eine „gute" Schätzfunktion zum einen zu den Daten passen muß, andererseits glatt sein soll. Diesen Konflikt nutzt man aus, indem man zwei Maße konkurrieren läßt: ein „goodness-of-fit"-Maß, das die Güte der Approximation mißt wie zum Beispiel den Abstand der Daten zu der Schätzkurve in der l^2-Norm, und ein Glattheitsmaß, das die Glattheit von $\hat{f}$ mißt wie zum Beispiel das Integral über das Quadrat der zweiten Ableitung. Man wählt dann als Schätzfunktion $\hat{f}$ die zweimal differenzierbare Funktion $\hat{f}$, für die der Term

$$S(\hat{f}) = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{f}(t_j))^2 + \lambda \int \hat{f}(x)^2 dx$$

minimal wird, wobei λ ein Glattheitsparameter ist. Für $\lambda = 0$ interpoliert $\hat{f}$ die Daten und für $\lambda \to \infty$ konvergiert $\hat{f}$ gegen den linearen „Least Squares"-Schätzer. Man kann zeigen, daß $\hat{f}$ zwischen zwei Datenpunkten t_i und t_{i+1} (vorausgesetzt, daß $t_j \leq t_{j+1}$ für alle $j < n$ ist) ein kubisches Polynom ist.

Auch für die Wahl von λ werden in der Literatur eine Reihe von Vorschlägen diskutiert (siehe zum Beispiel [29]), doch es ergibt sich wieder das gleiche Problem wie bei Kernschätzern, da Glattheit global definiert wird. Die Abbildungen 14 und 15 zeigen den Spline-Schätzer für den „Doppler"-Datensatz mit $\lambda = 43$ und für $\lambda = 61580$.

2.1.3 Fourier-Reihen-Schätzer

Schließlich werfen wir noch einen Blick auf *Fourier-Reihen-Schätzer*, die ähnlich definiert werden wie die Wavelet-Schätzer, die wir im nächsten Kapitel untersuchen. Wir setzen dazu voraus, daß die Zeitpunkte t_j gleichmäßig auf dem Intervall $[0; 1]$ liegen:

$$t_i = (i-1)/n.$$

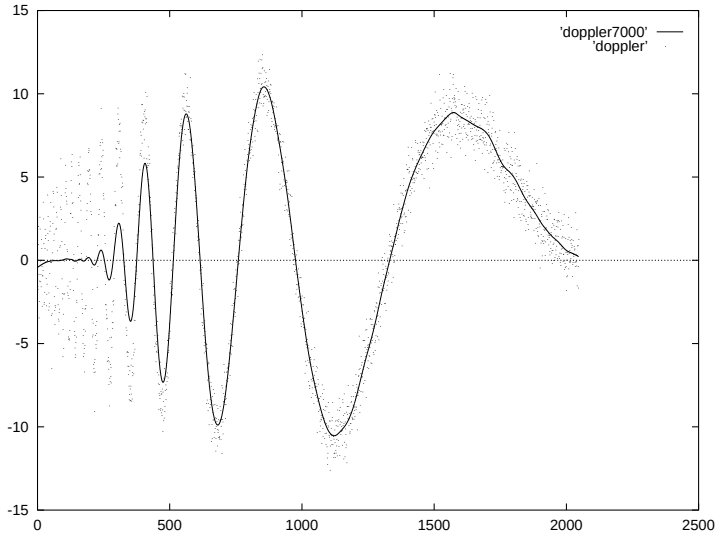

Abbildung 15: Spline Smoothing angewandt auf den Datensatz „Doppler" mit $\lambda = 61580$

Die Idee, die diesem Schätzer zugrunde liegt, ist (wie zu Beginn dieser Arbeit bereits erläutert), daß sich jede über $[0, 1]$ L^2-integrierbare Funktion f in ihre Fourier-Reihe

$$f(t) = \sum_{j=1}^{\infty} \beta_j \exp(2\pi i j t)$$

entwickeln läßt. Da $(\beta_j)_{j\in\mathbb{N}} \in l_2$, gibt es ein $\lambda \in \mathbb{N}$, so daß die endliche Summe

$$f_\lambda = \sum_{j=1}^{\lambda} \beta_j \exp(2\pi i j t)$$

eine gute Näherung für f darstellt, und es gilt

$$y_i \doteq \sum_{j=1}^{\lambda} \beta_j \exp(2\pi i j t_i) + \varepsilon_i. \tag{2.1}$$

Man erhält also für jedes λ ein lineares Modell, bei dem man Methoden der linearen Regression anwenden kann, um die Koeffizienten β_j zu bestimmen. Der Kleinste-Quadrate-Schätzer liefert insbesondere nach einer Reihe von Umformungen (siehe zum Beispiel [18])

$$\hat{f}_\lambda(t) = \frac{1}{n} \sum_{k=1}^{n} y_k K_\lambda(t - t_k),$$

wobei K_λ der *Dirichlet-Kern*

$$K\lambda(s) = \begin{cases} \frac{\sin(\pi(2\lambda+1)s}{\sin(\pi s)}, & s \notin \mathbb{Z} \\ 2\lambda + 1, & \text{sonst} \end{cases}$$

ist.

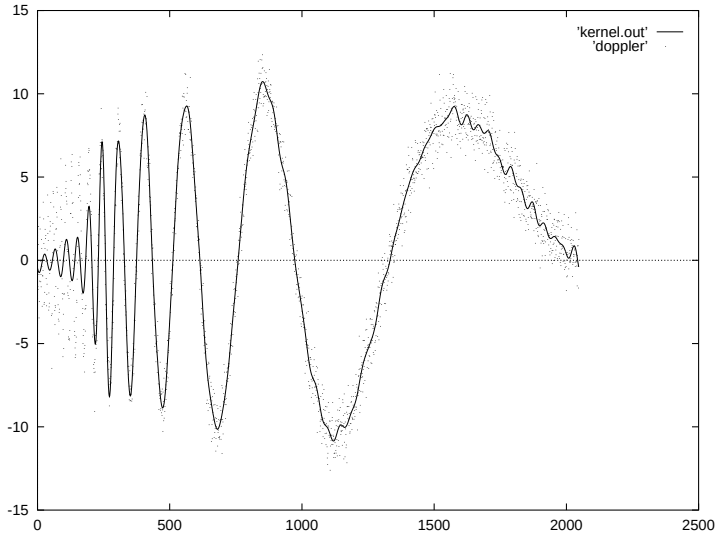

Abbildung 16: Fourier-Reihen-Schätzer angewandt auf den Datensatz „Doppler" mit $\lambda = 50$

Damit sind wir jedoch (im wesentlichen) wieder in der Situation von Kern-Schätzern, und es ist klar, daß die oben geschilderten Probleme auch bei Fourier-Reihen-Schätzern ihre Gültigkeit beibehalten, siehe zum Beispiel Abbildung 16. Nach der Diskussion des ersten Kapitels ist weiterhin offensichtlich, daß auch andere Wahlen der Koeffizienten β_j als Kleinste-Quadrate-Schätzer oder der Übergang zu beliebigen endlichen Teilsummen in (2.1) anstatt der ersten J Terme das Problem nicht lösen können, da die Fourier-Basis nicht in der Zeit, sondern nur in der Frequenz lokalisiert.

2.1.4 Fazit

Die globale Glattheit führt bei allen vorgestellten Schätzern zu dem gleichen Problem: Wendet man sie auf Datensätze an, für die ein adäquates Modell deutliche lokale Extrema aufweist, entstehen entweder rauhe Kurven oder die Extrema werden abgeschwächt oder sogar abgeschnitten. Gesucht sind also Verfahren, bei denen Glattheit anders behandelt wird. Folgende Ansätze wurden in der jüngeren Vergangenheit vorgeschlagen.

(Variable Wahl der Bandbreite) J. Fan und I. Gijbels stellten 1995 in [19] ein Verfahren vor, bei dem die Bandbreite lokal bestimmt wird. In den Beispielen, unter denen auch der „Doppler"-Datensatz ist, liefert das Verfahren stets ausgezeichnete Ergebnisse.

(Lokale Extrema) P. L. Davies führte ebenfalls 1995 in [9] ein anderes Glattheitskriterium ein, nämlich die Anzahl der lokalen Extremstellen. Dieses Kriterium wird minimiert unter allen Funktionen, bei denen die maximale Runlänge der Vorzeichen der Residuen einen theoretisch errechneten Wert nicht übersteigt, und liefert für jeden Zeitpunkt t_i ein Intervall, durch das eine Schätzfunktion $\hat{f}$ laufen muß.

(Wavelet-Shrinkage) Wavelet-Shrinkage ist Thema der nächsten Kapitel und wurde erstmals 1992 von D. L. Donoho, S. Mallat und anderen vorgeschlagen (siehe zum Beispiel [14]).

Der Ansatz ist der gleiche wie bei Fourier-Reihen-Schätzern – mit dem Unterschied, daß anstatt der Fourier-Basis die Wavelet-Basis benutzt wird. Durch Ausnutzen der Lokalisation in der Zeit-Variablen variiert das Ausmaß, mit dem geglättet wird lokal.

2.2 Wavelet-Regression

Wie angekündigt wollen wir in diesem Abschnitt sehen, wie mit Hilfe der Diskreten Wavelet-Transformation ein einfaches Verfahren zur nicht-parametrischen Regression konstruiert werden kann. Wir wandeln die Problemstellung leicht ab und fordern, daß die Zeitpunkte t_i gleichmäßig in $[0;1]$ verteilt sind mit

$$t_i = i/n$$

und daß die Anzahl der Datenpunkte eine Zweierpotenz

$$n = 2^J$$

ist, damit wir die DWT anwenden können. Falls n keine Zweierpotenz ist, kann man den Datensatz auf verschiedene Arten und Weisen erweitern, zum Beispiel durch Spiegeln, periodisches Fortsetzen, Ergänzen von Nullen, Wiederholen der Randwerte und so weiter. Eine Reihe dieser Ideen sind von Smith und Eddins in [30] analysiert worden. Delyon und Juditsky stellen in [11] aufwendigere Methoden vor, die auch mit dem Fall, daß die Zeitpunkte nicht gleichverteilt sind, fertig werden.

Wavelet-Regression-Schätzer arbeiten in der Regel wie folgt:

Algorithmus 2.1.

(1) Wende auf die Originaldaten $y = (y_1, \ldots, y_n)$ die diskrete Wavelet-Transformation an:

$$w = \mathcal{W}y$$

(2) Verändere die Wavelet-Koeffizienten und erhalte neue Koeffizienten w^*.

(3) Schätze f durch die inverse DWT:

$$\hat{f}_n^*(t_i)_1^n = \mathcal{W}^T w^*.$$

Der kritische Teil ist selbstverständlich der zweite Schritt. Die verbreitetesten Methoden kann man unter dem Begriff *Thresholding* zusammenfassen. Grundgedanke ist, kleine Koeffizienten auf 0 zu setzen und große zu behalten oder einzuschränken. Unterschiede gibt es in der Art und Weise, wie dies realisiert wird. Zunächst unterscheiden wir zwei *Thresholding-Regeln*, die wir mit zwei Funktionen identifizieren, die auf die Wavelet-Koeffizienten angewandt werden, nämlich *Hard-Thresholding*

$$T_{hard}(w_{jk}; t) = w_{jk}\{|w_{jk}| > t\}$$

und *Soft-Thresholding*

$$T_{soft}(w_{jk}; t) = \operatorname{sign}(w_{jk})(|w_{jk}| - t)\{|w_{jk}| > t\}.$$

Hard-Thresholding wird oft als *keep-or-kill*-Strategie, Soft-Thresholding als *shrink-or-kill*-Strategie bezeichnet.

Die einfachste und deswegen wohl auch verbreiteste Methode zur Wahl eines *Thresholds* heißt *VisuShrink* und wurde erstmals 1992 von Donoho in [14] vorgestellt. [14]

Hier wählt man den Threshold global als

$$t^{VS} = \sqrt{(2\log(n))\sigma} \tag{2.2}$$

Der Faktor $\sqrt{(2\log(n))}$ sichert, daß der VisuShrink-Schätzer asymptotisch rauschfrei ist, denn für eine Folge von weißem Rauschen gilt:

$$\mathbb{P}\left(\left\{\max_i |z_i| > \sqrt{2\log(n)}\right\}\right) \to 0, \quad n \to \infty.$$

Die Standardabweichung σ kann man in (2.2) (falls unbekannt) durch den MAD der Wavelet-Koeffizienten des feinsten Levels

$$\hat{\sigma} = \mathrm{MAD}((w_{J-1,k})_k)/0.6745 \tag{2.3}$$

schätzen, denn aufgrund der Orthonormalität von $\mathcal{W}$ ist die DWT von weißem Rauschen wieder weißes Rauschen und der Einfluß des Signals sollte bei den meisten *Fine-Scale*-Koeffizienten wegen $(\psi 3)$ nicht mehr spürbar sein, wenn man voraussetzt, daß f lokal hinreichend glatt ist. Man beachte dabei allerdings, daß mit linear wachsender Regularität des Wavelets die Anzahl der Datenpunkte, die diese Koeffizienten beeinflussen, ebenfalls linear wächst. Genaue Untersuchungen dieses Skalen-Schätzers sind mir nicht bekannt. Desweiteren weist Donoho in vielen seiner Artikel darauf hin, daß man T_{hard} und T_{soft} nur auf die Wavelet-Koeffizienten w_{jk} mit $j \geq j_0$ anwenden soll, wobei ursprünglich (in [15]) der *Cut-Off-Point* j_0 wie in (1.18) definiert wird, da für kleinere Skalen die Eigenschaft der verschwindenden Momente fehlt (siehe oben). In späteren Artikeln und bei anderen Autoren wird auf diese spezielle Wahl von j_0 nicht eingegangen. Ganz im Gegenteil betrachtet Donoho in [17] Asymptotik, hält aber das j_0 offenbar fest, wenigstens „scheint das die Intention der Autoren zu sein", wie Peter Hall und Prakash Patill in der Diskussion zu dem Artikel anmerken. Hall und Patil schlagen weiterhin vor, j_0 mit wachsendem n zu vergrößern, ohne freilich Details anzugeben. Die einzige (mir bekannte) Arbeit, bei der (unter anderem) der Einfluß von verschiedenen j_0 auf den Wavelet-Schätzer untersucht wird, stammt von Abramovich und Benjamini (1995) – aber auch hier wird nur durch eine Simulationsstudie die These belegt, daß ein solcher Einfluß in der Tat vorhanden ist.

[14] Zwei weitere beliebte Möglichkeiten sind:

1. *SureShrink:* Dieses Verfahren zur Bestimmung eines Threshold basiert auf Steins biasfreien Risiko-Schätzer ([31]) und wurde in [16] von Donoho und Johnstone vorgeschlagen. SureShrink spezifiziert für jeden Level $j_0 \leq j \leq J$ einen eigenen Threshold.

2. *Cross-Validation:* Hier wird versucht, einen Threshold zu wählen, der

$$M(t) = \mathbb{E}\left(\int \left(\hat{f}_t(x) - f(x)\right)^2 dx\right)$$

minimiert. Um für ein $t > 0$ einen Schätzwert für $M(t)$ zu erhalten, stellt Nason in [25] eine Variation der klassischen Cross-Validation vor, bei der nicht jeder einzelne Datenpunkt weggelassen wird (weil dann die Anzahl der verbleibenden Datenpunkte keine Potenz von 2 wäre), sondern bei der erst auf die ungeraden, dann auf die geraden Datenpunkte verzichtet wird.

Außerdem gibt es Ansätze von Abramovich und Benjamani in [1], bei denen für jeden Wavelet-Koeffizienten einzeln die Hypothese überprüft wird, ob er auf 0 gesetzt wird oder nicht, sowie Bayesianische Methoden von Vidakovich in [33].

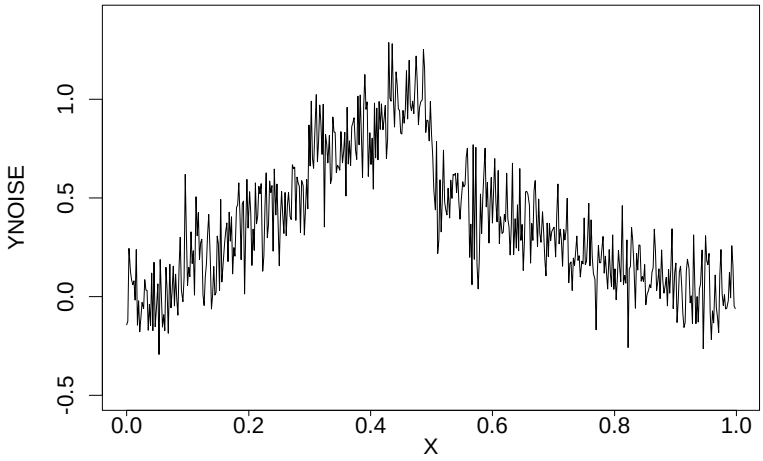

Abbildung 17: Das stückweise definierte Polynom mit Rauschen

Wir betrachten nun wiederum das stückweise definierte Polynom dritten Grades, das wir oben in (1.22) definiert haben und addieren weißes Rauschen mit Standardabweichung 0.15 (siehe Abbildung 17). Abbildung 2.2 zeigt die korrespondierenden Wavelet-Koeffizienten bzgl. des Daubechies-Wavelets der Ordnung 1. In Abbildung 2.2 kann man sehen, wie die Wavelet-Koeffizienten nach Anwendung eines globalen Thresholds aussehen, und Abbildung 2.2 zeigt das Ergebnis der Rekonstruktion mit der inversen Wavelet-Transformation. Schließlich zeigt Abbildung 2.2 das Ergebnis des Thresholding-Schätzers, wenn man Daubechies-Wavelets der Ordnung 4 benutzt.

Donoho zeigt in einer Reihe von Artikeln für diesen Schätzer unter der Annahme, daß das Rauschen normalverteilt mit konstanter Varianz ist, mehrere Eigenschaften, von denen wir hier zwei näher vorstellen wollen:

(smooth) Mit hoher Wahrscheinlichkeit ist $\hat{f}_n^*$ mindestens so glatt wie f, wobei Glattheit durch eine breite Palette von Glattheitsmaßen gemessen werden kann.

(adapt) $\hat{f}_n^*$ erzielt fast den Minimax Mean Square Error (MSE) über eine große Anzahl von Glattheitsräumen, darunter viele Räume, wo traditionelle lineare Schätzer den MSE nicht erzielen.

Um die Aussage (smooth) zu präzisieren, nutzen wir aus, daß die Wavelet-Basis nicht nur eine Orthonormalbasis des $L^2(\mathbb{R})$ ist, sondern überdies eine unbedingte Basis (siehe Fußnote [3]) für eine breite Palette von Glattheitsräumen darstellt, nämlich für die sogenannten Besov-Räume B_{pq}^{σ} und Triebel-Räume F_{pq}^{σ} mit $0 < \sigma < m$, wobei $m+1$ die Ordnung des Wavelets bezeichne (siehe [23]). Wir wollen hier auf Definitionen oder Charakterisierungen dieser Räume verzichten und nur bemerken, daß auf jedem dieser Räume eine Norm definiert ist, die Glattheit mißt. Zum Beispiel definiert man die als Spezialfall aus den Besov-Räumen hervorgehenden Hölder-Räume $C^s(\mathbb{R})$ für $0 < s < 1$ (und ähnlich für $s > 1$) als den Banach-Raum der beschränkten, stetigen Funktionen, deren Stetigkeitsmodul $\omega(h)$ der Bedingung $\omega(h) \leq Ch^s$ für eine Konstante C

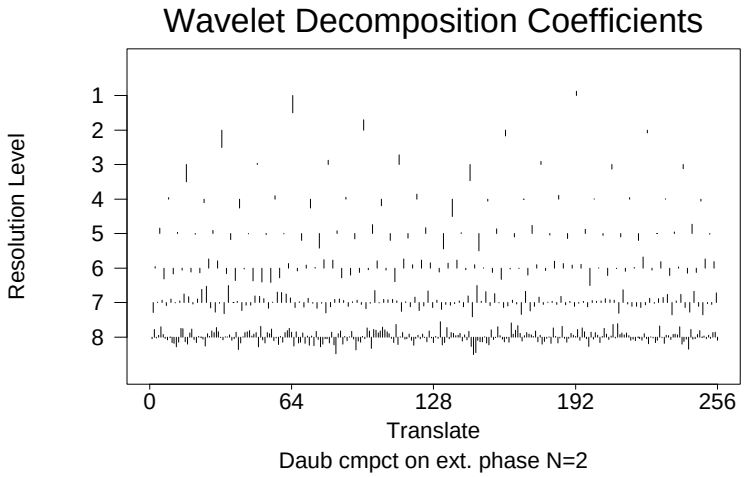

Abbildung 18: Die Wavelet-Koeffizienten bezüglich des Daubechies-Wavelets der Ordnung 1

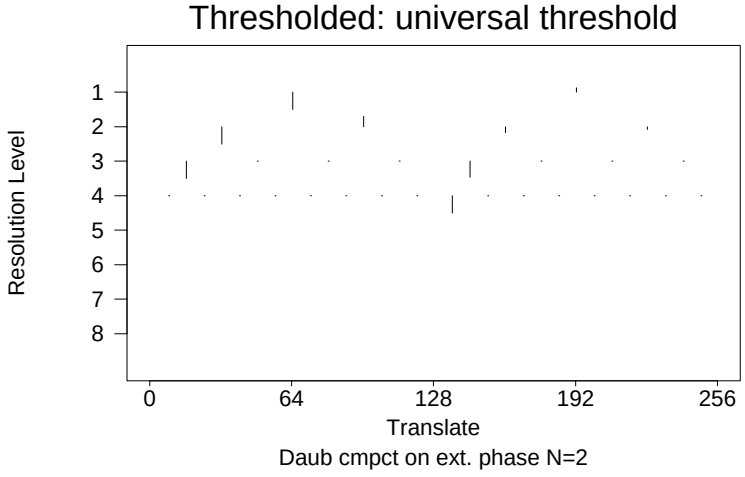

Abbildung 19: Die Wavelet-Koeffizienten nach Anwendung eines globalen Thresholds

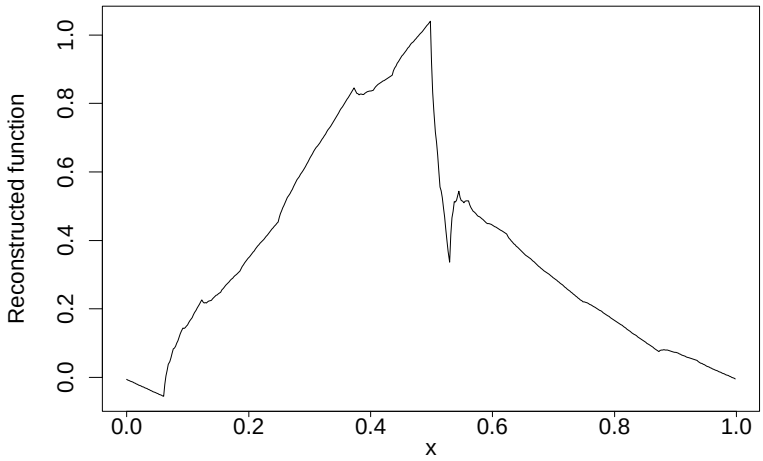

Abbildung 20: Rekonstruktion durch inverse Wavelet-Transformation

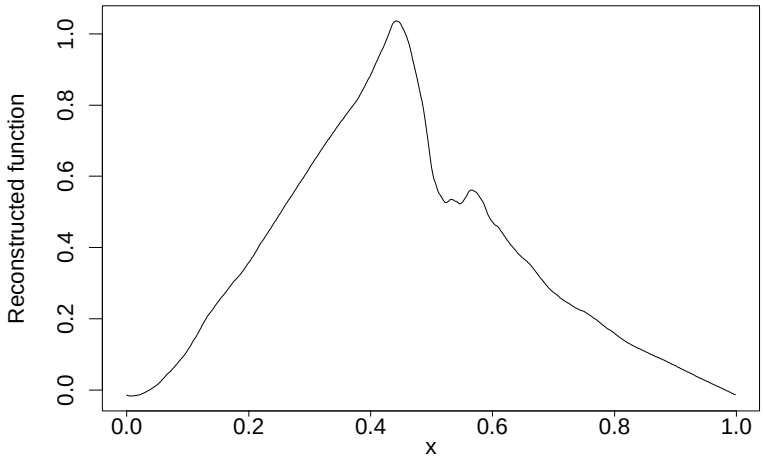

Abbildung 21: Ergebnis von Wavelet-Thresholding mit Daubechies-Wavelet der Ordnung 4

genügt.[15] Die Norm von $f \in C^s(\mathbb{R})$ ist dann $||f||_\infty + \sup_{0<h\leq 1} \omega(h)h^s$.

Wir fassen nun in $\mathcal{S}$ alle Räume $B^\sigma_{p,q}$ und alle Räume $F^\sigma_{p,q}$ mit $1/p < \sigma < m$ zusammen, die stetig in $C[0,1]$ eingebettet sind. Insbesondere stellt die Wavelet-Basis für jeden von ihnen eine unbedingte Basis dar ([17]).

Bevor wir eine Aussage über die Glattheit von $\hat{f}^*_n$ treffen können, müssen wir zunächst klären, was wir überhaupt unter $\hat{f}^*_n$ verstehen wollen. Der Algorithmus 2.1 liefert zunächst nur einen Vektor für die geschätzten Funktionswerte $(\hat{f}^*_n(t_i))^n_{i=1}$. Donoho stellt dazu die Hybrid-Transformation vor, die der üblichen Wavelet-Transformation sehr ähnlich ist. Insbesondere existiert für eine beliebige orthonormale Wavelet-Basis mit kompaktem Träger der Ordnung m und für $n = 2^J$ ein System von Funktionen $(\tilde{\phi}_k), (\tilde{\psi}_{jk})$ mit $0 \leq k < 2^j$ und $j > 0$ so, daß sich jede auf $[0,1]$ stetige Funktion f als

$$f = \sum_{k=-\infty}^{\infty} \tilde{\beta}_{j_0 k}\tilde{\phi}_{j_0 k} + \sum_{j=j_0}^{\infty}\sum_{k=-\infty}^{\infty} \tilde{\alpha}_{jk}\tilde{\psi}_{jk}$$

darstellen läßt und die ersten n Koeffizienten $\left(\tilde{\beta}_{j_0}, \tilde{\alpha}_{j_0}, \ldots, \tilde{\alpha}_{J-1}\right)$ mit den Koeffizienten der diskreten Wavelet-Transformation (angewandt auf die Samples $f(i/n)$) übereinstimmen. Die Basisfunktionen $\tilde{\psi}_{jk}$ entstehen allerdings nicht aus Verschiebungen und Streckungen endlich vieler Grundfunktionen, vielmehr gibt es für jeden Level j ein eigenes Mutterwavelet, das um ganzzahlige Werte verschoben wird.

Es sei im folgenden $\hat{f}^*_n$ die Funktion auf [0,1], deren diskrete Wavelet-Koeffizienten mit den Koeffizienten der Hybrid-Transformation übereinstimmen. Diese interpoliert dann insbesondere die durch den Algorithmus 2.1 geschätzten Funktionswerte. Donoho beweist in [14] folgenden Satz:

Satz 2.1. *Es gibt universelle Konstanten (π_n) mit $\pi_n \to 1$ für $n = 2^J \to \infty$ und Konstanten $C_1(\mathcal{F}, \psi)$, die vom Funktionenraum $\mathcal{F}[0,1] \in \mathcal{S}$ und von der Wavelet-Basis abhängen, aber nicht von n oder f so, daß*

$$\mathbb{P}\left(\left\{||\hat{f}_n||_{\mathcal{F}} \leq C_1(\mathcal{F}, \psi) \cdot ||f||_{\mathcal{F}} \forall \mathcal{F} \in \mathcal{S}\right\}\right) \geq \pi_n. \tag{2.4}$$

Donoho merkt an, daß Satz 2.1 eine präzise Formulierung dafür ist, zu sagen, daß die Rekonstruktion rauschfrei ist. Eine einfache, aber sehr anschauliche Folgerung ist die folgende: Falls f die Nullfunktion ist, dann ist mit einer Wahrscheinlichkeit von mindestens π_n auch $\hat{f}^*_n$ die Nullfunktion. Andere Verfahren wie Spline Smoothing, Kernschätzer oder lokale polynomiale Regression liefern Rekonstruktionen, die infolge des Rauschens in den Beobachtungen (natürlich nur leicht) oszillieren.

Wir befassen uns nun näher mit Aussage (adapt). Um die Güte der Approximation von $\hat{f}^*_n$ zu bestimmen, betrachten wir das Risiko

$$R(\hat{f}^*_n, f) = n^{-1}\sum_{i=1}^{n} \mathbb{E}(\hat{f}^*_n(i/n) - f(i/n))^2,$$

[15] Der Stetigkeitsmodul einer Funktion f ist definiert als

$$\omega(h) = \sup\{|f(x) - f(y) : |x - y| \leq h\}.$$

das natürlich möglichst klein sein sollte. Desweiteren bezeichnen wir für einen Funktionenraum $\mathcal{F} \in \mathcal{S}$ mit $\mathcal{F}_C$ die Kugel der Funktionen $\{f : ||f||_{\mathcal{F}} \leq C\}$. Das schlechteste Verhalten unseres Schätzers können wir mit

$$\sup_{\mathcal{F}_C} R(\hat{f}_n^*, f)$$

messen, und das kann für keinen meßbaren Schätzer besser sein als der Minimax-MSE

$$\inf_{\hat{f}} \sup_{\mathcal{F}_C} R(\hat{f}, f).$$

Donoho beweist (ebenfalls in [14]) folgenden Satz:

Satz 2.2. *Für jede Kugel $\mathcal{F}_C$, die von einem Funktionenraum $\mathcal{F} \in \mathcal{S}$ abgeleitet ist, existiert eine Konstante $C_2(\mathcal{F}_C, \psi)$, die nicht von n abhängt, so daß für alle $n = 2^J$*

$$\sup_{f \in \mathcal{F}_{C\,C}} R(\hat{f}_n^*, f) \leq C_2(\mathcal{F}_C, \psi) \inf_{\hat{f}} \sup_{f \in \mathcal{F}_{C\,C}} R(\hat{f}, f).$$

Donoho merkt an, daß derzeit kein anderes Verfahren bekannt ist, daß in vergleichbarer Weise in so vielen Glattheitsräumen annähernd den Minimax-MSE erzielt. Im allgemeinen erzielen Nicht-Wavelet-Methoden, die in einer Weise konstruiert werden, durch die der Minimax-MSE minimiert wird, ihre Erfolge nur in einer kleinen Auswahl der Kugeln $\mathcal{F}_C$, vornehmlich der L^2-Sobolev-Kugeln, und die Mittel, die eingesetzt werden, um dieses Ziel zu erreichen, sind kompliziert. Im Gegensatz dazu läßt sich der Wavelet-Schätzer $\hat{f}_n^*$ sehr leicht konstruieren und analysieren, und ist bis auf einen logarithmischen Faktor optimal für jede Kugel $\mathcal{F}_C$ aus $\mathcal{S}$.

In [17] zeigt Donoho, daß VisuShrink noch bzgl. einiger anderen Kriterien fast optimal ist, darunter das Schätzen einer Funktion in einem festen Punkt t_0.

Zum Schluß wollen wir noch anhand zweier Beispiele zeigen, welche Ergebnisse Wavelet-Shrinkage bei der Anwendung auf Datensätze liefert. Zunächst betrachten wir den „Doppler"-Datensatz, der im vorangegangenen Abschnitt bei klassischen Verfahren für große Probleme sorgte. Dazu benutzen wir Hard-Thresholding, wobei der Threshold mit VisuShrink berechnet und die Daubechies-Wavelet-Basis der Ordnung $m = 4$ mit $j_0 = 3$ benutzt wird. Das Ergebnis zeigt Abbildung 22 und ist sehr zufriedenstellend.

Schließlich betrachten wir noch Spektroskopie-Daten. Hier besteht das Problem darin, daß die resultierenden Kurven scharfe Peaks aufweisen, die natürlich nicht weggeglättet werden sollen. Wie Prof. Williams vom Department of Chemistry der Clemson University berichtete, werden in der Spektroskopie im Augenblick hauptsächlich „Moving Window Type"-Verfahren benutzt – die im letzten Abschnitt beschriebenen Probleme sind auch ihm sehr gut bekannt.

Abbildung 23 zeigt einen Datensatz aus der NMR-Spektroskopie, der in dem Wavelet-Softwarepaket „WaveLab", das die Stanford University entwickelt hat, enthalten ist und von Prof. Adrian Maudsley (VA Medical Center, San Francisco) stammt. Spline-Smoothing liefert den Peak nur auf Kosten eines hohen Restrauschens (Abbildung 24). Im Gegensatz dazu beseitigt Wavelet Shrinkage mit einem Wavelet der Ordnung 4 das Rauschen und behält den Peak, wie in Abbildung 2.2 zu sehen ist.

2.3 Wavelet-Shrinkage und Robustheit

Wir haben im vorangegangenen Abschnitt Wavelet-Shrinkage als einen Lösungsansatz in der nicht-parametrischen Regression kennengelernt. Es zeigte sich, daß Wavelet Shrinkage eine Reihe von Vorteilen gegenüber klassischen Verfahren aufweist:

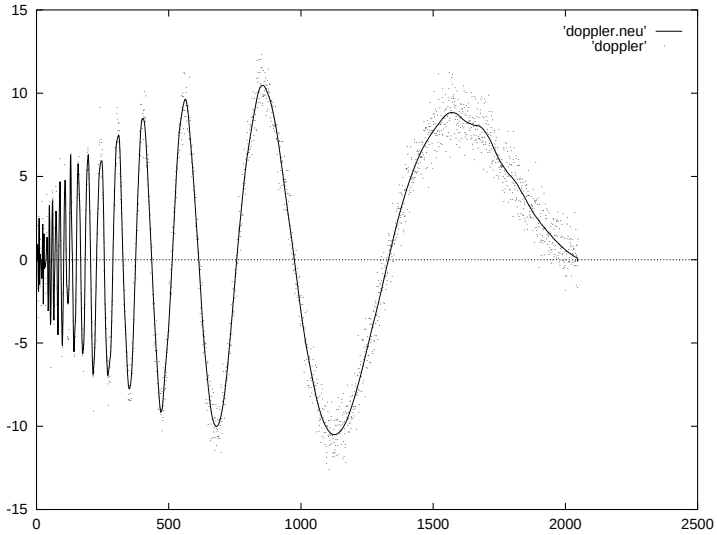

Abbildung 22: Hard-Thresholding angewandt auf den Datensatz „Doppler” mit der Daubechies-Wavelet-Basis der Ordnung $m = 4$ und $j_0 = 3$

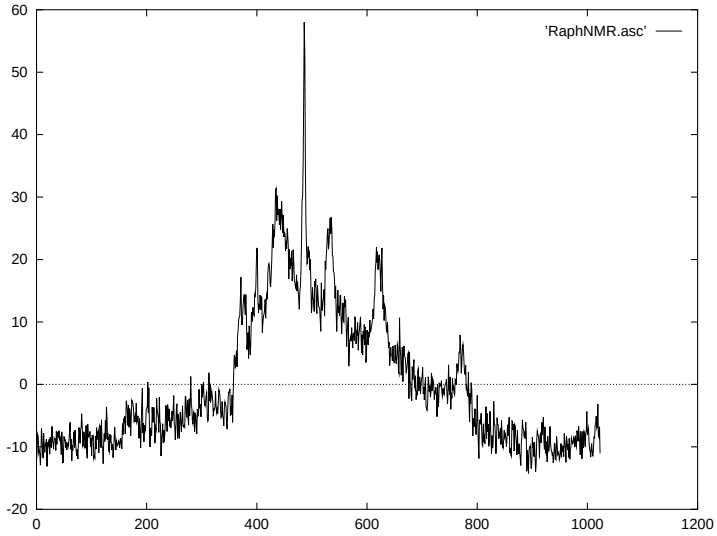

Abbildung 23: Ein Datensatz aus der NMR-Spektroskopie

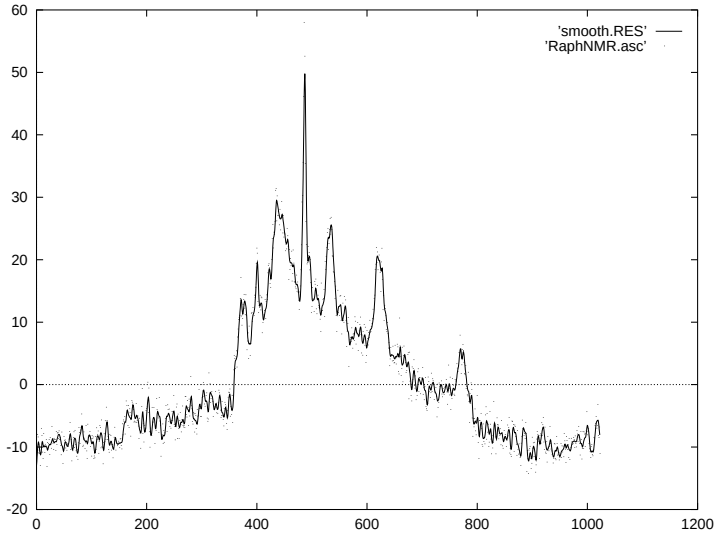

Abbildung 24: Spline Smoothing angewandt auf den Spektroskopiedatensatz mit $\lambda = 0.4$

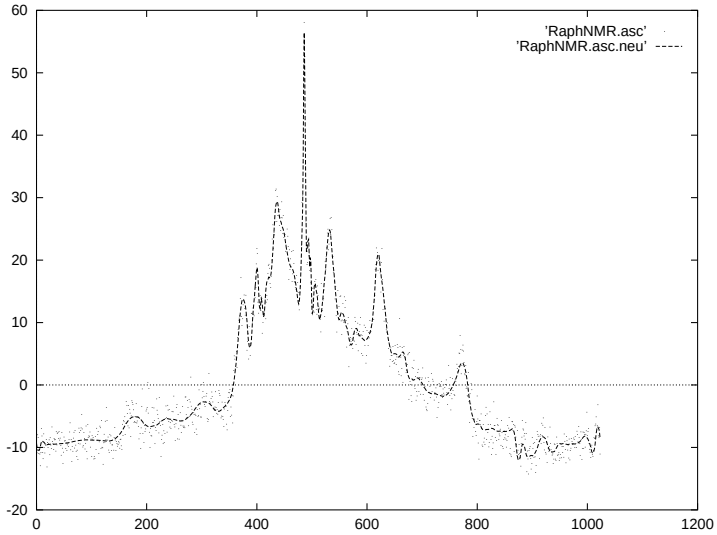

Abbildung 25: Hard-Thresholding angewandt auf den Datensatz aus der Spektroskopie mit der Daubechies-Wavelet-Basis der Ordnung $m = 4$ und $j_0 = 4$

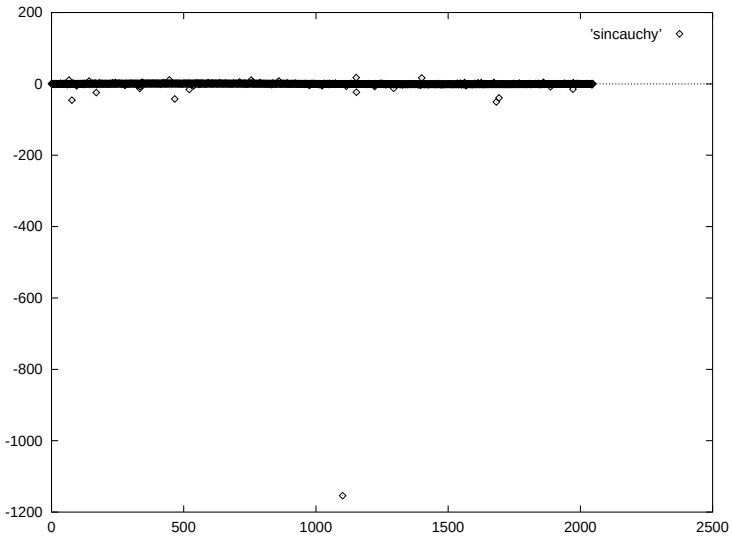

Abbildung 26: Eine Cauchy-verrauschte Sinus-Kurve

- Wavelet-Shrinkage benutzt weder einen globalen Smoothing-Parameter, noch eine globale Fenstergröße. Das Rauschen wird somit geglättet, Unstetigkeiten bleiben erhalten.
- Wavelet-Shrinkage ist schnell, nämlich ein $O(n)$-Verfahren.

Desweiteren läßt sich Wavelet-Shrinkage ohne Probleme auf mehrdimensionale Problemstellungen wie Bildverarbeitung übertragen (siehe zum Beispiel [26]).

Grundannahme war jedoch bislang, daß das Rauschen normalverteilt ist. Donoho merkt in [13] an, daß lineare, orthogonale Wavelet-Analysis in natürlicher Weise zu dieser Annahme führt. Wir wollen nun überlegen, wie Wavelet Shrinkage reagiert, falls diese Annahme nicht erfüllt ist und stören dazu eine Sinus-Kurve mit 2048 Samples einer $\mathfrak{C}(0, 0.1)$-Cauchy-Verteilung. Betragsmäßig waren mehr als 80 Prozent des Rauschens kleiner als 0.3 und 132 Samples größer als 1 (siehe Abbildung 26), was ungefähr den theoretisch errechneten Quantilen einer $\mathfrak{C}(0, 0.01)$-Verteilung entspricht ($F(0.9) \doteq 0.308$ und $F(1 - 66/2048) \doteq 0.984$). Das Rauschen ist stellenweise so groß, daß es die Plot-Skala dominiert und von der ursprünglichen Sinus-Kurve nichts zu erkennen ist. Es ist klar, wie Wavelet Shrinkage reagiert: Die Ausreißer lassen die Koeffizienten in der Nähe der Ausreißer (und zwar insbesondere die „Fine Scale"-Koeffizienten) groß werden, so daß nichts auf 0 gesetzt wird und die großen Ausreißer wieder zurückgeliefert werden (Abbildung 27). Wir versuchen nun „unser Möglichstes" und setzen zudem **alle** Koeffizienten der vier feinsten Skalen (von 11) auf 0. Wiederum ist die resultierende Kurve stark durch Ausreißer beeinflußt und läßt sich die Sinus-Kurve nur schwach erkennen (Abbildung 28).

Ein zweites Beispiel liefern uns die Motorrad-Daten, die wir zu Beginn dieses Kapitels betrachtet haben. Dieser Datensatz ist durch mehrere Ausreißer kontaminiert und weist außerdem eine variable Varianz auf. Abbildung 29 zeigt das Ergebnis von Hard-Thresholding mit VisuShrink und dem Daubechies-Wavelet der Ordnung 4 und $j_0 = 2$. [16] Das Ergebnis ist sehr stark durch die Ausreißer beeinflußt und damit unbrauchbar.

[16] Vor Anwendung von Wavelet-Thresholding wurden die letzten vier der ursprünglich 132 Datenpunkte gestrichen, um eine Zweierpotenz zu erhalten.

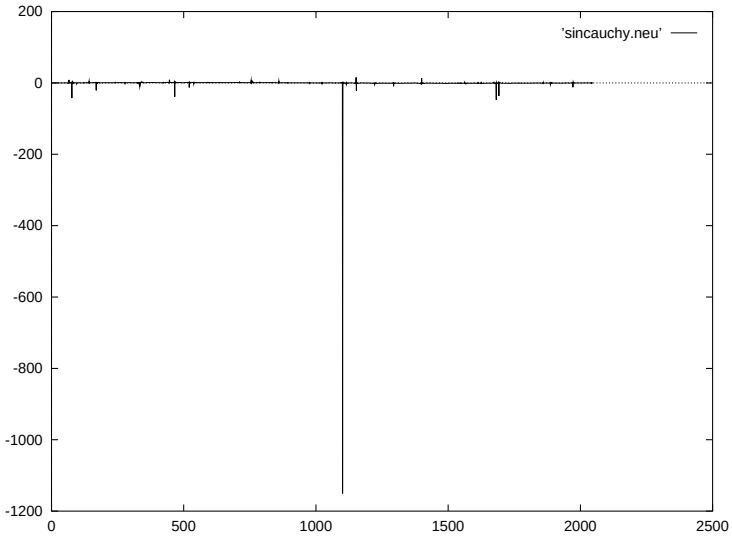

Abbildung 27: Die Sinus-Kurve nach Thresholding

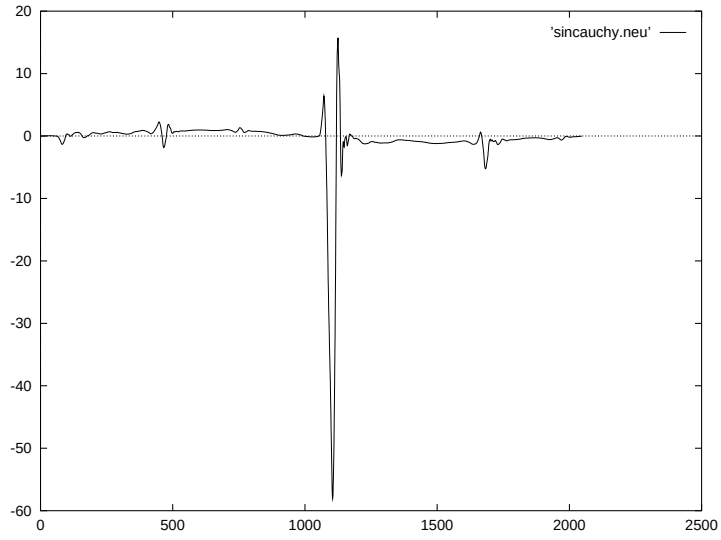

Abbildung 28: Die Sinus-Kurve, nachdem alle Koeffizienten der vier feinsten Skalen auf 0 gesetzt worden sind

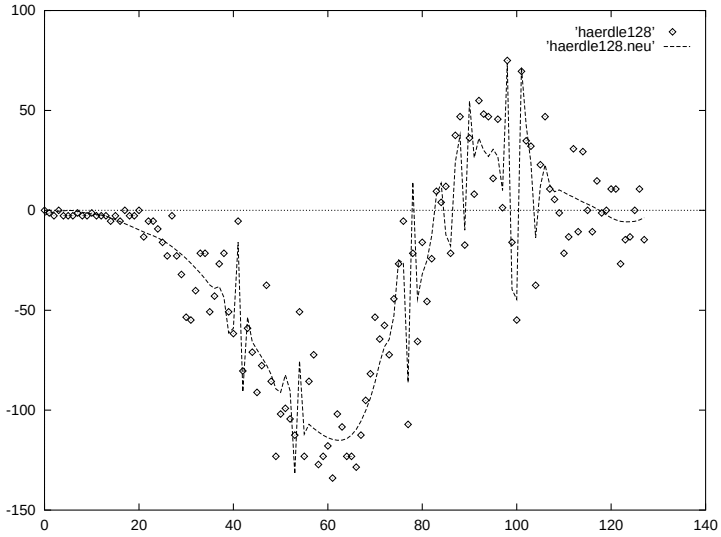

Abbildung 29: Hard-Thresholding angewandt auf die Motorrad-Crash-Daten mit der Daubechies-Wavelet-Basis der Ordnung $m = 4$ und $j_0 = 3$

Die beiden folgenden Lemmata sollen erklären, warum die Thresholding-Schätzer in dieser Art und Weise reagieren. Das erste erläutert den Effekt eines einzelnen Ausreißers auf den Thresholding-Schätzer. Konvergiert ein Datenpunkt gegen ∞, so kovergiert an der entsprechenden Stelle der Soft- oder Hard-Thresholding-Schätzer ebenfalls gegen ∞, wobei aber der Abstand zwischen Datenpunkt und Schätzfunktion konstant bleibt.

Lemma 2.1. *Es seien* $\mathrm{Soft} : \mathbb{R}^n \to \mathbb{R}^n$ *und* $\mathrm{Hard} : \mathbb{R}^n \to \mathbb{R}^n$ *Soft- und Hard-Thresholding-Verfahren, die beide auf einer Daubechies-Wavelet-Basis mit Ordnung kleiner als* $2^{J-2} - 1$ *operieren. Weiterhin seien* $y \in \mathbb{R}^n$, $i \in \{1, \ldots, n\}$ *und* $f : \mathbb{R} \to \mathbb{R}^n$ *mit*

$$(f(\lambda))_j = \begin{cases} \lambda, & i = j, \\ y_i & \textit{sonst.} \end{cases}$$

Dann gibt es $\lambda_1 < \lambda_2$ *und* $C_1, C_2 \in \mathbb{R}$*, so daß für alle* $\lambda < \lambda_1$

$$(f(\lambda) - \mathrm{Soft}(f(\lambda)))_i = C_1$$

bzw.

$$(f(\lambda) - \mathrm{Hard}(f(\lambda)))_i = 0$$

und für alle $\lambda > \lambda_2$

$$(f(\lambda) - \mathrm{Soft}(f(\lambda)))_i = C_2$$

bzw.

$$(f(\lambda) - \mathrm{Hard}(f(\lambda)))_i = 0$$

gilt.

Beweis. Wir überprüfen zunächst, wieviele Detailkoeffizienten im feinsten Level, von der Änderung *eines* Datenpunktes betroffen sind. Wenn man sich noch einmal den Algorithmus der

diskreten Wavelet-Transformation anschaut, sieht man, daß diese Zahl halb so groß ist, wie Filterkoeffizienten von Null verschieden sind. Da die Ordnung der zugrundeliegenden Wavelet-Basis nach Voraussetzung kleiner als $2^{J-2}-1$ ist und der Filter zu einem Daubechies-Wavelet der Ordnung k aus $2\cdot(k+1)$ von Null verschiedenen Filterkoeffizienten besteht, beeinflußt also die Änderung eines Datenpunktes im feinsten Level höchstens $2^{J-2}-1$ der insgesamt 2^{J-1} Detailkoeffizienten, also weniger als die Hälfte.

Es sei nun $(\lambda_n)_{n\in\mathbb{N}}$ eine Folge, die streng monoton gegen $-\infty$ konvergiert. Weil $\mathcal{W}$ eine lineare Abbildung ist, konvergieren dann auch alle Einträge von $\mathcal{W}f(\lambda_n)$, auf die der i-te Datenpunkt Einfluß hat, streng monoton gegen ∞ oder $-\infty$.

Insbesondere gibt es $\gamma_1\in\mathbb{R}$, so daß für alle $\lambda<\gamma_1$ die geschätzte Varianz von $f(\lambda)$ konstant bleibt, die wie oben beschrieben als MAD der Detailkoeffizienten des feinsten Levels berechnet wird. Da der Threshold t proportional von der Varianz abhängt, ist auch er konstant für alle $\lambda<\gamma_1$.

Weiterhin gibt es aber auch ein $\lambda_1\le\gamma_1$, so daß alle Waveletkoeffizienten von $\mathcal{W}f(\lambda_1)$, die von dem i-ten Datenpunkt beeinflußt werden und die aus dem j_0-ten oder einem höheren Level stammen, betragsmäßig größer oder gleich t sind.

Wendet man nun Hard-Thresholding auf $f(\lambda)$ für ein $\lambda<\lambda_1$ an, bleiben somit alle vom i-ten Datenpunkt beeinflußten Koeffizienten unverändert. Die Folge ist, daß $(f(\lambda)-\mathrm{Hard}(f(\lambda)))_i=0$. Wendet man andererseits Soft-Thresholding an, so werden alle diese Koeffizienten im Betrag um t dezimiert. Da dies unabhängig von λ geschieht, ist die Differenz dieser Koeffizienten vor und nach Abschneiden für $\lambda<\lambda_1$ konstant, es gibt also ein $w\in\mathbb{R}^n$ so, daß für alle $\lambda<\lambda_1$

$$w=\mathcal{W}f(\lambda)-T^{\mathrm{Soft}}\left(\mathcal{W}f(\lambda)\right)$$

gilt, wobei die Einträge, die vom i-ten Datenpunkt beeinflußt werden und größer als j_0 sind, alle gleich t oder $-t$ sind, und der Rest 0 ist. Also ist wegen der Linearität der DWT auch die Differenz $f(\lambda)-\mathrm{Soft}(f(\lambda))$ und insbesondere das Residuum $(f(\lambda)-\mathrm{Soft}(f(\lambda)))_i$ an der Stelle i konstant. Die Existenz von λ_2 folgt analog und damit die Behauptung. □

Das zweite Lemma zeigt, daß der maximale Abstand zwischen Daten und Threshold-Schätzer nach oben durch eine Konstante beschränkt ist, die proportional vom Threshold abhängt (und damit im Falle von *VisuShrink* auch von der Varianz).

Lemma 2.2. *Es sei* $\mathrm{Thresh}:\mathbb{R}^n\to\mathbb{R}^n$ *ein Thresholding-Verfahren und* $t^{\mathrm{Thresh}}:\mathbb{R}^n\to\mathbb{R}$ *die Abbildung, die einem Datenvektor* y *den Threshold zuordnet, der bei Anwendung von* Thresh *auf* y *benutzt wird. Dann gilt für alle* $y\in\mathbb{R}^n$*:*

$$||y-\mathrm{Thresh}(y)||_\infty\le\sqrt{n}\cdot t^{\mathrm{Thresh}}(y)$$

Beweis. Aus der Parzeval-Identität und wegen

$$\mathrm{Thresh}(y)=\mathcal{W}^T\left(T^{\mathrm{Thresh}}\left(\mathcal{W}y\right)\right)$$

folgt, daß

$$||y-\mathrm{Thresh}(y)||_2=||\mathcal{W}y-T^{\mathrm{Thresh}}(\mathcal{W}y)||_2.$$

Weil außerdem für beliebige $a\in\mathbb{R}^n$

$$||a||_\infty\le||a||_2\le\sqrt{n}\cdot||a||_\infty$$

gilt, folgt zusammen mit der Definition der Thresholding-Regeln die Behauptung. □

Man kann sich leicht Fälle in der Praxis vorstellen, bei denen ein Verhalten (wie in den Beispielen illustriert) unerwünscht ist. Man denke zum Beispiel an Signalübertragung über schlechte Telefonleitungen, wo hin und wieder ein Knacken zu hören ist. Wir wollen daher im folgenden andere Verfahren zur nicht-parametrischen Regression, die robust gegenüber Ausreißern reagieren, entwickeln und untersuchen.

Wenn man von Robustheit bei nicht-parametrischer Regression spricht, stößt man jedoch zwangsläufig auf ein Problem, das ein kleiner Vergleich gut demonstiert. Wir haben im letzten Abschnitt ein Beispiel aus der Spektroskopie gesehen, bei dem Wavelet Shrinkage in ähnlicher Weise reagierte wie bei unserer Cauchy-verrauschten Sinus-Kurve – mit dem Unterschied, daß dort die starken Sprünge nicht vom Rauschen, sondern vom Signal stammten und somit erwünscht waren. Wir werden im nächsten Kapitel auf einen anderen Datensatz stoßen, in denen 50 und mehr Ausreißer aufeinander folgen. Es ist nicht möglich, daß ein und derselbe Algorithmus in beiden Situationen funktionieren kann. Wir müssen daher immer im Auge behalten, welche Formen von Ausreißern ein Algorithmus erkennt, und möglichst dem Anwender solcher Verfahren zur robusten nicht-parametrischen Regression die Möglichkeit bieten, selbst anzugeben, wieviele Ausreißer maximal aufeinander folgen können.

3 Robuste nicht-parametrische Regression mit Wavelets

Wir haben im zweiten Teil dieser Arbeit gesehen, daß Wavelet Shrinkage gegenüber klassischen Verfahren der nicht-parametrischen Regression einige Vorteile aufweist. Probleme ergeben sich jedoch bei Datensätzen, deren Beobachtungen durch Ausreißer gestört sind. Wir wollen daher in diesem Abschnitt versuchen, andere Verfahren der nicht-parametrischen Regression zu entwickeln, die Wavelets und ihre Vorteile benutzen, aber zugleich gegenüber Ausreißern robust sind.

Wenn wir uns noch einmal Algorithmus (2.1) anschauen, ergeben sich prinzipiell vier Ideen für neue robuste Verfahren, die auf Wavelets basieren und die sich zum Teil kombinieren lassen.

1. Verfahren, die immer noch auf Wavelet Shrinkage beruhen, allerdings in folgender Art und Weise: Auf die Originaldaten wird zunächst konventionelles Wavelet Thresholding angewendet, dann aus der geglätteten Kurve und den Originaldaten neue Ausgangsdaten $\tilde{y}$ berechnet, auf die wiederum Wavelet Shrinkage angewendet wird, und so weiter, bis ein Abbruchkriterium erfüllt ist.

2. Verfahren, bei denen die Ausreißer mit Hilfe der Wavelet-Koeffizienten identifiziert und eliminiert werden und die Schätzfunktion durch Anwendung von Thresholding auf die geänderten Daten berechnet wird.

3. Verfahren, bei denen Ausreißer ohne Wavelets identifiziert und neue Ausgangsdaten generiert werden, auf die dann Wavelet Shrinkage angewandt wird.

4. Verfahren, bei denen die Wavelet-Koeffizienten der Schätzfunktion nicht durch die diskrete Wavelet-Transformation, sondern durch robuste Alternativmethoden berechnet werden.

Wir werden die neuen Algorithmen in dieser Reihenfolge vorstellen und analysieren.

3.1 Ausreißererkennung mit Wavelet Shrinkage

In diesem Abschnitt sollen Verfahren untersucht werden, die weiterhin Thresholding verwenden. Die Hoffnung besteht darin, daß klassisches Wavelet Shrinkage zwar auf Ausreißer reagiert, daß die Residuen aber dort, wo der Datensatz durch Ausreißer kontaminiert ist, verhältnismäßig groß sind. Die Algorithmen arbeiten nach dem folgenden Schema:

Algorithmus 3.1.

(1) Wende auf die Originaldaten $y = (y_1, \ldots, y_n)$ Wavelet Shrinkage an und erhalte eine Näherung $\hat{f}^*$.

(2) Falls ein Abbruchkriterium erfüllt ist, fertig.

(3) Generiere neue Ausgangsdaten $\tilde{y}$.

(4) Weiter mit Schritt 1, verwende allerdings die neuen Daten $\tilde{y}$ statt der Originaldaten y.

Dieses Schema läßt natürlich viel Spielraum für Variationen zu. In Schritt 1 muß man sich bereits entscheiden, wie man Wavelet Shrinkage durchführt (Hard/Soft-Thresholding, Wahl des Thresholds, etc.). Im zweiten Schritt kommen sowohl Abbruchkriterien aufgrund einer eingetretenen Konvergenz in Frage wie residuenbasierte Kriterien, bei denen zum Beispiel abgebrochen wird, sobald die maximale Runlänge der Vorzeichen der Residuen einen vorher bestimmten theoretischen Wert unterschreitet. Kritisch ist freilich der dritte Schritt, in dem aufgrund eines Vergleiches von Thresholding-Schätzer und Originaldaten (bzw. den aktualisierten Originaldaten) Ausreißer gefunden und beseitigt (oder doch zumindestens gemildert) werden sollen.

Folgenden einfachen Algorithmus wollen wir näher betrachten:

Algorithmus 3.2.

(1) Setze $f_0 := y = (y_1, \ldots, y_n)$ *und* $k := 0$. $\mathrm{Soft} : \mathbb{R}^n \to \mathbb{R}^n$ *sei das Soft-Thresholding-Verfahren VisuShrink. Schätze die Standardabweichung* $\hat{\sigma}$ *der Daten über den MAD der Fine Scale-Koeffizienten einer Wavelet-Transfomration wie in (*2.3*).*

(2) Wende auf f_k *Soft-Thresholding an und erhalte eine Näherung* $\hat{f}_k^* := \mathrm{Soft}(f_k)$. *Benutze bei der Berechnung des Thresholds* t *die im ersten Schritt bestimmte Standardabweichung der Originaldaten* y.

(3) Sei

$$r_k := ||f_k - f_k^*||_\infty$$

und

$$i_k = \min\{i \in \{1, \ldots, n\} | (f_k)_{i_k} - (f_k^*)_{i_k}| = r_k\}.$$

(4) Falls $r_k < \delta$, *fertig.*

(5) Definiere neue Ausgangsdaten f_{k+1} *durch*

$$(f_{k+1})_i = \begin{cases} (f_k^*)_i, & \textit{falls } i = i_k, \\ (f_k)_i & \textit{sonst.} \end{cases}$$

(6) Setze $k := k + 1$ *und setze bei Schritt 2 fort.*

Wir wenden nun diesen Algorithmus mit dem Daubechies-Wavelet der Ordnung 6 und $\delta = 0.025$ auf die Motorrad-Daten, die mit Cauchy-Rauschen versehene Sinus-Kurve und den Datensatz *NoisyBlocks* [17] an. j_0 wurde jeweils 3, 4 und 7 gewählt, 297, 2119 und 3299 Iterationen waren notwendig. Die Ergebnisse zeigen die Abbildungen 30, 31, und 32.

Wir untersuchen als erstes die Stetigkeit des Verfahrens.

Satz 3.1. *Wenn wir mit* $T : \mathbb{R}^n \to \mathbb{R}^n$ *die Abbildung bezeichnen, die auf einen Datenvektor* $y \in \mathbb{R}^n$ *die Schritte 2,3 und 5 des Algorithmus anwendet, so ist* T *fast überall stetig, aber nicht Lipschitz-stetig.*

[17] Für den Datensatz *NoisyBlocks* wurde zu 2048 Funktionswerten einer stückweise konstanten Funktion normalveteiltes Rauschen mit Standardabweichung $\sigma = 0.8$ hinzuaddiert.

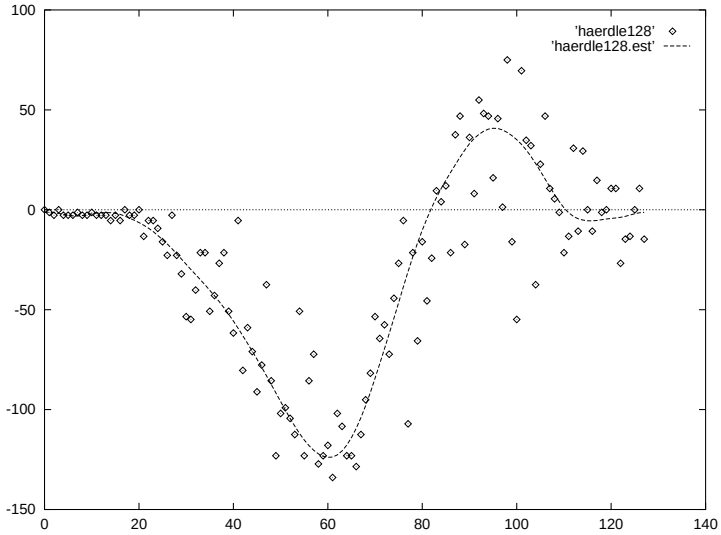

Abbildung 30: Algorithmus (3.2) angewandt auf die Motorrad-Daten

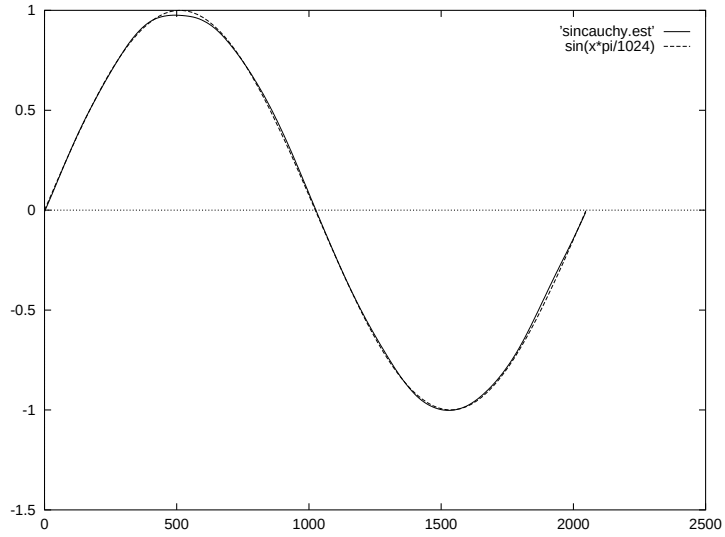

Abbildung 31: Algorithmus (3.2) angewandt auf die mit Cauchy-Rauschen gestörte Sinus-Kurve. Zum Vergleich eine echte Sinus-Kurve.

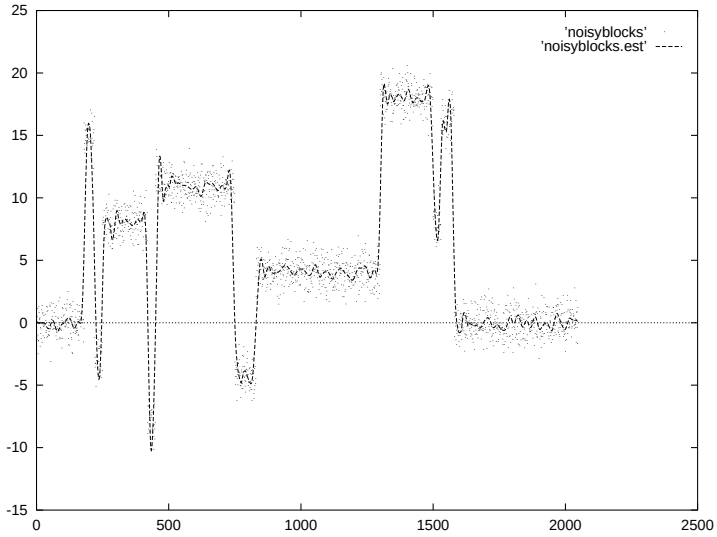

Abbildung 32: Algorithmus (3.2) angewandt auf den Datensatz *NoisyBlocks*.

Beweis. Wir klären zunächst die Frage der Lipschitz-Stetigkeit. Bei allen hier betrachteten Beispielen ist die Folge der $(r_k)_{k\in\mathbb{N}}$ nicht streng monoton fallend. Es gibt also jeweils ein $k \in \mathbb{N}$ mit $r_{k+1} > r_k$. Somit gilt für $x_0 := T^{k+1}(y)$ und $x_1 := T^k(y)$

$$||T(x_0) - T(x_1)||_\infty = ||f_{k+2} - f_{k+1}||_\infty = r_{k+1} > r_k = ||x_0 - x_1||.$$

Selbst durch Einschränkung des Definitionsbereiches auf

$$D_T = \{x \in \mathbb{R}^n | \exists k \in \mathbb{N}_0 : x = T^k(y)\}$$

wird T also nicht Lipschitz-stetig.

Damit kommen wir zur Stetigkeit. Es ist klar, daß $\mathcal{W}$, $\mathcal{W}^T$ sowie T^{Soft} stetig sind, also sind es auch die Abbildungen Soft und

$$g : \mathbb{R}^n \to \mathbb{R}^n, \quad g(x) = x - \mathrm{Soft}(x).$$

T ist damit bereits stetig in allen Punkten $x \in \mathbb{R}^n$, in denen das maximales Residuum eindeutig bestimmt ist, für die also genau ein $i \in \{1, \ldots, n\}$ existiert mit $|(g(x))_i| = r_k$, denn aufgrund der Stetigkeit von g gibt es eine Umgebung U von x, so daß für alle $\tilde{x} \in U$ gilt, daß

$$i = \min\{j \in \{1, \ldots, n\} | (g(\tilde{x}))_j| = |g(\tilde{x})|_\infty\} \tag{3.1}$$

und somit ändert T auf U stets die gleiche Koordinate ab.

T ist außerdem stetig in allen Punkten $x \in \mathbb{R}^n$, in denen zwar keine Eindeutigkeit des maximalen Residuums vorliegt, für die aber trotzdem eine Umgebung U und ein $i \in \{1, \ldots, n\}$ existieren, so daß (3.1) für alle $\tilde{x} \in U$ gilt. [18] Wir definieren daher

$$\begin{aligned} D := \{x \in \mathbb{R}^n : \exists i, j \in \{1, \ldots, n\}, i < j : |g(x)_i| = |g(x)_j| = |g(x)|_\infty \\ \wedge \exists (y_n) \subset \mathbb{R}^n : y_n \to x, |g(y_n)_i| < |g(y_n)_j|\} \end{aligned} \tag{3.2}$$

[18] Es läßt sich (vielleicht überraschenderweise) ziemlich einfach zeigen, daß diese Punkte *keine* Nullmenge bilden.

und zeigen, daß D eine Nullmenge bzgl. des Lebesgue-Maßes ist.

Zunächst vereinbaren wir, daß J die Indexmenge der Waveletkoeffizienten der Level j_0 und größer ist. Außerdem seien für $i, j \in \mathbb{N}$ mit $i < j$ sowie für $I \subset J$

$$D_{i,j}^I := \{x \in \mathbb{R}^n : |g(x)_i| = |g(x)_j| \wedge \forall k \in I : |(\mathcal{W}x)_k| > t \wedge \forall k \in J \setminus I : |(\mathcal{W}x)_k| < t\} \tag{3.3}$$

und für $i \in \{1, \ldots, n\}$

$$C_i := \{x \in \mathbb{R}^n : |(\mathcal{W}x)_i| = t\}.$$

Es ist klar, daß

$$D \subset \left(\bigcup_{\substack{I \subset J \\ i,j \in \{1,\ldots,n\} \\ i<j}} D_{i,j}^I \right) \cup \left(\bigcup_{i \in \{1,\ldots,n\}} C_i \right).$$

Es gilt aber sogar

$$D \subset \left(\bigcup_{\substack{I \subsetneq J \\ i,j \in \{1,\ldots,n\} \\ i<j}} D_{i,j}^I \right) \cup \left(\bigcup_{i \in \{1,\ldots,n\}} C_i \right), \tag{3.4}$$

denn falls $x \in D_{i,j}^J$ für $i, j \in \{1, \ldots, n\}$ mit $i < j$, so ist $|(\mathcal{W}x)_k| > t$ für alle $k \in J$, und wegen der Stetigkeit von g gibt es eine Umgebung U von x, so daß für alle $\tilde{x} \in U$ und für alle $k \in J$ wieder $|(\mathcal{W}\tilde{x})_k| > t$ gilt. Das bedeutet aber, daß

$$\begin{aligned} g(\tilde{x}) &= \tilde{x} - \mathcal{W}^T(T^{\text{Soft}}(\mathcal{W}\tilde{x})) \\ &= \mathcal{W}^T(\mathcal{W}\tilde{x} - T^{\text{Soft}}(\mathcal{W}\tilde{x})) \\ &= \mathcal{W}^T c \end{aligned} \tag{3.5}$$

auf U konstant ist, wobei $c_i = t \cdot \text{sign}(x_i)$ ist. Damit ist aber $x \notin D$ und die Darstellung (3.4) folgt.

Da die Vereinigungen endlich sind, müssen wir lediglich zeigen, daß die daran teilnehmenden Mengen Nullmengen sind. Wir beginnen mit den C_i. Für ein $i \in \{1, \ldots, n\}$ sei

$$C_i^1 := \{x \in \mathbb{R}^n : (\mathcal{W}x)_i = t\}.$$

Da $\mathcal{W}(C_i^1) = \{y \in \mathbb{R}^n : y_i = t\}$ eine Hyperebene des $\mathbb{R}^n$ und $\mathcal{W}$ ein Automorphismus ist, ist C_i^1 eine Nullmenge bezüglich des Lebesgue-Maßes. Genauso ist auch

$$C_i^2 := \{x \in \mathbb{R}^n : (\mathcal{W}x)_i = t\}$$

und insgesamt C_i eine Nullmenge.

Es seien nun $I \subsetneq J$ und $i, j \in \{1, \ldots, n\}$ mit $i < j$ beliebig, aber fest. Wir betrachten

$$\tilde{D}_{i,j}^I := \{x \in \mathbb{R}^n : g(x)_i = g(x)_j \wedge \forall k \in I : (\mathcal{W}x)_k| > t \wedge \forall k \in J \setminus I : |(\mathcal{W}x)_k| < t\} \tag{3.6}$$

und zeigen, daß $\tilde{D}^I_{i,j}$ eine Nullmenge ist. Der Beweis wird zeigen, daß jede andere Auflösung der Betragsstriche ebenfalls ganz analog zu Nullmengen führt. [19]

Da $\mathcal{W}$ ein Automorphismus ist, ist $\tilde{D}^I_{i,j}$ genau dann eine Nullmenge, wenn

$$\tilde{D} := \mathcal{W}(\tilde{D}^I_{i,j})$$

eine Nullmenge ist. Wir betrachten daher die Abbildung

$$h : \mathbb{R}^n \to \mathbb{R}^n, \quad h(y) = y - T^{\mathrm{Soft}}(y).$$

Es gilt

$$\tilde{D} = \{y \in \mathbb{R}^n : (\mathcal{W}^T h(y))_i = (\mathcal{W}^T h(y))_j \wedge \forall k \in I : y_k > t \wedge \forall k \in J \setminus I : |y_k| < t\}$$

Wir untersuchen nun das Bild von $\tilde{D}$ unter der Abbildung h. Es gilt einerseits für alle $y \in \tilde{D}$, daß

$$(h(y))_k = \begin{cases} y_k & k \in J \setminus I \\ t & k \in I \end{cases} \tag{3.7}$$

und somit insbesondere

$$h(\tilde{D}) \subset F := \{z \in \mathbb{R}^n : z_k = t \text{ für alle } k \in I\},$$

wobei F ein affiner Unterraum des $\mathbb{R}^n$ mit $\dim F = n - \#I$ ist.

Andererseits ist aber $V := \{x \in \mathbb{R}^n : x_i = x_j\}$ eine Hyperebene des $\mathbb{R}^n$, also auch $\tilde{C} := \mathcal{W}(V)$, und es gilt $h(\tilde{D}) \subset \tilde{V}$.

Insgesamt erhält man, daß $h(\tilde{D}) \subset \tilde{V} \cap F$, wobei $\tilde{V} \cap F$ ein affiner Unterraum des $\mathbb{R}^n$ mit $\dim \tilde{V} \cap F \leq n - (1 + \#I)$ ist. Für $\tilde{D}$ selbst gilt dann $\tilde{D} \subset h^{-1}(\tilde{V} \cap F)$. Falls $\tilde{V} \cap F = \emptyset$ ist, sind wir bereits fertig, denn $\tilde{D}$ ist dann insbesondere eine Nullmenge.

Sei also $\tilde{V} \cap F \neq \emptyset$. Durch geeignetes Umnumerieren der Koeffizienten können wir

$$\tilde{V} \cap F = \{t\}^{\#I} \times \tilde{F}$$

schreiben, wobei $\tilde{F}$ wiederum ein affiner Unterraum des $\mathbb{R}^{n-\#I}$ ist mit $\dim \tilde{F} \leq n - (1 + \#I)$. Wegen (3.7) gibt es für jedes $x \in h^{-1}(\tilde{V} \cap F)$ ein $\tilde{x} \in \tilde{V} \cap F$ mit $x_k = \tilde{x}_k$ für alle $k \in J \setminus I$. Also gilt

$$h^{-1}(\tilde{V} \cap F) \subset \mathbb{R}^{\#I} \times \tilde{F}.$$

Da aber $\dim(\mathbb{R}^{\#I} \times \tilde{F}) \leq n - 1$ ist, muß $\mathbb{R}^{\#I} \times \tilde{F}$ und damit auch $\tilde{D}$ als Teilmenge eine Nullmenge sein. Die Behauptung folgt. □

Interessanter als die Frage, ob der Algorithmus stetig ist oder nicht, ist natürlich die Untersuchung des Konvergenzverhaltens. In der Praxis hat das Verfahren stets konvergiert, doch trotz der Einfachheit des Verfahrens scheint es nicht möglich, diese Konvergenz nachzuweisen. Wie wir soeben gesehen haben, ist zum Beispiel der Banachsche Fixpunktsatz nicht anwendbar, denn T ist noch nicht einmal auf

$$D_T = \{x \in \mathbb{R}^n | \exists k \in \mathbb{N}_0 : x = T^k(y)\}$$

[19] Der besseren Übersichtlichkeit wegen wurde auf die Einführung weiterer Indizes verzichtet

Lipschitz-stetig.

Eine andere Frage ist hingegen erstaunlich einfach beantwortbar, nämlich die Frage nach möglichen Grenzwerten von f_k. Eine Antwort gibt der folgende Satz.

Satz 3.2. *$\mathcal{W}^{j_0} : \mathbb{R}^n \to \mathbb{R}^{n-2^{j0}}$ sei die Abbildung, die einem Vektor $y \in \mathbb{R}^n$ die Waveletkoeffizienten ab Level j_0 zuordnet (Anordnung der Koeffizienten spielt hier keine Rolle). Desweiteren sei für den zu untersuchenden Datensatz y vorausgesetzt, daß die im ersten des Schritt des Algorithmus berechnete Standardabweichung echt größer als 0 ist. Dann gilt:*

Die Folge der $(r_k)_{k\in\mathbb{N}}$ konvergiert genau dann gegen 0, wenn $\mathcal{W}^{j_0}(f_k)$ gegen 0 konvergiert.

Beweis. Einerseits gilt nach Definition

$$r_k = ||f_j - \mathrm{Soft}(f_j)||_\infty,$$

andererseits folgt aus der Parzeval-Gleichung, daß

$$||f_j - \mathrm{Soft}(f_j)||_2 = ||\mathcal{W}f_j - \mathcal{W}\mathrm{Soft}(f_j)||_2.$$

Da $||.||_2$ und $||.||_\infty$ im $\mathbb{R}^n$ äquivalent sind, konvergiert also r_k genau dann gegen 0, wenn $||\mathcal{W}f_j - \mathcal{W}\mathrm{Soft}(f_j)||_\infty$ gegen 0 konvergiert.

Dann gilt

$$\begin{aligned} &||\mathcal{W}f_j - \mathcal{W}\mathrm{Soft}(f_j)||_\infty \\ &\qquad = \max_i |(\mathcal{W}f_j)_i - \mathrm{sign}((\mathcal{W}f_j)_i) \cdot |(\mathcal{W}f_j)_i - t| \cdot \{|(\mathcal{W}f_j)_i| > t\}| \\ &\qquad = \max_i \{\min\{|(\mathcal{W}f_j)_i|, t\}\} \\ &\qquad = \min\{||(\mathcal{W}^{j_0}f_j||_\infty, t\}. \end{aligned} \tag{3.8}$$

Damit folgt die Behauptung. □

Aus diesem Satz lassen sich eine Reihe von Schlußfolgerungen ziehen:

1. Traditionelles Wavelet-Thresholding führt in Anwesenheit von Ausreißern zu störenden Peaks in der Kurve des Wavelet-Schätzers, wie wir oben in mehreren Beispielen gesehen haben. Die Kurven, die Algorithmus (3.2) liefert, weisen solche Peaks nicht auf, denn zu deren Darstellung würden High-Level-Koeffizienten benötigt, die aber (wie der Satz zeigt) bei dem Grenzvektor $\hat{f}$, gegen den das Verfahren konvergiert, alle 0 sind, sofern man j_0 relativ klein wählt. In dieser Hinsicht läßt sich der zu analysierende Algorithmus als robust bezeichnen.

2. Die Wahl von j_0 ist offenbar bei Algorithmus (3.2) wesentlich bedeutsamer als bei klassischem Wavelet-Thresholding. Dem Leser ist vielleicht aufgefallen, daß für die Cauchy-verrauschte Sinus-Kurve und für NoisyBlocks deutlich unterschiedliche Werte für j_0 gewählt wurden, obwohl beide Datensätzen die gleiche Länge aufweisen. Erfahrungen mit Thresholding zeigen, daß $j_0 = 5$ bei Datensätzen der Länge 2048 eine gute Wahl ist. Diese Wahl wird auch von Donoho et al. in [17] benutzt. Die Abbildungen 33 und 34 zeigen, was passiert, wenn man in Algorithmus 3.2 für j_0 diese Wahl trifft.

   Um j_0 geeignet zu wählen, kommt folgender Algorithmus in Betracht:

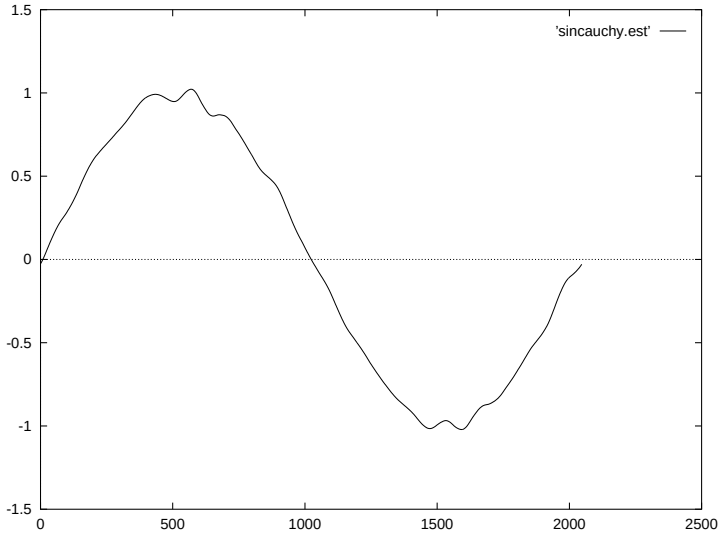

Abbildung 33: Algorithmus (3.2) angewandt auf die mit Cauchy-Rauschen gestörte Sinus-Kurve mit $j_0 = 5$ statt $j_0 = 4$

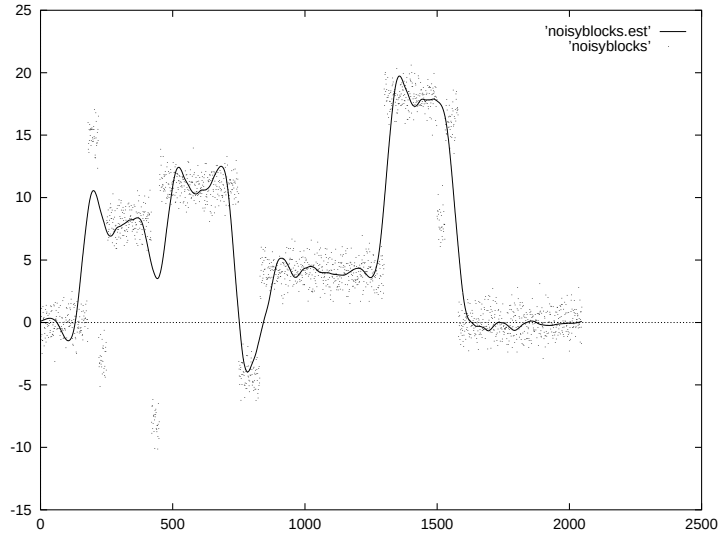

Abbildung 34: Algorithmus (3.2) angewandt auf den Datensatz *NoisyBlocks* – diesmal aber mit $j_0 = 5$ statt $j_0 = 7$

Algorithmus 3.3.

(1) Starte mit $j_0 := 1$.

(2) Wende Algorithmus 3.2 *auf die Daten an und erhalte einen Schätzvektor* $\hat{f}$.

(3) Berechne die maximale Runlänge m *der Vorzeichen der Residuen.*

(4) Falls $m < m_0$*, fertig.*

(5) Erhöhe j_0 *um 1, und weiter bei Schritt (2).*

Die Größe m_0 kann entweder durch Simulation oder theoretische Überlegungen wie in [20] bestimmt werden. Man erhält für die maximale Runlänge einer Folge der Länge n die folgenden Quantile:

n	5%	50%	95%
128	5	7	11
256	6	8	12
512	7	9	13
1024	8	10	14
2048	9	11	15
4096	10	12	16

Für die drei Datensätze liefert Algorithmus 3.2 für verschiedene j_0 Kurven mit den folgenden maximalen Runlängen der Residuen:

j_0	Crash-Daten	Cauchy-Sinus	NoisyBlocks
1	19	28	280
2	11	11	224
3	6	11	182
4	6	11	88
5	4	13	54
6	2	9	30
7	0	8	15
8	0	6	7

Wenn man bei der Wahl von m_0 das 50%-Quantil benutzt, erhält man bei den Motorrad-Daten und der Sinus-Kurve die zu Beginn dieses Abschnitts benutzten Werte für j_0 und bei den NoisyBlocks $j_0 = 8$ (statt $j_0 = 7$).

3. Aus dem Satz folgt aber noch ein anderes Problem: Die Wahl von j_0 bedeutet bei Algorithmus 3.2, zu entscheiden, welche Frequenzen man zuläßt und welche nicht. Dadurch gelangt man in die Situation, in der wir im letzten Kapitel bei Fourier-Reihen-Schätzern gewesen sind, und tatsächlich stößt man bei Anwendung auf die „Doppler"-Daten auf das gleiche Dilemma, das einem schon dort begegnet ist: Entweder werden kleine Frequenzen eliminiert oder die Kurve ist zu wackelig. Abbildung 35 zeigt das Ergebnis mit $j_0 = 7$, wobei j_0 mit Algorithmus 3.3 ermittelt wurde und zu einer maximalen Runlänge von 15 bei 2048 Daten führt. [20]

[20] Eine Lösung dieses Problems könnte möglicherweise darin bestehen, Wavelet Shrinkage derart zu verallgemei-

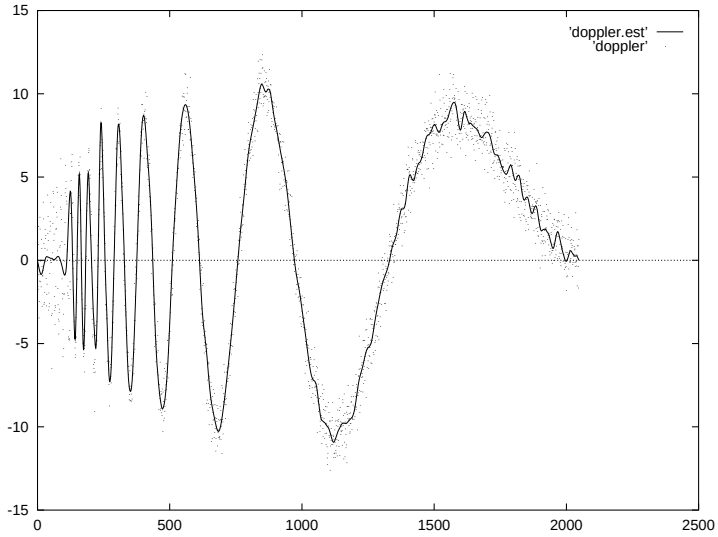

Abbildung 35: Algorithmus (3.2) angewandt auf den Datensatz *Doppler* mit $j_0 = 7$

Wir wollen nun noch einige Änderungen des vorgestellten Verfahrens diskutieren.

1. Zunächst ist zu betonen, daß es sehr wichtig ist, daß Soft-Thresholding verwendet wird. Hard-Thresholding würde, wie wir in Lemma (2.1) gesehen haben, genügend große Ausreißer unverändert lassen. Das entstehende Residuum wäre also somit an dieser Stelle 0 und der Ausreißer würde nicht eliminiert.

2. Statt die Varianz und damit den Threshold konstant zu wählen, kann man die Varianz natürlich auch während des Verfahrens in regelmäßigen Abständen neu berechnen. Dies sollte dann allerdings sinnvollerweise nicht über den MAD der „Fine Scale"-Koeffizienten geschehen, da diese in der Regel zu schnell gegen 0 konvergieren. Statt dessen kann man aber den MAD der Residuen berechnen (und ggfs. wieder geeignet skalieren, um zum Beispiel Konsistenz mit der Normalverteilung zu erhalten). Dadurch ergibt sich manchmal eine leichte Verbesserung der Laufzeit.

3. Bei den Beispielen wurde für die Abbruchgrenze δ stets 0.025 gewählt – unabhängig von Skala und Varianz der Daten. Diese Lösung ist noch zu verbessern, denn es drohen zwei Gefahren: Für Datensätze wie die Motorrad-Daten mit großer Skala und Varianz scheint 0.025 unnötig klein und ist es auch tatsächlich, denn schon nach 16 der 297 Iterationen lieferte Soft-Thresholding eine Kurve, die nur minimal von der Schlußkurve abwich. Während hier das Problem vorliegt, daß man unnötig viel Rechenarbeit leistet, kann es bei Datensätzen mit sehr kleiner Varianz andersherum passieren, daß Ausreißer übersehen werden, denn wie wir in Lemma (2.2) gesehen haben, ist r_k nach oben durch eine Konstante beschränkt, die proportional zum Threshold, damit aber auch im Falle von VisuShrink proportional zur Varianz ist. Es braucht also bei vorhandenem Ausreißer nur die Skala

nern, daß die Thresholding-Regel nicht nur auf die Koeffizienten angewandt wird, die größer als ein Cut-Off-Point j_0 sind, sondern für beliebiges $I \subset \{1, \ldots, n\}$ auf alle mit Indizes aus I indizierten Koeffizienten. Allerdings stellt sich die Frage nach der Wahl von I.

des übrigen Rauschens klein genug gemacht werden, damit auch r_k kleiner als jede feste Schranke ist.

Eine bessere Wahl könnte zum Beispiel darin bestehen, δ in Abhängigkeit von der Varianz zu wählen, etwa $\delta = c \cdot \sigma$, wobei die Konstante c von der Ordnung des Wavelets sowie insbesondere der Wahl von j_0 und der Länge des Datensatzes abhängt. Eine Möglichkeit, um c per Simulation zu bestimmen, sieht so aus: Man zieht Stichproben der Länge n aus einer Standardnormalverteilung, setzt jeweils einen Datenpunkt auf 1.96 (0.975-Quantil), wendet Soft-Thresholding an und berechnet das entstehende Residuum. c wählt man als das 0.05-Quantil dieser Residuuen. Die folgenden Tabellen zeigen so ermittelte Konstanten für das Daubechies-Wavelet der Ordnung 3:

n	$j_0 = 3$	$j_0 = 4$	$j_0 = 5$	$j_0 = 6$
128	1.53	1.42	1.00	0.07
512	1.68	1.64	1.55	1.43
1024	1.84	1.68	1.64	1.55
2048	1.88	1.84	1.70	1.64

und der Ordnung 6:

n	$j_0 = 3$	$j_0 = 4$	$j_0 = 5$	$j_0 = 6$
128	1.39	1.21	0.86	0.29
512	1.71	1.66	1.37	1.20
1024	1.82	1.71	1.65	1.38
2048	1.85	1.82	1.73	1.67

Angewandt auf die drei untersuchten Datensätze bricht das Verfahren nach 33 (Motorrad), 535 (Sinus) und 410 (NoisyBlocks) Schritten ab und lieferte im wesentlichen die gleichen Ergebnisse wie oben, wo der Algorithmus erst nach 297, 2119 und 3299 Iterationen terminierte.

Eine andere Alternative besteht darin, aufgrund der Aussage von Satz (3.2) abzubrechen, sobald alle Wavelet-Koeffizienten der Level $j_0, \ldots, J$ in der Wavelet-Darstellung von f_k betragsmäßig kleiner als der Threshold sind.

4. Schließlich noch ein paar Überlegungen zur Laufzeit des Algorithmus. Allgemeine Aussagen sind nicht möglich, denn die Laufzeit variiert insbesondere mit der Länge des Datensatzes, der Anzahl der Ausreißer, der Größe der Ausreißer und der Form der Kurve. Mit der eben vorgestellten Bestimmung der Abbruchgrenze von δ konnte die Laufzeit auf 25 Sekunden im Fall der Sinus-Kurve reduziert werden (NoisyBlocks: 20 Sekunden, Crash-Daten: < 1 Sekunden), wobei das j_0 allerdings vorgegeben und nicht mit Algorithmus 3.3 bestimmt wurde. [21] Ein Großteil der Zeit wurde jedoch dazu verwendet, den größten Ausreißer bei 1102 zu elimieren. Deshalb hätte schon ein einziger zweiter Ausreißer der gleichen Größenordnung fast zu einer Verdopplung der notwendigen Iterationen und damit der Laufzeit geführt. Eine Optimierung der Rechenzeit könnte man also insbesondere erzielen, indem man

[21] Die Zeiten wurden gemessen auf einem Pentium-PC mit 90 MHz unter dem Betriebssystem Linux

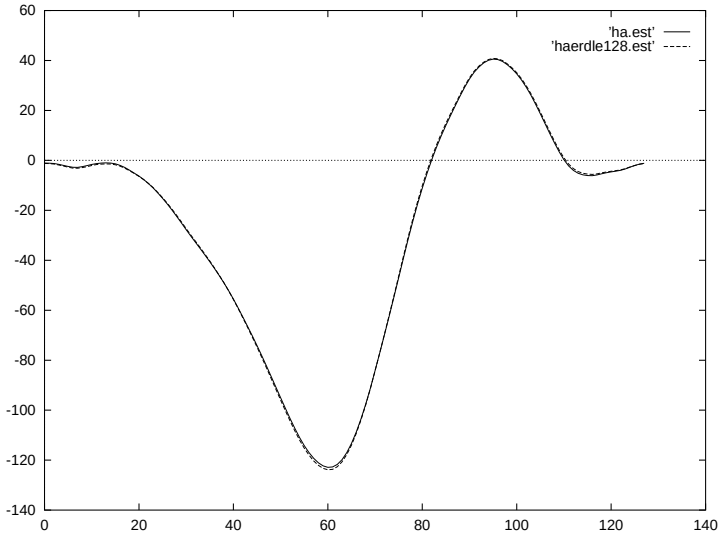

Abbildung 36: Mehrfache Datenabänderung pro Iteration bei den Motorrad-Daten

(a) in jedem Schritt mehr als nur einen Datenpunkt ändert,

(b) jeden zu ändernden Datenpunkt nicht durch den Wert des Wavelet-Schätzers an der jeweiligen Stelle ersetzt, sondern versucht, direkt einen besseren Schätzwert zu finden.

Zunächst zum ersten Punkt: Die Abbildungen 36, 37 und 38 zeigen, was passiert, wenn man in jedem Schritt alle Datenpunkte auf den Thresholding-Schätzer zurückzieht, bei denen das Residuum größer als $0.95 \cdot r_k$ ist, wobei r_k wieder das größte Residuum bezeichne. Zum Vergleich ist jeweils das Ergebnis von Algorithmus 3.2 abgebildet. Es ist kein wesentlicher Unterschied erkennbar. Die Anzahl der Iterationen und die Rechenzeit konnten allerdings weiter reduziert werden, nämlich auf 22 (Motorrad), 273 (Sinus) und 55 (NoisyBlocks) Iterationen.

Der zweite Punkt bereitet jedoch noch Probleme. Probiert wurde eine Schätzung des Datenpunktes durch einem lokalen Mittelwert oder Median, doch stets geriet der Algorithmus bei der Sinus-Kurve in eine Endlosschleife. Der Grund dafür scheint auf den ersten Blick kurios: Die Schätzungen durch lokalen Mittelwert sind zu gut!

Ursache der Probleme ist eine ungüstige Ausreißerkonstellation nahe der Datenpunkte $y_{1153} = 17.68$ und $y_{1154} = -23.19$. Das größte Residuum entsteht nämlich weder bei 1153 noch bei 1154, sondern bei 1155, wo aber kein Ausreißer vorliegt. Lokale Mittelwert- oder Medianbildung schätzen den Datenpunkt einigermaßen korrekt und ändern ihn nur wenig (später gar nicht) ab. Bei der nächsten Iteration ergibt sich also wieder genau das gleiche Problem usw.

Algorithmus 3.2 und seine Varianten hingegen ändern den eigentlich korrekten Datenpunkt y_{1155} deutlich ab und erreichen damit, daß bei der nächsten Iteration das größte Residuum an der Stelle 1154 angenommen wird.

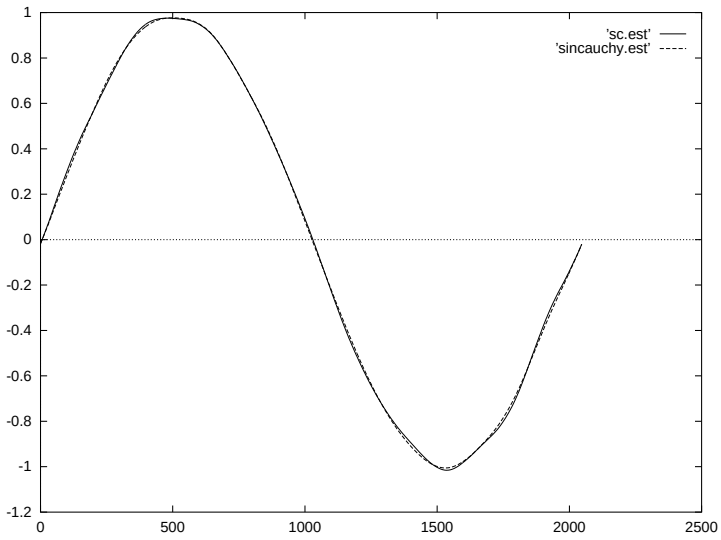

Abbildung 37: Mehrfache Datenabänderung pro Iteration bei der Cauchy-Sinus-Kurve

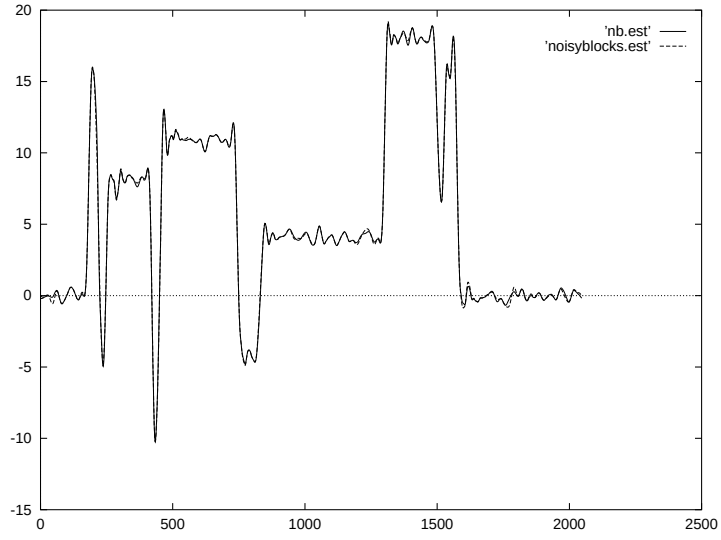

Abbildung 38: Mehrfache Datenabänderung pro Iteration bei den NoisyBlocks

3.2 Ausreißererkennung mit der Wavelet-Transformation

Ausgangspunkt des letzten Abschnittes war der Gedanke, daß Anwendung von Soft-Thresholding in der Nähe von Ausreißern zu verhältnismäßig großen Residuen führt, anhand derer man sie identifizieren kann. Zurückziehen auf den Thresholding-Schätzer milderte die Ausreißer dann, mehrfaches Iterieren eliminierte sie. Während dort also das Problem auf der Ebene der Daten betrachtet wurde, wollen wir in diesem Abschnitt versuchen, Ausreißer anhand ihrer Wavelet-Koeffizienten zu identifizieren. Idee ist dabei, daß Ausreißer typischerweise zu großen FineScale-Koeffizienten führen.

Bevor wir einen Algorithmus vorstellen, müssen wir zunächst die letzte Aussage etwas präzisieren und den Effekt von Ausreißern auf die Wavelet-Koeffizienten untersuchen. Dazu sei nun ein Daubechies-Wavelet ψ der Ordnung m und die zugehörige Diskrete Wavelet Transformation $\mathcal{W}$ betrachtet. Außerdem sei A_i das lineare Funktional $A_i : \mathbb{R}^n \to \mathbb{R}$ mit $A_i(y) = (\mathcal{W}y)_i$, das einem Datenvektor y den i-ten Wavelet-Koeffizienten zuordnet. Aus dem ersten Kapitel dieser Arbeit ist klar, daß für Koeffizienten in feinen Leveln die Matrix des korrespondierenden A_i aus vielen Nullen besteht. Falls die Datenpunkte y_i die Einfluß auf A_i haben, sich modellieren lassen als $y_i = p(i/n) + \varepsilon_i$, wobei p ein Polynom vom Grad höchstens m und $\varepsilon \sim \mathfrak{N}(0, \sigma)$ unabhängig identisch verteiltes Rauschen ist, dann folgt aus Lemma 1.1 und der Orthonormalität der Diskreten Wavelet-Transformation sofort, daß auch $A_i \sim \mathfrak{N}(0, \sigma)$. Wir dürfen also erwarten, daß dort, wo ein Polynom m-ten Grades lokal ein gutes Modell für die Daten darstellt, die Wavelet-Koeffizienten klein sind, solange keine Ausreißer vorliegen. Weiterhin ist natürlich klar (siehe auch Beweis zu Lemma 2.1), daß bei Konvergenz *eines* Datenpunktes gegen ∞ auch alle betroffenen Wavelet-Koeffizienten gegen ∞ oder $-\infty$ konvergieren.

Konkreteres kann man leider über den Zusammenhang zwischen Ausreißern und den Koeffizienten der Diskreten Wavelet-Transformation nicht sagen. Man beachte die folgenden Probleme:

- Sei A_i das Funktional, das mit einem Wavelet des feinsten Levels korrespondiere. A_i entspricht dann im wesentlichen dem Filter $\mathcal{G}$ von Kapitel 1. Dann sind genau $2m+2$ Einträge der Matrix zu A_i von Null verschieden. Der Kern von A_i ist ein Vektorraum der Dimension $2m + 1$. Andererseits ist

  $$V := \{v \in \mathbb{R}^{2m+2} : \exists \text{ Polynom } p \text{ vom Grad } m \text{ mit } p(i/n) = v_i\}$$

  ein Vektorraum der Dimension $m+1$ mit $V \subset \operatorname{Ker} A_i$. Es gibt somit für $m > 0$ (also außer beim Haar-Wavelet) Vektoren $v \in \operatorname{Ker} A_i$, die nicht auf einem Polynom m-ten Grades liegen. Durch entsprechende Skalierung gelangt man dann zu Datenpunkten, die man in den Anwendungen als Aureißer betrachten würde und die Einfluß auf den i-ten Koeffizienten haben, ohne daß dieser groß ist. Diese Beobachtung beeinflußt nicht unbedingt die Analyse, ob Ausreißer vorliegen oder nicht, da vermutlich mindestens ein anderer Wavelet-Koeffizient das Vorliegen eines Ausreißers anzeigt, aber sie erschwert die Entscheidung, welcher der Datenpunkte der Ausreißer ist.

- Verschieben (bzw. Rotieren) des Datensatzes um einen Punkt führt unter Umständen zu verschiedenen Ergebnissen, da die Diskrete Wavelet Transformation nicht translationsinvariant ist. Betrachte dazu folgendes Beispiel. Es sei $y_i = 0$ für alle $i \leq 128$ mit $i \neq 64$ und $y_{64} = 10$. Die Wavelet-Koeffizienten des feinsten Levels bzgl. des Daubechies-Wavelets der Ordnung 4 sind fast alle 0 bis auf:

d_{97}	-1.60
d_{98}	7.24
d_{99}	2.42
d_{100}	-0.78
d_{101}	0.13

Wir setzen nun $y_{63} = 10$ und $y_{64} = 0$, wenden wieder die DWT an und erhalten:

d_{97}	6.04
d_{98}	1.39
d_{99}	-0.32
d_{100}	-0.06
d_{101}	0.03

Bei Hinzuaddieren von Stichproben einer Standardnormalverteilung sollten die Datenpunkte 64 bzw. 63 als Ausreißer identifiziert werden. Zählt man jedoch, wieviele Koeffizienten größer als eine vorgegebene Grenze (z.B. 1.96) sind, so wird man mit hoher Wahrscheinlichkeit im ersten Fall eine größere Zahl als im zweiten erhalten.

Beim Haar-Wavelet, wo sich die Koeffizienten als die skalierte Differenz zweier Nachbarpunkte berechnen, ergibt sich die unangenehme Situation, daß Doppel-Ausreißer (so seien zwei direkt nebeneinander liegende Ausreißer bezeichnet) je nach Lage zu zwei oder gar keinem großen Koeffizienten im feinsten Level führen.

- Große Koeffizienten müssen nicht auf Ausreißer zurückzuführen sein, sondern können auch von Unstetigkeiten stammen, die ein mit den Daten adäquates Modell eventuell aufweist.

Wenigstens das zweite Problem läßt sich beseitigen. Silverman und Nason haben in [27] die *stationäre Wavelet-Transformation* vorgestellt, die man aus der Diskreten Wavelet-Transformation im wesentlichen (d.h. bis auf Anordnung der Koeffizienten) durch Weglassen des binären Dezmierungsoperators $\mathcal{D}_0$ erhält. Coifman und Donoho beschreiten in [6] ähnliche Wege und wenden die DWT nicht nur auf die Daten selbst, sondern auch auf Verschiebungen an. Die Stationäre Wavelet-Transformation läßt sich mit Coifmans *CycleSpinning* mit einem Aufwand von $O(n\log(n))$ berechnen und führt zu $J\cdot 2^J$ Koeffizienten und somit zu einer überbestimmten Darstellung der Daten. Erste Ideen für den Einsatz der SWT in der nicht-parametrischen Regression sehen so aus, daß auf die Koeffizienten der SWT eine Thresholding-Funktion angewandt wird und dann durch geeignete Mittelwertbildung die Koeffizienten der SWT in eine Koeffizientendarstellung einer DWT transformiert werden, die dann wieder invertiert werden kann. Andere Ideen sind die sogenannten „Best Shift"-Algorithmen. Hier wird auf jede der n möglichen Verschiebungen der Daten die DWT angewandt. Minimierung eines „Entropie"-Funktionals (wie zum Beispiel die l^2-Norm der Koeffizienten nach Thresholding) gibt dann die „beste" Verschiebung. Anwendung der inversen DWT auf diese Koeffizienten gefolgt von Zurückrotieren ergibt den „Best Shift"-Schätzer.

Wir verwenden die Stationäre Wavelet-Transformation für einen anderen Zweck, nämlich für die Identifizierung und Eliminierung von Ausreißern. Da wir ohnehin an den feinsten Leveln interessiert sind, wollen wir hier nicht näher darauf eingehen, wie Coifmans $O(n\log(n))$-Algorithmus zur Bestimmung der Koeffizienten aussieht. Bei uns sei $\mathcal{S}_j$ der Endomorphismus im $\mathbb{R}^n$, der einem Vektor $v \in \mathbb{R}^n$ die Koeffizienten des j-ten Levels einer SWT zuordnet. Die

Matrix zu $\mathcal{S}_j$ hat dann die Gestalt

$$
\begin{pmatrix}
c_1 & c_2 & \dots & c_{r-1} & c_r & & & \\
 & c_1 & c_2 & & c_{r-1} & c_r & & \\
 & & \ddots & \ddots & & \ddots & \ddots & \\
 & & & c_1 & c_2 & & c_{r-1} & c_r \\
c_r & & & & c_1 & c_2 & & c_{r-1} \\
\vdots & \ddots & & & & \ddots & \ddots & \vdots \\
c_3 & & c_r & & & & c_1 & c_2 \\
c_2 & & & c_r & & & & c_1
\end{pmatrix}. \tag{3.9}
$$

Die Koeffizienten (c_i) stimmen im Falle $j = J$ mit den Filterkoeffizienten von $\mathcal{H}$ aus dem Abschnitt über die Diskrete Wavelet-Transformation überein. Würde man mit $\mathcal{W}_j : \mathbb{R}^n \to \mathbb{R}^{2^{j-1}}$ die Abbildung bezeichnen, die einem Datenvektor y die Koeffizienten des j-ten Levels einer DWT zuordnet, so entsteht die Matrix zu $\mathcal{W}_j$ aus Matrix (3.9) durch Extrahieren der $i \cdot 2^{J-j+1} + 1$-ten Zeilen ($i \in \{0, \dots, 2^{j-1}\}$). Interessant ist die folgende kurze Bemerkung:

Bemerkung 3.1. *Der Endomorphismus $\mathcal{S}_j$ ist für $j \geq j_0$*[22] *singulär.*

Beweis. Da sich wegen Lemma 1.1 die Eigenschaft der verschwindenden Momente auf die Diskrete Wavelet-Transformation überträgt, gilt insbesondere $\sum_{i=1}^{r} c_i = 0$. Aus dem Grund ergibt die Summe der Zeilen- oder Spaltenvektoren der Matrix (3.9) den Nullvektor, sind also nicht linear unabhängig. Also ist $\mathcal{S}_j$ singulär. □

Aufgrund der Gestalt der Matrix (3.9) ist klar, daß die SWT translationsinvariant ist, es verschwindet also das zweite der oben angesprochenen Probleme, wenn man versucht, Ausreißer mit der SWT statt mit der DWT zu identifizieren. Grundidee des Verfahrens, das weiter unten vorgestellt wird, wird im wesentlichen sein, durch Abänderung eines Datenpunktes (oder wenigen benachbarten) die Anzahl der FineScale-Koeffizienten, die größer als ein Threshold sind, bzgl. der Stationären Wavelet-Transformation zu verringern. Man kann diese Idee auch von einer praktischen Seite aus sehen: Wir sind daran naturgemäß daran interessiert, eine glatte Kurve zu erhalten. Dazu müssen aber in der Wavelet-Darstellung der Schätzfunktion möglichst viele Koeffizienten der hohen Level gleich 0 sein.

Es ist jedoch zu überlegen, in welcher Weise diese Abänderungen geschehen sollen. Wenn wir annehmen, daß das Rauschen von einer Normalverteilung stammt und nur an einigen Stellen durch zusätzliche Ausreißer gestört ist, [23] scheint eine Minimierung der Wavelet-Koeffizienten in der l^2-Norm gerechtfertigt.

Genauer: Angenommen, wir wollen bei gegebenen Daten $y_i = f(i/n) + \varepsilon_i$ die Funktionswerte $f(k/n), \dots, f((k+l)/n)$ schätzen. Wir betrachten dann den folgenden Schätzer: Minimiere

$$||\mathcal{S}_J \tilde{y}||_2^2 \tag{3.10}$$

unter allen $\tilde{y} \in \mathbb{R}^n$, für die $\tilde{y}_i = y_i$ für alle $i < k$ und $i > k+l$ gilt. Wir schätzen dann $\hat{f}(i/n) = \tilde{y}_i$ für $i \in \{k, \dots, k+l\}$. Es gilt der folgende Satz:

[22] j_0 sei hier wieder wie in 1.18 im Abschnitt über die Diskrete Wavelet-Transformation definiert.

[23] Diese Annahme scheint bei einer Reihe von Anwendungsbeispielen durchaus nicht ungerechtfertigt zu sein. Man denke zum Beispiel an eine schlechte Telefonleitung, wo ein permanent vorhandenes Grundrauschen manchmal durch ein Knacken gestört wird.

Satz 3.3. *Falls das Daubechies-Wavelet der Ordnung m mit $N = 2m+2$ Koeffizienten benutzt wird und falls ein Polynom von Grad kleiner oder gleich m existiert, so daß (ggfs. durch periodisches Fortsetzen am Rand) $f(i/n) = p(i/n)$ und $\mathbb{E}(\varepsilon_i) = 0$ für alle $i \in \{k-N+1, \ldots, k+l+N-1\}$ ist und falls außerdem $l < n - N$,* [24] *dann gilt*

$$\mathbb{E}(\hat{f}(i/n)) = f(i/n).$$

Beweis. Seien

$$V_1 := \{x \in \mathbb{R}^n : \forall i \in \{k, \ldots, k+l\} : x_i = 0\}$$

und

$$V_2 := \{x \in \mathbb{R}^n : \forall i \in \{1, \ldots, k-1, k+l+1, \ldots, n\} : x_i = 0\}.$$

Es ist dann $\mathbb{R}^n = V_1 \oplus V_2$. Weiterhin seien P_1 und P_2 die orthogonalen Projektionen auf V_1 und V_2. Offenbar gilt dann für $v \in \mathbb{R}^n$, daß

$$(P_1 v)_i = \begin{cases} 0, & \text{falls } i \in \{k, \ldots, k+l\} \\ v_i, & \text{sonst} \end{cases}$$

und

$$(P_2 v)_i = \begin{cases} v_i, & \text{falls } i \in \{k, \ldots, k+l\} \\ 0, & \text{sonst} \end{cases}.$$

Mit diesen Bezeichnungen ist klar, daß man die Lösung des Minimierungsproblems (3.10) erhält, wenn man

$$||\mathcal{S}_J(P_1(y) + P_2(v))||_2^2 \tag{3.11}$$

für alle $v \in \mathbb{R}^n$ minimiert und $\tilde{y} = P_1(y) + P_2(v)$ wählt. Mit $A := \mathcal{S}_J P_2$ und $b := -\mathcal{S}_J P_1 y$ muß also $||Av - b||_2^2$ minimiert werden. Aus der linearen Regression ist bekannt, daß dies genau dann der Fall ist, wenn

$$A^T A v = A^T b,$$

also gilt auch

$$A^T A \mathbb{E}(v) = A^T \mathbb{E}(b).$$

Um Schreibarbeit zu sparen, sei $g \in \mathbb{R}^n$ mit $g_i = f(i/n)$, es ist dann

$$\begin{aligned}(\mathbb{E}(P_1 y))_i &= \begin{cases} 0, & \text{falls } i \in \{k, \ldots, k+l\} \\ g_i, & \text{sonst} \end{cases} \\ &= g - P_2 g\end{aligned}$$

und damit

$$A^T A \mathbb{E}(v) = A^T \mathcal{S}_J (P_2 g - g) = A^T A g - A^T \mathcal{S}_J g.$$

Die Voraussetzung $l \leq n - N$ sichert, daß die Spaltenvektoren von A, die keine Nullvektoren sind, linear unabhängig sind, denn A läßt sich (ggfs. durch Zeilenvertauschungen) in der Form

$$\left(\begin{array}{c|c} \text{B} & 0 \\ \hline 0 & 0 \end{array}\right)$$

[24] Diese Ungleichung ist natürlich in der Praxis immer erfüllt, da in den gängigen Wavelet-Implementationen N nicht größer als 20 wird, typische Werte für n wohl nicht unter 128 liegen und es erst recht keinen Sinn macht, mehr als die Hälfte der Datenpunkte aufgrund der übrigen Daten zu schätzen.

mit

$$B := \begin{pmatrix} c_r & & & \\ c_{r-1} & c_r & & \\ \vdots & & \ddots & \\ c_1 & & & c_r \\ & c_1 & & c_{r-1} \\ & & \ddots & \vdots \\ & & & c_1 \end{pmatrix}$$

schreiben und hat damit den Spaltenrang $l+1$. Durch weitere Überlegungen stellt man fest, daß dann der Rang von $A^T A$ ebenfalls gleich $l+1$ ist. Da offensichtlich $V_1 \subset \operatorname{Ker} A^T A$ ist, folgt daraus insbesondere, daß $\operatorname{Ker} A^T A = V_1$. Es ist also nur noch zu überlegen, daß

$$A^T \mathcal{S}_J g = 0, \tag{3.12}$$

denn dann ist $\mathbb{E}(v) - g \in \operatorname{Ker} A^T A$ und es gibt ein $w \in \operatorname{Ker} V_1$ mit $\mathbb{E}(v) = g + w$ und damit

$$\mathbb{E}(\tilde{y}_k, \ldots, \tilde{y}_{k+l}) = \mathbb{E}(P_1(y_k, \ldots, y_{k+l}) + P_2(v_k, \ldots, v_{k+l})) = (g_k, \ldots, g_{k+l}),$$

woraus die Behauptung folgt.

Offensichtlich hat die Matrix (3.9) in den Zeilen $k - N + 1$ bis $k + l$ nur Einträge in den Spalten $k - N + 1$ bis $k + l + N - 1$. Da aber nach Voraussetzung $g_{k-N+1}, \ldots, g_{k+l+N-1}$ auf einem Polynom vom Grad kleiner oder gleich m liegen, folgt nach Lemma 1.1, daß für $\tilde{g} := \mathcal{S}_J g$ die Einträge g_{k-N+1} bis g_{k+l} alle verschwinden. Betrachtet man andererseits die Zeilen k bis $k + l$ von $\mathcal{S}_J^T$, so sieht man, daß dort nur in den Spalten $k - N + 1$ bis $k + l$ von Null verschiedene Einträge stehen. Folglich sind die k-te bis $k+l$-te Komponente von $\mathcal{S}_J^T \tilde{g}$ alle gleich Null. Anschließendes Anwenden von P_2 setzt auch die übrigen Komponenten auf Null. Damit folgt (3.12). □

Nach diesen Vorarbeiten können wir nun endlich den Algorithmus präsentieren. Dazu sei m_{max} die Ordnung des Daubechies-Wavelets, mit dem wir die Schätzfunktion $\hat{f}$ konstruieren wollen. l_{max} gebe an, wieviele Datenpunkte höchstens gleichzeitig abgeändert werden dürfen. Schließlich ist klar, daß man bei der oben vorgestellten Methode zur Schätzung der Datenpunkte $\hat{f}(k/n), \ldots, \hat{f}((k+l)/n)$ auch S_J durch S_j ersetzen kann. Dies ist bei der Erkennung von benachbarten Ausreißern manchmal ein Vorteil. d_{max} sei die Anzahl der Level, auf die wir unser Minimierungsverfahren anwenden wollen.

Algorithmus 3.4.

(1) Bestimme einen Threshold t, zum Beispiel wie bei VisuShrink.

(2) Starte mit $m := 0$.

(3) Setze $l := 0$.

(4) Setze $d := 0$.

(5) Setze $k := 1$.

(6) Schätze die Datenpunkte $\hat{f}(k/n), \ldots, \hat{f}((k+l)/n)$ wie oben beschrieben. $\tilde{y}$ sei wieder der Datenvektor, der aus y entsteht, indem y_k bis y_{k+l} durch die Schätzwerte ersetzt werden.

(7) Prüfe ob die beiden folgenden Bedingungen erfüllt sind:

(a) Die Anzahl der Koeffizienten in den Leveln $J - d_{max}$ bis J bei einer SWT von $\tilde{y}$, deren Betrag größer als t ist, ist echt kleiner als die entsprechende Anzahl bzgl. einer SWT von y.

(b) Wenn ein Koeffizient der Level $J - d$ (N.B.: nicht $J - d_{max}$!) bis J einer SWT von $\tilde{y}$ im Betrag größer als t ist, dann ist auch der entsprechende Koeffizient bei einer SWT von y im Betrag größer als t.

(8) Falls die Bedingungen erfüllt sind, so ersetze y durch $\tilde{y}$.

(9) Falls $k + l + 1 < n$, so setze $k := k + 1$ und weiter bei (6).

(10) Falls Bedingung (7) für ein k erfüllt war, starte noch mal bei (5).

(11) Falls $d + 1 < d_{max}$, so setze $d := d + 1$ und weiter bei (5).

(12) Falls $l < l_{max}$, so setze $l := l + 1$ und weiter bei (4).

(13) Falls $m < m_{max}$, so setze $m := m + 1$ und weiter bei (3).

(14) Wende auf y Hard- oder Soft-Thresholding an und benutze dabei den Threshold t.

Nach so viel Theorie wollen wir uns nun zunächst einige Ergebnisse anschauen. Die Abbildungen 39 bis 42 zeigen das Resultat bei den schon mehrfach untersuchten Beispielen. Die folgende Tabelle zeigt, wie die Parameter gewählt wurden:

Datensatz	l_{max}	m_{max}	d_{max}	j_0
Crash-Daten	3	6	2	3
SinCauchy	7	6	5	3
NoisyBlocks	6	6	3	4
Doppler	5	6	3	4

In der nächsten Tabelle ist dargestellt, wieviele Datenpunkte von dem Algorithmus modifiziert wurden:

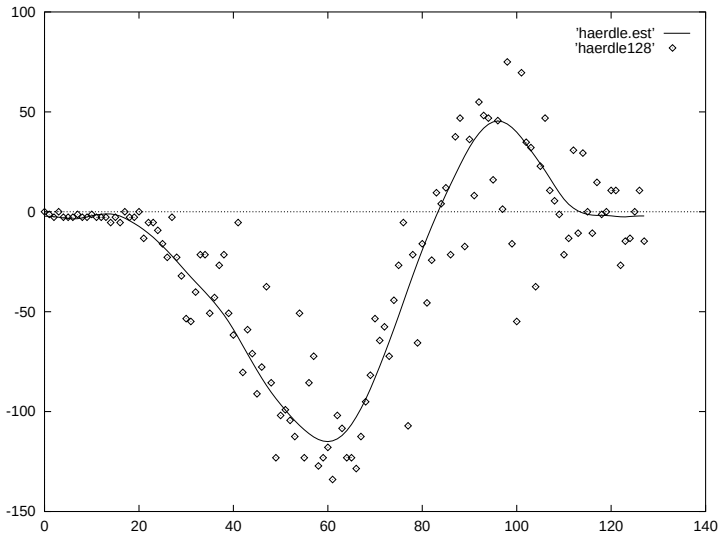

Abbildung 39: Anwendung von Algorithmus 3.4 auf die Motorrad-Daten

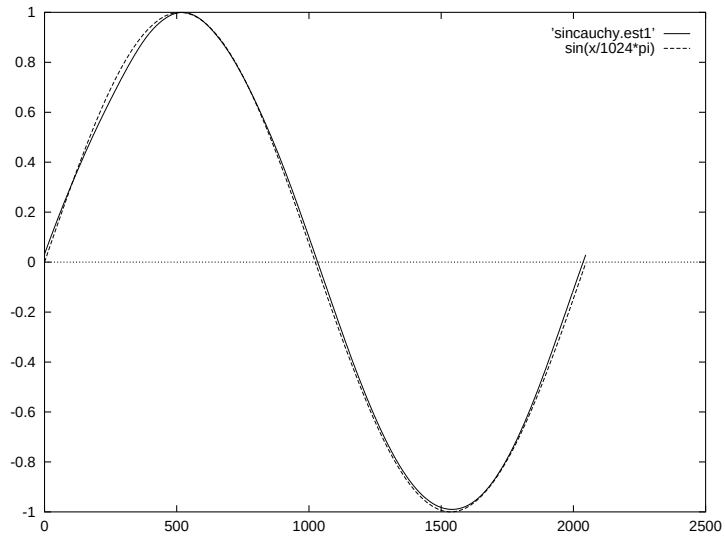

Abbildung 40: Anwendung von Algorithmus 3.4 auf die Cauchy-verrauschte Sinus-Kurve, im Vergleich wie immer eine echte Sinus-Kurve

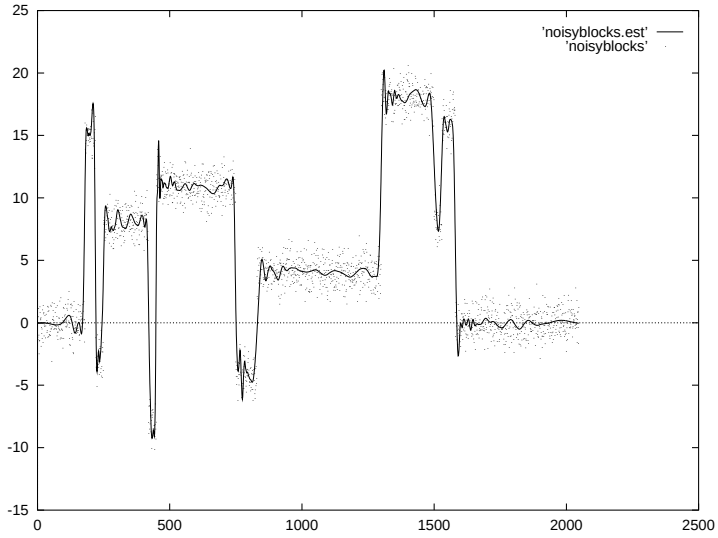

Abbildung 41: Anwendung von Algorithmus 3.4 auf den Datensatz „NoisyBlocks”

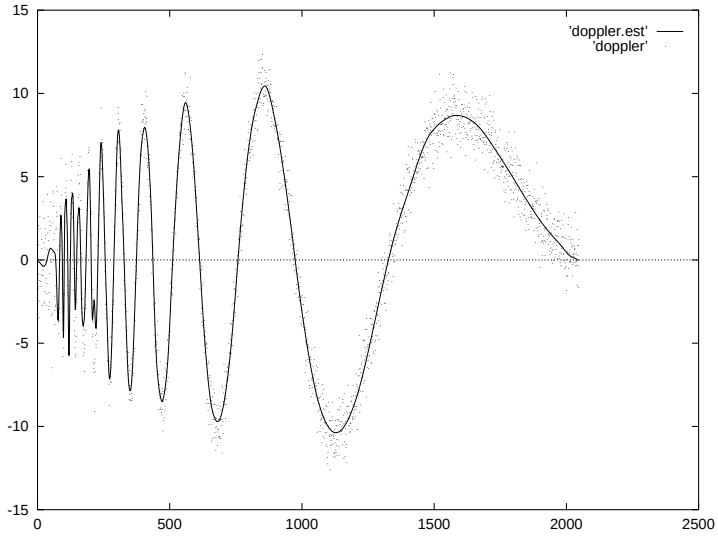

Abbildung 42: Anwendung von Algorithmus 3.4 auf den Datensatz „Doppler”

Datensatz	Größe des Datensatzes	Anzahl der geänderten Punkte
Crash-Daten	128	20
SinCauchy	2048	192
NoisyBlocks	2048	150
Doppler	2048	127

Offenbar ändert Algorithmus 3.4 nur verhältnismäßig wenige Datenpunkte ab, was darauf hindeutet, daß die nicht durch Ausreißer kontaminierten Datenpunkte im wesentlichen erhalten bleiben. Im Falle der Cauchy-verrauschten Sinus-Kurve wurden nur 34 Datenpunkte geändert, an denen die Differenz zwischen Datenpunkt und der echten Sinus-Kurve kleiner als 0.5 ist.[25] Die Abbildungen 43 und 44 zeigen, wie bei den Datensätzen „NoisyBlocks" und „Doppler" die Daten nach den Änderungen aussehen. Außerdem sind die Datenpunkte aus den Originaldaten, die verändert wurden, hervorgehoben eingezeichnet. Wie zu erwarten war, wurden nahe der Unstetigkeitsstellen der „NoisyBlocks" und da, wo die Doppler-Kurve stark oszilliert, Punkte geändert. Offenbar hat dies das Ergebnis jedoch kaum verschlechtert: Was den Doppler-Datensatz betrifft, so besteht in dem Bereich von 0 bis 80, also dort, wo die Schätzfunktion von Algorithmus 3.4 die Oszillation nicht mitmacht, eine komplette Sinus-Schwingung aus höchstens 10 Punkten – da $l_{max} = 5$ gewählt wurde, also erlaubt wurde, bis zu fünf Punkte gleichzeitig abzuändern, war sogar zu hoffen, daß hier die Schätzfunktion konstant ist (man denke in diesem Zusammenhang an das Grundproblem von robuster nicht-parametrischer Regression, das am Ende des zweiten Kapitels kurz beschrieben wurde). Bezüglich der „NoisyBlocks" zeigt ein direkter Vergleich in Abbildung 45 zwischen der Schätzfunktion von Algorithmus 3.4 und dem Hard-Thresholding-Schätzer, daß die Steigung nahe der Unstetigkeitsstellen nur minimal flacher geworden ist. Was man aus drucktechnischen Gründen in der Abbildung leider nur schwer erkennen kann, ist, daß in der Kurve von Algorithmus 3.4 einige unmotivierte Wackler fehlen, die in der Thresholding-Kurve noch vorhanden sind, aber das spricht natürlich nur für den vorgestellten Algorithmus.

Auf der anderen Seite sieht man in den Abbildungen aber auch, daß von Ausreißern hervorgerufene Peaks nicht mehr vorhanden sind. Wavelet-Transformationen der Schätzfunktionen zeigen, daß tatsächlich fast alle Koeffizienten in den jeweils betrachteten Leveln verschwinden.[26] Das war allerdings aufgrund der Konstruktion von Algorithmus 3.4 durchaus zu erwarten.

Leider bedeutet dies jedoch nicht, daß Ausreißer auf das vorgestellte Verfahren keinen Einfluß haben oder gar daß alle Ausreißer durch den Algorithmus eliminiert werden. Bei der Sinus-Kurve ist dies der Schätzfunktion zwar nicht anzumerken – eine genaue Untersuchung der abgeänderten Daten zeigt aber, daß zum Beispiel 25 Datenpunkte, wo die Differenz zu einer echten Sinus-Kurve größer als 0.9 ist, nicht geändert wurden. Deutlicher ist dieses Problem jedoch bei den Motorrad-Daten zu erkennen, wo das Minimum der Schätzfunktion ziemlich hoch liegt. Betrachtet man in Abbildung 46 die geänderten Daten, so sieht man den Grund dafür, der darin besteht, daß drei Ausreißer an den Punkten 55, 57 und 58 nicht geändert wurden und die Kurve nach oben ziehen. Ihr Einfluß auf die Ausreißer der beiden feinsten Level ist gerade noch so klein, daß kein von ihnen beeinflußter Koeffizient größer als der Threshold ist, so daß Algorith-

[25] Hierzu sei angemerkt, daß 80 Prozent des Rauschens kleiner als 0.3 war (siehe oben) und das Programm (unter der falschen Annahme einer Normalverteilung) die Standardabweichung zu etwa 0.224 bestimmte

[26] Bei der Sinus-Kurve verschwinden alle Wavelet-Koeffizienten ab dem vierten der elf Level, bei den anderen Datensätzen sind jeweils die Koeffizienten der beiden feinsten Level gleich Null. Bei „Doppler" und „NoisyBlocks" ist der neunte Level, der dort ebenfalls betrachtet wurde, zwar nicht leer, enthält aber nur zwei bzw. einen von Null verschiedenen Koeffizienten.

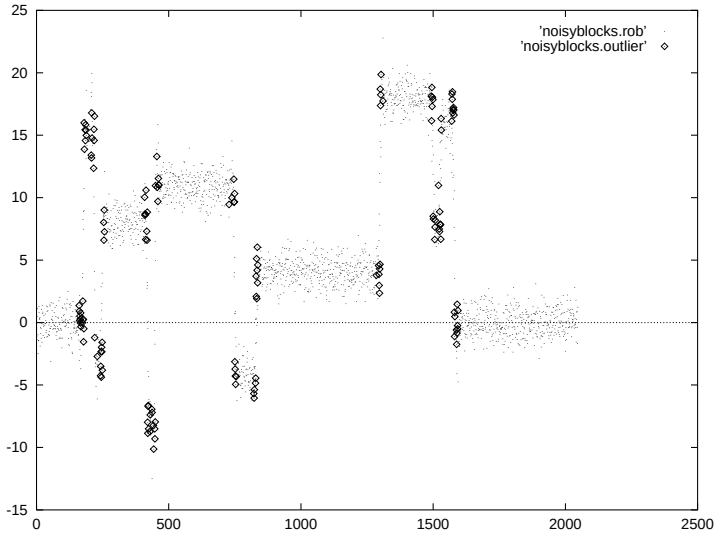

Abbildung 43: Die abgeänderten Daten bei Anwendung von Algorithmus 3.4 auf den Datensatz „NoisyBlocks”. Dick eingezeichnet die Punkte aus den Originaldaten, die abgeändert wurden.

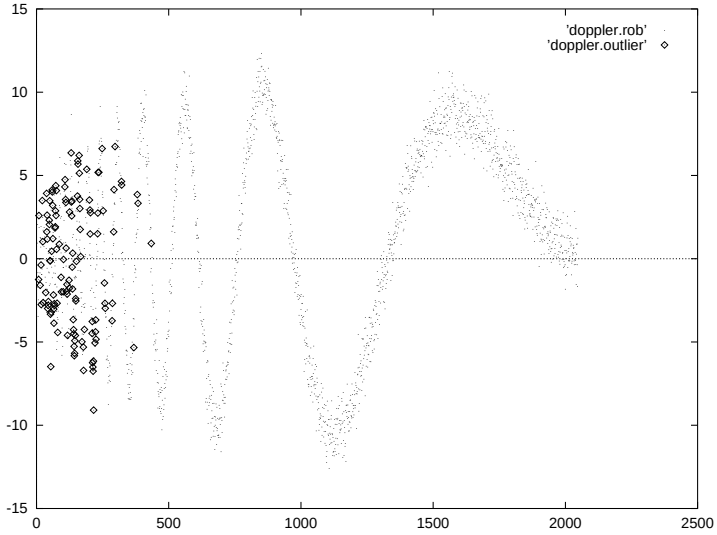

Abbildung 44: Die abgeänderten Daten bei Anwendung von Algorithmus 3.4 auf den Datensatz „Doppler”. Dick eingezeichnet die Punkte aus den Originaldaten, die abgeändert wurden.

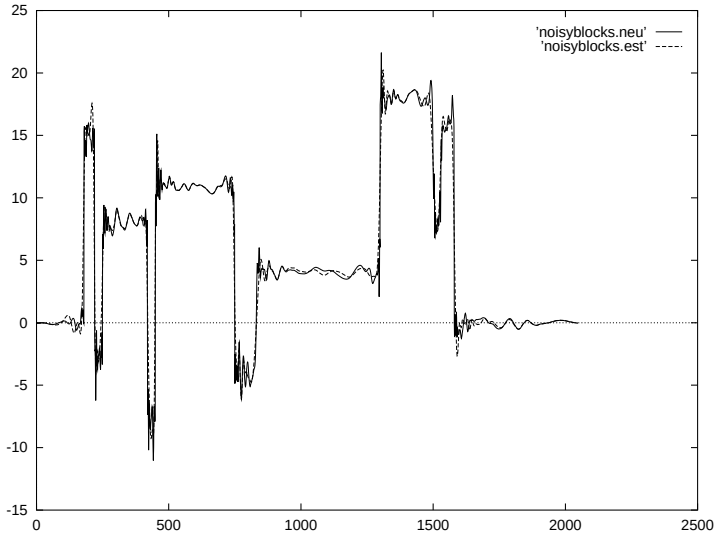

Abbildung 45: Vergleich der Schätzfunktionen von Algorithmus 3.4 und Hard-Thresholding bei dem Datensatz „NoisyBlocks"

mus 3.4 keine Möglichkeit hat, sie zu erkennen oder gar zu beseitigen. Auch Berücksichtigung gröberer Level durch eine größere Wahl von d_{max} zeigt keine Änderung. Sinnvoller scheint eine kleinere Wahl des Thresholds. Benutzt man beispielweise für die Schritte (2) bis (13) $\tilde{t} = t/1.4$, so werden die drei Ausreißer tatsächlich erkannt, und das Minimum der Schätzfunktion ist kleiner (siehe Abbildung 47).

Es scheint jedoch fraglich, ob dieses „Ad hoc"-Verfahren in der Praxis stets zu „besseren" Kurven führt, wie überhaupt die Bestimmung des Thresholds ein Problem darstellt: Zu große Wahl führt, wie wir eben gesehen haben, dazu, daß Ausreißer eventuell nicht erkannt werden. Ausprobieren verschiedener Thresholds zeigt jedoch, daß eine zu kleine Wahl ebenfalls zu Problemen führt. In diesem Fall führt die große Anzahl an Koeffizienten, die größer als der Threshold sind, und die Arbeitsweise des Verfahrens, bei der von links nach rechts Datenpunkte daraufhin untersucht werden, ob sie ein Ausreißer sind oder nicht, oft dazu, daß auch bereits die nicht durch Ausreißer kontaminierten Datenpunkte links neben Ausreißern abgeändert werden. Diese Änderungen sind dann allerdings noch durch die Ausreißer beeinflußt. Insgesamt dürfte die Bestimmung eines geeigneten Thresholds für dieses Verfahren sehr schwierig sein.

Während der Arbeit an diesem Algorithmus wurden eine Reihe von Modifikationen betrachtet, die allerdings alle zu keinen Vorteilen oder sogar zu Nachteilen führten:

1. In Schritt (6) werden die Datenpunkte $\hat{f}(k/n), \ldots, \hat{f}((k+1)/n)$ so gewählt, daß die l^2-Norm der Wavelet-Koeffizienten minimal ist. Hier kann man natürlich auch die l^1-Norm betrachten. Die Wahl der $\hat{f}(i/n)$ ist dann nicht eindeutig, kann aber so geschehen, daß $l+1$ Wavelet-Koeffizienten verschwinden. Daher können [27] zur Bestimmung der neuen Datenpunkte mehrere lineare Gleichungssysteme gelöst und die Wavelet-Koeffizienten der verschiedenen Lösungen in der l^1-Norm miteinander verglichen werden. Konkretisiert man diese Überlegungen bei der Haar-Basis und $l = 0$, so ergibt sich, daß $\hat{f}(k/n) = y_{k-1}$ oder

[27] Alternativ kann der Algorithmus von Barrodale verwendet werden, auf den in Abschnitt 3.4 kurz eingegangen wird.

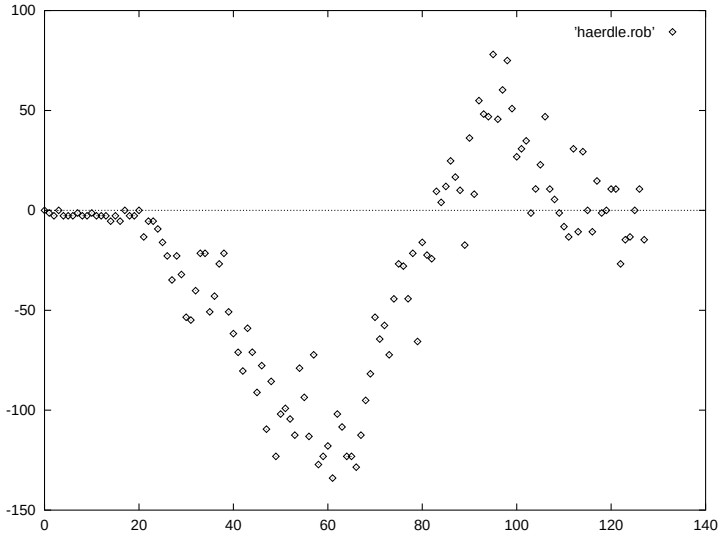

Abbildung 46: Die abgeänderten Daten bei Anwendung von Algorithmus 3.4 auf die Motorrad-Daten

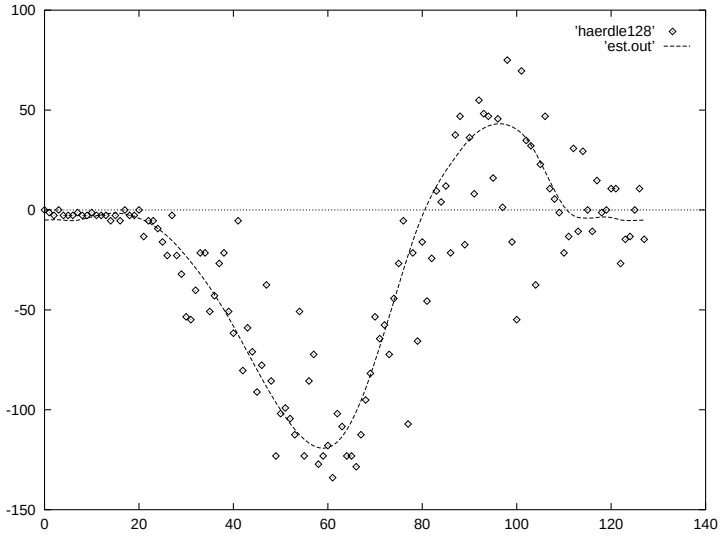

Abbildung 47: Die Schätzfunktion bei Anwendung von Algorithmus 3.4 auf die Motorrad-Daten mit kleinerem Threshold

$\hat{f}(k/n) = y_{k+1}$. Dies scheint auf den ersten Blick ein Vorteil gegenüber der Bestimmung in der l^2-Norm zu sein, wo $\hat{f}(k/n) = 0.5 \cdot (y_{k-1} + y_{k+1})$ gewählt wird und somit bei Unstetigkeiten abgeflacht werden, aber bei Benutzung von anderen Wavelet-Basen ist dieser minimale Vorteil nicht mehr zu erkennen, und es ergeben sich qualitativ keine Unterschiede.

2. Die Bestimmung von Varianz und Threshold in Schritt (1) wird eventuell durch Ausreißer beeinflußt. In manchen Fällen brachte daher ein Vertauschen der Schritte (1) und (2) Vorteile. Der Threshold ist dann nicht konstant, sondern wird mehrmals aufgrund korrigierter Daten neu berechnet. Dies führt jedoch in der Regel dazu, daß der Threshold beim zweiten Mal kleiner als beim ersten Mal gewählt wird, wodurch mehr Datenpunkte abgeändert werden und zu einem noch kleineren Threshold im dritten Schritt führen, usw. Insgesamt ergeben sich die oben skizzierten Probleme, die durch zu kleine Wahl des Thresholds entstehen.

3. Statt mit der Haar-Basis zu beginnen und schrittweise die Ordnung der Wavelet-Basis zu erhöhen, kann man auch in Schritt (2) direkt $m = m_{max}$ setzen und auf Schritt (13) verzichten. Dies führt jedoch zu Problemen, falls Ausreißer dicht nebeneinander liegen und so gleichzeitig Einfluß auf Wavelet-Koeffizienten nehmen. Abänderung eines Datenpunktes führt dann oft nicht zu kleinen Wavelet-Koeffizienten.

4. Die Bedingungen in (7) wurden auf verschiedene Arten und Weisen verschärft oder abgeschwächt, um zu erreichen, daß nur die Ausreißer abgeändert werden. Zum Beispiel wurde zusätzlich gefordert, daß Datenpunkte nur abgeändert werden durften, falls die Differenz zu den Originaldaten größer als $1.96 \cdot \sigma$ war. Ein anderes Mal wurden in Bedingung (7a) nur die Koeffizienten im Level $J - d$ betrachtet oder in (7b) Koeffizienten der Level $J - d_{max}$ bis J. Alle diese Ansätze wurden kombiniert mit verschiedenen Wahlen des Thresholds und ergaben sehr unterschiedliche Ergebnisse. Meistens konnte erreicht werden, daß das modifizierte Verfahren bei einem der Datensätze tatsächlich die Ausreißer fand, dafür versagte der gleiche Algorithmus bei einem der anderen Datensätze.

3.3 Ausreißererkennung mit einem laufenden Median

Im letzten Abschnitt wurden Daten abgeändert, um die l^2-Norm der Wavelet-Koeffizienten zu minimieren. Dabei ergab sich das Problem, daß jede Abänderung durch eventuell (noch) nicht eliminierte Ausreißer beeinflußt wurde. Dieses Problem wollen wir nun beseitigen, indem wir alle Datenpunkte, deren Abstand zu einem lokalen Median größer als $1.96 \cdot \sigma$ ist, als ausreißerverdächtig markieren und anschließend alle markierten Punkte auf einmal abändern. Die Abänderung geschieht dabei wieder so, daß die l^2-Norm der Wavelet-Koeffizienten minimal wird.

Wir formulieren direkt Algorithmus 3.5. Dazu sei $l \geq 0$ die Fensterbreite, die bei der Berechnung der lokalen Mediane benutzt wird, und $d \in \{0, \ldots, J-1\}$ so, daß die Koeffizienten der Level $J - d$ bis J zu minimieren sind. Es sei $\mathcal{W}^d : \mathbb{R}^n \to \mathbb{R}^{n-2^{d-1}}$ die Abbildung, die einem Vektor $y \in \mathbb{R}^n$ die Wavelet-Koeffizienten der Level $J - d$ bis J einer diskreten Wavelet-Transformation von y zuordnet. Algorithmus 3.5 sieht dann so aus:

Algorithmus 3.5.

(1) Schätze die Standardabweichung σ der Daten.

(2) Berechne $r_i := y_i - \mathrm{MED}\{y_j : 1 \leq j \leq n \wedge |i-j| \leq l\}$.

(3) Es sei $I := \{i \in \{1, \ldots, n\} : |r_i| \leq 1.96 \cdot \sigma\}$.

(4) Minimiere $||\mathcal{W}^d \tilde{y}||_2$ *unter allen* $\tilde{y} \in \mathbb{R}^n$ *mit* $\tilde{y}_i = y_i$ *für alle* $i \in I$.

(5) Wende auf $\tilde{y}$ *Wavelet Shrinkage an.*

Wir zeigen zunächst, was aus den Standard-Datensätzen wird, wenn man Algorithmus 3.5 auf sie anwendet. Das Ergebnis ist in den Abbildungen 48 bis 51 zu sehen. Es wurde stets das Daubechies-Wavelet der Ordnung 6 benutzt mit $j_0 = 3$. Bei den Motorrad-Daten waren $l = 3$ und $d = 2$, sonst $l = 5$ und $d = 3$. Das Verfahren änderte bei den Motorrad-Daten 28 (der 128) Datenpunkte ab, bei der Sinus-Kurve 315 (2048), bei dem Datensatz „NoisyBlocks" 122 (2048) und bei dem Datensatz „Doppler" 152 (2048).

Es scheint, als ob Algorithmus 3.5 die unterschiedlichen Probleme löst, die die vier Datensätze stellen. Bei den Crash-Daten ergibt sich eine glatte Kurve, deren Minimum nicht zu hoch liegt. Bei der verrauschten Sinus-Kurve erhält man fast eine echte Sinus-Kurve. Bei den „Noisy-Blocks" ist kein qualitativer Unterschied zu klassischem Wavelet-Shrinkage erkennbar, und zur Schätzkurve bei Anwendung auf den Datensatz „Doppler" ist das gleiche zu sagen wie im vorangegangenen Abschnitt bei Anwendung von Algorithmus 3.4.

Die Rechenzeit betrug im Falle der Crash-Daten weniger als eine Sekunde, bei der Sinus-Kurve vier Minuten und bei den beiden anderen Datensätzen ungefähr eine Minute. Da das Computerprogramm, das im Rahmen dieser Arbeit zu diesem Verfahren geschrieben wurde, aber noch in mehreren Punkten verbessert werden kann, ist davon auszugehen, daß man durch geschickte Programmierung kürzere Zeiten erreichen könnte. Insbesondere führt Schritt (4) (wie im letzten Abschnitt) zu einem „Least Squares"-Problem, bei dem die Matrix sehr dünn besetzt ist. Vielleicht gibt es Möglichkeiten, die Kleinste-Quadrate-Lösung unter Ausnutzung der speziellen Gestalt der Matrix besonders schnell zu berechnen. Dann könnte sich sogar ein deutlicher Geschwindigkeitszuwachs ergeben, aber das wäre ein Thema für eine eigenständige Arbeit.

Wir wollen nun ein paar Anmerkungen zu den Schritten (2), (3) und (4) machen und dabei insbesondere die Wahl der Parameter l und d diskutieren. In den Schritten (2) und (3) wird versucht, Ausreißerkandidaten zu ermitteln. Unter der Annahme, daß die lokale Variation eines Modells für die Daten in den „Fenstern" $[k-l, k+l]$ klein ist (insbesondere kleiner als die Ausreißer), stellt ein lokaler Median eine robuste Schätzung für den k-ten Datenpunkt dar. Der laufende Median wurde hauptsächlich verwendet, weil er einfach und schnell zu berechnen ist – im Prinzip kann hier aber jedes andere robuste Lagefunktional benutzt werden. In Betracht kommen auch Schätzungen durch eine lokale robuste lineare Regression (zum Beispiel „Least Median of Squares"), die den Datenpunkt in Fenstern, wo die Kurve sehr steil verläuft, besser schätzen, allerdings auch in der Nähe von Unstetigkeitsstellen zu Problemen führen.

Wichtig ist in jedem Falle, daß alle Ausreißer in einem Datensatz erkannt werden, was bei der Wahl von l und eventuell bei der Bestimmung des Ausreißerniveaus in Schritt (3) unbedingt berücksichtigt werden muß. Zum Beispiel können m aufeinanderfolgende Ausreißer nur identifiziert werden, wenn zumindestens $l \geq m$ ist. Die größtmögliche Anzahl von aufeinanderfolgenden Ausreißern stellt somit eine untere Schranke für l dar, die jedoch, wie wir bereits am

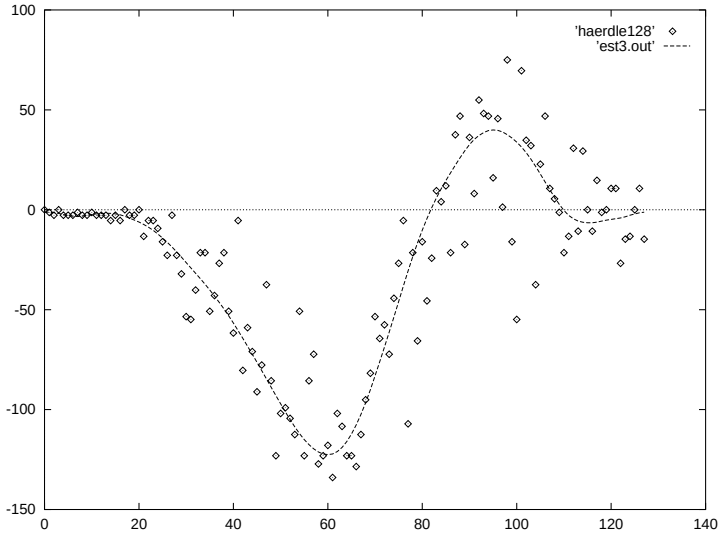

Abbildung 48: Anwendung von Algorithmus 3.5 auf die Motorrad-Daten

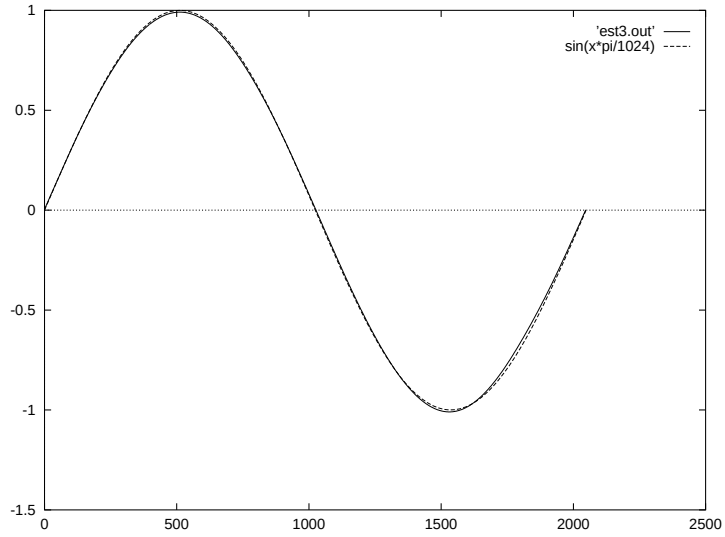

Abbildung 49: Anwendung von Algorithmus 3.5 auf die Cauchy-verrauschte Sinus-Kurve – zum Vergleich eine echte Sinus-Kurve

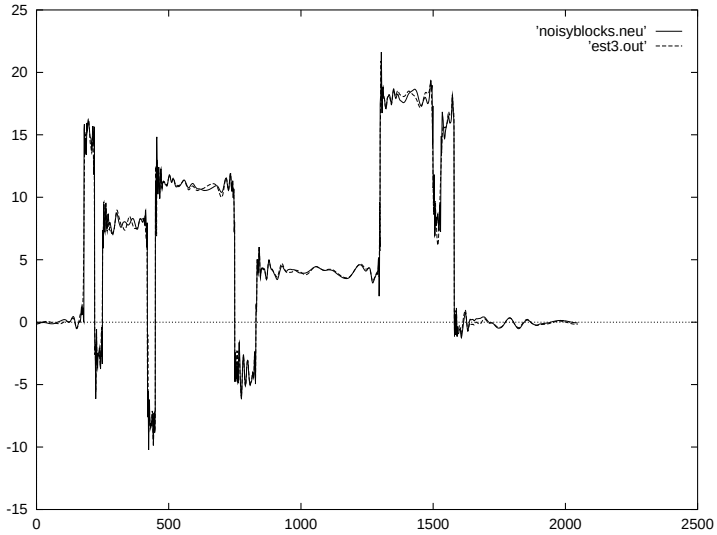

Abbildung 50: Anwendung von Algorithmus 3.5 auf die „NoisyBlocks” – zum Vergleich klassisches Wavelet-Thresholding

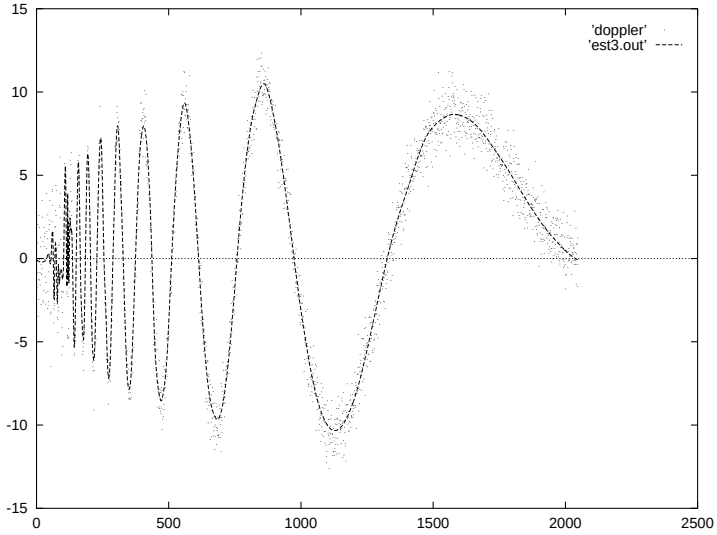

Abbildung 51: Anwendung von Algorithmus 3.5 auf den Datensatz „Doppler”

Ende des zweiten Kapitels gesehen haben, nur mit Vorwissen über den Datensatz angegeben werden kann. Dort hatten wir einen Datensatz aus der Spektroskopie betrachtet, der es erforderlich machte, l sehr klein zu wählen, damit Peaks nicht fälschlicherweise als Ausreißer eliminiert wurden. Wir werden am Ende dieses Abschnittes einen anderen Datensatz sehen, wo bis zu 150 aufeinanderfolgende Beobachtungen als Ausreißer interpretiert werden sollen.

Auf der anderen Seite führt eine zu große Wahl von l unter Umständen zu Problemen, wenn die Annahme, daß ein Modell für die Daten in jedem Fenster $[k-l, k+l]$ nahezu konstant ist, nicht adäquat ist. Man kann dann Fälle konstruieren, in denen der laufende Median Ausreißer übersieht, aber in der Regel ist die einzige Folge, daß lokale Extrema und benachbarte Punkte als Ausreißer deklariert werden, was aber nicht besonders tragisch ist, da im Eliminationsschritt die Ausreißerkandidaten nicht durch die (dort schlechten) Schätzungen des laufenden Medians substituiert werden, sondern durch die Wavelet-Schätzung des vierten Schrittes.

Als nächstes beschäftigen wir uns mit dem vierten Schritt. Es erstaunt vielleicht, daß hier die Wavelet-Koeffizienten einer Diskreten Wavelet-Transformation benutzt werden, nachdem im letzten Abschnitt eine ähnliche Anwendung mit den Koeffizienten der Stationären Wavelet-Transformation vorgestellt wurde. Das Problem besteht darin, daß es unbedingt erforderlich ist, daß Wavelet-Koeffizienten mehrerer Level gleichzeitig minimiert werden müssen. Das folgende Beispiel zeigt, was passiert, wenn man nur die feinsten Koeffizienten einer Stationären Wavelet-Transformation minimiert.

Abbildung 52 zeigt einen Ausschnitt aus dem Ballon-Datensatz, der am Ende dieses Abschnittes untersucht wird. Aus Gründen, auf die wir hier noch nicht eingehen wollen, ist die Kurve im Intervall $[5350; 5500]$ stark durch Ausreißer gestört. Die Parameter wurden so justiert, daß die Beobachtungen $y_{5374}, \ldots, y_{5466}$ alle als Ausreißer markiert wurden. Anschließend wurden die Ausreißerkandidaten so geschätzt, daß die l^2-Norm der Koeffizienten einer Stationären Wavelet-Transformation bzgl. des Daubechies-Wavelets der Ordnung 2 minimiert wurde. Das Ergebnis ist in Abbildung 53 zu sehen: Die Datenpunkte wurden derart abgeändert, daß sie größtenteils auf einem Polynom zweiten Grades liegen. Dadurch daß an den Rändern die nicht abgeänderten Datenpunkte stark fallen bzw. steigen, ist auch die Steigung des Polynoms an den Rändern betragsmäßig sehr groß. Die Folge ist, daß die „robustifizierten" Daten noch schlechter als die Ausgangsdaten sind.

Indem man auch Koeffizienten gröberer Level minimiert, verschwinden solche Probleme. Da die Translationsinvarianz der SWT für diese Zwecke keine Vorteile bringt, sondern nur einen unnötig hohen Rechenaufwand produziert, benutzen wir dabei die Koeffizienten der Diskreten Wavelet-Transformation. Es ist jedoch noch zu klären, wie der Parameter d zu bestimmen ist, der angibt, wieviele Level minimiert werden sollen.

Zunächst einmal muß d zumindestens so groß gewählt sein, daß die Matrix des linearen Regressionsproblems, auf das der vierte Schritt führt, linear unabhängige Spalten hat. Diese Matrix erhält man offenbar, indem man in $\mathcal{W}^d$ für jedes $i \in I$ die i-te Spalte streicht (vgl. dazu auch den Beweis von Satz 3.3). Benutzt man zum Beispiel die Haar-Basis und findet in den Schritten (2) und (3) zwei Ausreißerkandidaten an den Positionen 51 und 52, so darf man nicht $d = 0$ wählen. Andernfalls ergibt sich eine Matrix, die aus zwei Spalten und $n/2$ Zeilen besteht, von denen eine die Filter-Koeffizienten der Haar-Basis ($2^{-1/2}$ und $-2^{-1/2}$) und die andere nur Nullen enthält. Allgemeiner kann man sich leicht überlegen, daß d bei Verwendung der Haar-Basis mindestens so groß gewählt werden muß, daß für jedes $j \in \{0, \ldots, 2^{n-d-1} - 1\}$ höchstens die Hälfte der Beobachtungen $y_{j2^{d+1}+1}, \ldots, y_{(j+1)2^{d+1}}$ Ausreißerkandidaten sein dürfen. Diese Überlegungen sind jedoch mit zunehmender Ordnung der Wavelet-Basis wegen der ebenfalls größer werden-

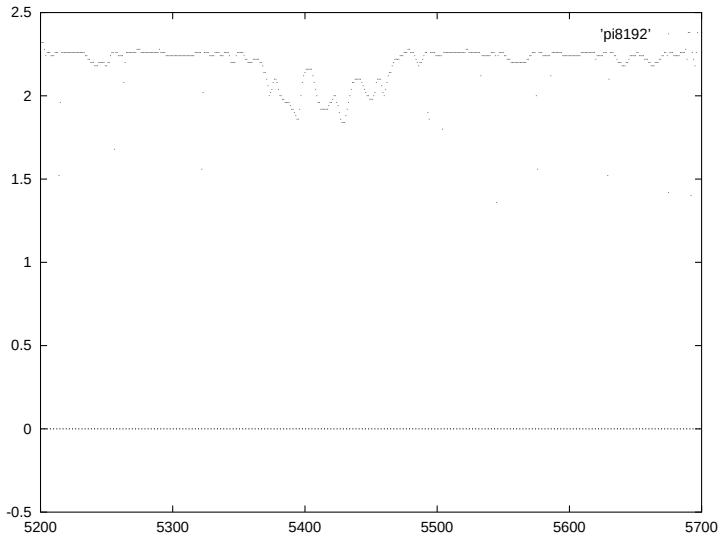

Abbildung 52: Ein Ausschnitt aus dem Ballon-Datensatz

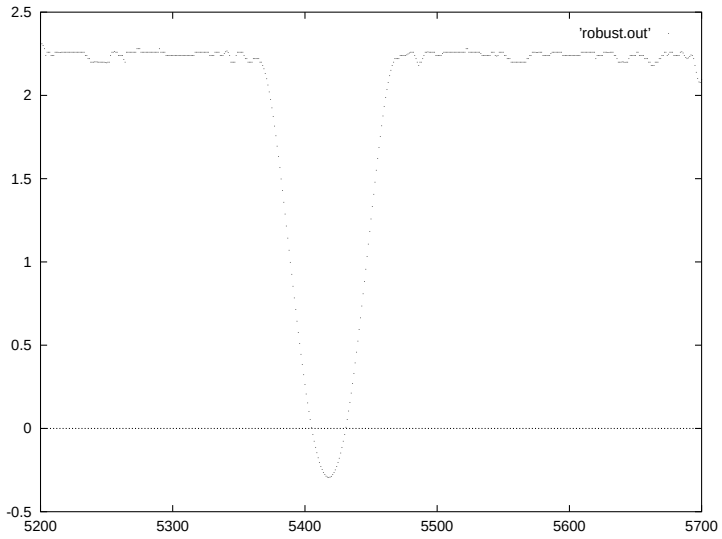

Abbildung 53: Die „robuste” Schätzung der Ausreißerkandidaten

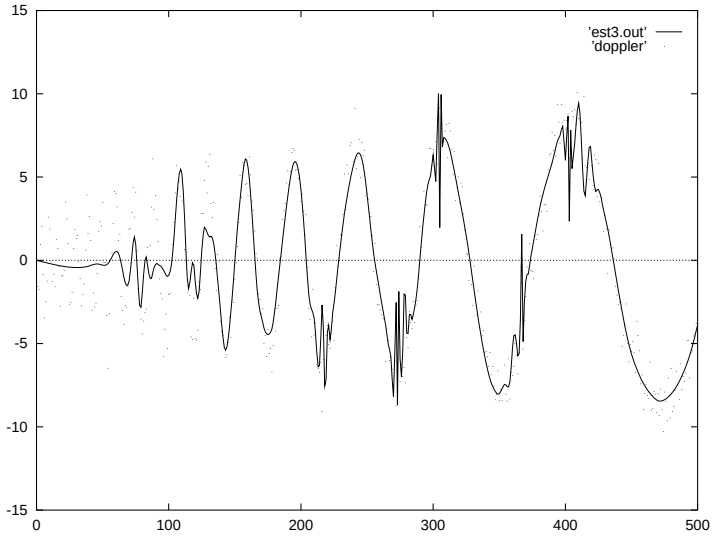

Abbildung 54: Die Schätzkurve, die sich bei Anwendung von Algorithmus 3.5 auf den Datensatz „Doppler" ergibt, mit $d = 5$. (Ausschnitt)

den Zahl an Filterkoeffizienten von immer geringerem Interesse.

Bislang haben wir uns überlegt, welche Folgen eine zu kleine Wahl von d hat, nun wollen wir umgekehrt sehen, welche Konsequenzen eine zu große Wahl hat. Es ist klar, daß das Minimieren eines Wavelet-Koeffizienten nur Sinn macht, solange die Datenpunkte, die Einfluß auf den Koeffizienten haben, auf einem Polynom liegen, dessen Grad kleiner als die Ordnung der Wavelet-Basis ist. Diese Forderung ist für Koeffizienten von groben Leveln in der Regel nicht erfüllt. Wir wenden Algorithmus 3.5 noch einmal auf den Datensatz „Doppler" an, dieses Mal wählen wir jedoch $d = 5$ statt $d = 3$. Die Folgen sind verheerend: In der Schätzkurve lassen sich an einigen Stellen unmotivierte Fine-Scale-Phänomene beobachten (Abbildung 54), die auf eine sehr schlechte Abänderung der Datenpunkte im vierten Schritt von Verfahren 3.5 zurückzuführen sind, wie Abbildung 55 demonstriert.

Eine genaue Betrachtung der abgeänderten Beobachtungen läßt jedoch auch vermuten, was in Satz 3.4 bewiesen wird: Die abgeänderten Datenpunkte liegen alle auf der Kurve, die die l^2-Norm der Nicht-Ausreißer-Residuen minimiert, unter allen Kurven, deren Wavelet-Koeffizienten der Level $J - d$ bis J Null sind. Mit Hilfe dieser Aussage werden wir weiter unten ein Verfahren vorstellen, mit dem eine Bestimmung von d möglich ist.

Wir wollen den Satz direkt etwas allgemeiner definieren und bezeichnen daher für zwei Teilmengen $A, I \subset \{1, \ldots, n\}$ mit $\mathcal{W}_I^A$ die Matrix, die aus den Einträgen $\{w_{ij}\}_{i \in A, j \in I}$ besteht. Man extrahiert also von $\mathcal{W}$ für jedes $i \in A$ die i-te Zeile und für jedes $j \in I$ die j-te Spalte. Mit I^C und A^C seien die Komplemente von I und A in $\{1, \ldots, n\}$ bezeichnet. Es gelten die beiden folgenden Lemmata:

Lemma 3.1. *Für $A, I \subset \{1, \ldots, n\}$ gilt:*

a) $(\mathcal{W}_{I^C}^A)^T \mathcal{W}_I^A = -(\mathcal{W}_{I^C}^{A^C})^T \mathcal{W}_I^{A^C}$

b) $(\mathcal{W}_{I^C}^A)^T \mathcal{W}_{I^C}^A = \mathrm{Id}_{I^C} - (\mathcal{W}_{I^C}^{A^C})^T \mathcal{W}_{I^C}^{A^C}$

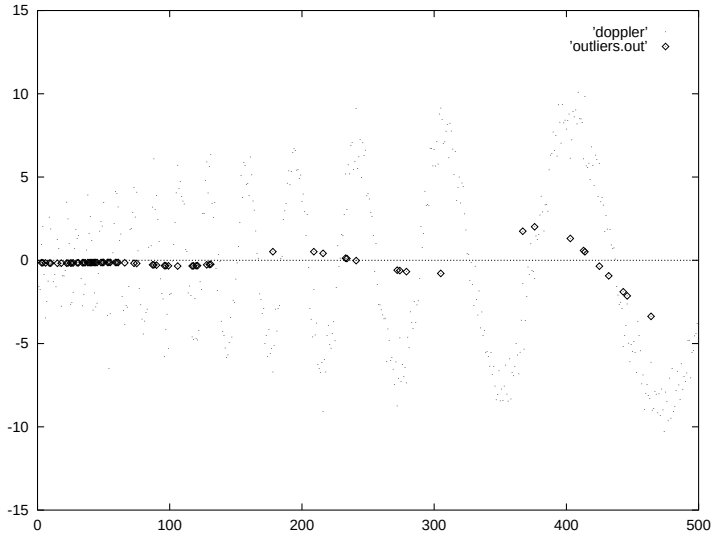

Abbildung 55: Die abgeänderten Daten, die sich bei Anwendung von Algorithmus 3.5 auf den Datensatz „Doppler" ergeben, mit $d = 5$. (Ausschnitt)

Beweis. Der Beweis ist eigentlich klar und folgt direkt aus der Orthonormalität der Matrix $\mathcal{W}$. Zum Beispiel ist für $j_1 \in I$ und $j_2 \in I^C$ das Vektor-Produkt

$$\begin{aligned}(\mathcal{W}_{j_2}^A)^T \mathcal{W}_{j_1}^A &= \sum_{i \in A} w_{i,j_2} w_{i,j_1} \\ &= \sum_{i \in \{1,\ldots,n\}} w_{i,j_2} w_{i,j_1} - \sum_{i \in A^C} w_{i,j_2} w_{i,j_1} \\ &= -\sum_{i \in A^C} w_{i,j_2} w_{i,j_1} \\ &= (\mathcal{W}_{j_2}^{A^C})^T \mathcal{W}_{j_1}^{A^C}.\end{aligned}$$

□

Lemma 3.2. *Für eine $k \times l$-Matrix B, für die $\mathrm{Id}_l - B^T B$ und $\mathrm{Id}_k - BB^T$ invertierbar sind, gilt:*

$$(\mathrm{Id}_l - BTB)^{-1} B^T = B^T (\mathrm{Id}_k - BB^T)^{-1}.$$

Beweis. Der Beweis ergibt sich, indem man in der Gleichung

$$B^T(\mathrm{Id}_k - BB^T) = (\mathrm{Id}_l - B^T B)B^T$$

links und rechts mit der passenden Inversen multipliziert. □

Bevor wir nun endgültig den Satz formulieren, führen wir noch eine neue Bezeichnung ein: Für einen Vektor $v \in \mathbb{R}^n$ und $I \subset \{1, \ldots, n\}$ sei v_I der Vektor, der nur aus den Einträgen $\{v_j\}_{j \in I}$ besteht.

Satz 3.4. *Es seien $y \in \mathbb{R}^n$ und $A, I \subset \{1, \ldots, n\}$, so daß die Matrizen $(\mathcal{W}_I^{A^C})^T$ und $\mathcal{W}_{IC}^A$ beide vollen Spaltenrang haben. Weiter seien $w \in \mathbb{R}^{|A|}$ und $\tilde{y} \in \mathbb{R}^{|I^C|}$ die Lösungen der Minimierungsprobleme*

$$||y_I - (\mathcal{W}_I^{A^C})^T w||_2 = \textit{min} \tag{3.13}$$

und

$$||\mathcal{W}_I^A y_I + \mathcal{W}_{I^C}^A \tilde{y}||_2 = \textit{min}. \tag{3.14}$$

Dann gilt

$$(\mathcal{W}_{I^C}^{A^C})^T w = \tilde{y}. \tag{3.15}$$

Beweis. Aus der Linearen Regression ist bekannt, daß das Least-Squares-Problem

$$||Ax - b||_2 = \min$$

für eine Matrix A mit vollem Spaltenrang die eindeutig bestimmte Lösung

$$x = (A^T A)^{-1} A^T b$$

besitzt. Daher gilt für beliebiges $y \in \mathbb{R}^n$, daß

$$w = \left(\mathcal{W}_I^{A^C} (\mathcal{W}_I^{A^C})^T\right)^{-1} \mathcal{W}_I^{A^C} y_I$$

und

$$\tilde{y} = \left((\mathcal{W}_{I^C}^A)^T \mathcal{W}_{I^C}^A\right)^{-1} (\mathcal{W}_{I^C}^A)^T (-\mathcal{W}_I^A y_I),$$

was nach Anwenden von Bemerkung 3.1a) zu

$$\tilde{y} = \left((\mathcal{W}_{I^C}^A)^T \mathcal{W}_{I^C}^A\right)^{-1} (\mathcal{W}_{I^C}^{A^C})^T \mathcal{W}_I^{A^C} y_I$$

führt. Daher ist zum Beweis von (3.15) nur noch zu zeigen, daß

$$(\mathcal{W}_{I^C}^{A^C})^T \left(\mathcal{W}_I^{A^C} (\mathcal{W}_I^{A^C})^T\right)^{-1} = \left((\mathcal{W}_{I^C}^A)^T \mathcal{W}_{I^C}^A\right)^{-1} (\mathcal{W}_{I^C}^{A^C})^T.$$

Diese Aussage ergibt sich unmittelbar aus den Bemerkungen 3.1b) und 3.2. □

Setzt man in obigem Satz $A = 2^{J-d-1}, \ldots, n$, so ist das Minimierungsproblem (3.14) exakt das Minimierungsproblem aus dem vierten Schritt von Algorithmus 3.5. Wie bereits angesprochen liefert der Satz also eine Alternativmethode zur Berechnung der abgeänderten Daten: Approximiere die Nicht-Ausreißer in den Daten durch die Wavelet-Vektoren der Level kleiner als $J-d$ in der l^2-Norm und wähle die abgeänderten Daten als die Zwischenwerte des Wavelet-Schätzers an den Ausreißer-Stellen. Bei großem d und relativ vielen Ausreißerkandidaten ist dieser Algorithmus schneller als der vierte Schritt von Algorithmus 3.5.

Doch viel interessanter als Laufzeitbetrachtungen ist, daß hierdurch eine Möglichkeit zur Bestimmung von d gegeben ist. Der Parameter m, der in dem folgenden Algorithmus vorkommt, kann wie immer entweder durch Simulation oder theoretische Überlegungen bestimmt werden. Man beachte aber, daß m hier nicht von n, sondern von $|I|$ abhängt.

Algorithmus 3.6.

(1) Starte mit $d := J - 1$.

(2) Approximiere die Nicht-Ausreißer mit Wavelets der Level $1, \ldots, J-d-1$ *durch Minimieren der Residuen in der* l^2*-Norm.*

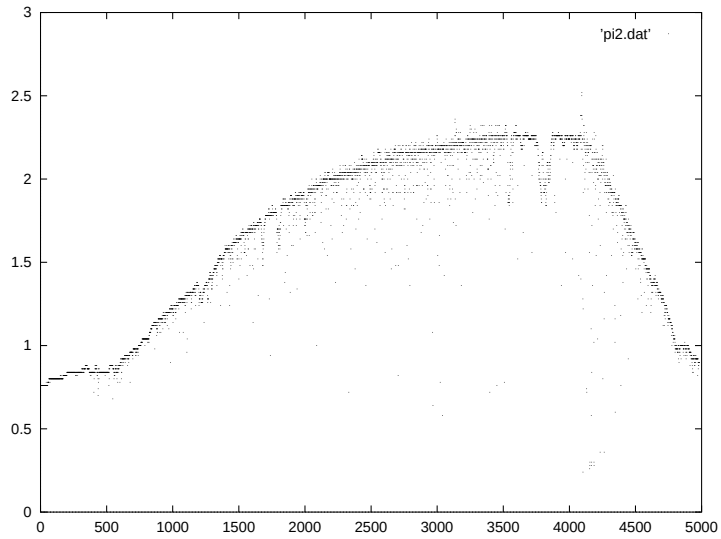

Abbildung 56: Der Ballon-Datensatz

(3) Falls die maximale Runlänge der Residuen kleiner oder gleich m ist, fertig.

(4) Sonst setze $d := d - 1$ und weiter bei (2).

Der Algorithmus wurde auf die verschiedenen Datensätze angewandt, wobei für m stets das 0.5-Quantil benutzt wurde, und lieferte für die Crash-Daten $d = 3$, für die „NoisyBlocks" und den Datensatz „Doppler" $d = 2$ und für die verrauschte Sinus-Kurve $d = 8$. Die resultierenden Schätzkurven waren mit bloßem Auge nicht von den Kurven der Abbildungen 48 bis 51 zu unterscheiden.

Wir wollen nun Algorithmus 3.5 auf einen weiteren Datensatz anwenden, der auch schon von P. L. Davies in [9] und [10] analysiert worden ist. Ein Ballon war mit einer Vorrichtung zur Messung der Sonneneinstrahlung ausgerüstet. Abbildung 56 zeigt den Datensatz, der aus 4984 Beobachtungen des Meßballons besteht. Während des Fluges drehte sich der Ballon, so daß zeitweilig die Seile, an denen die Meßapparatur befestigt war, die direkte Sonneneinstrahlung unterbrachen und Ausreißer in den Daten verursachten. Bis zu 150 aufeinanderfolgende Beobachtungen sind durch die abgeschnittene Sonneneinstrahlung gestört.

Um den Datensatz überhaupt mit Wavelets analysieren zu können, wurden zunächst 3208 „Beobachtungen" ergänzt, so daß die Anzahl der Beobachtungen 8192, also eine Zweierpotenz ist. Zu diesem Zweck wurden der linke und der rechte Datenpunkt jeweils 1604mal auf der linken bzw. rechten Seite wiederholt. Zugunsten einer übersichtlicheren Darstellung wurde aber in den folgenden Abbildungen darauf verzichtet, die ergänzten Punkte mitabzudrucken.

Ein weiteres Problem bei der Anwendung unseres Verfahrens zeigt sich direkt im ersten Schritt, also bei der Schätzung der Standardabweichung. Schaut man sich die Beobachtungen näher an, stellt man fest, daß auch das nicht durch Ausreißer kontaminierte Rauschen nicht unabhängig ist. Vielmehr sind meistens mehrere aufeinanderfolgende (stellenweise 70 und mehr) Beobachtungen identisch. Dies führt dazu, daß die Standardabweichung mit 0.013 sehr niedrig geschätzt wird, wenn man wieder den (skalierten) Median der Koeffizienten des feinsten Levels, die Einfluß auf die 4984 Beobachtungen haben, zugrunde legt, und im zweiten Schritt 2029 Ausreißer erkannt werden. Wir benutzen daher für diese Zwecke nicht die Koeffizienten des feinsten

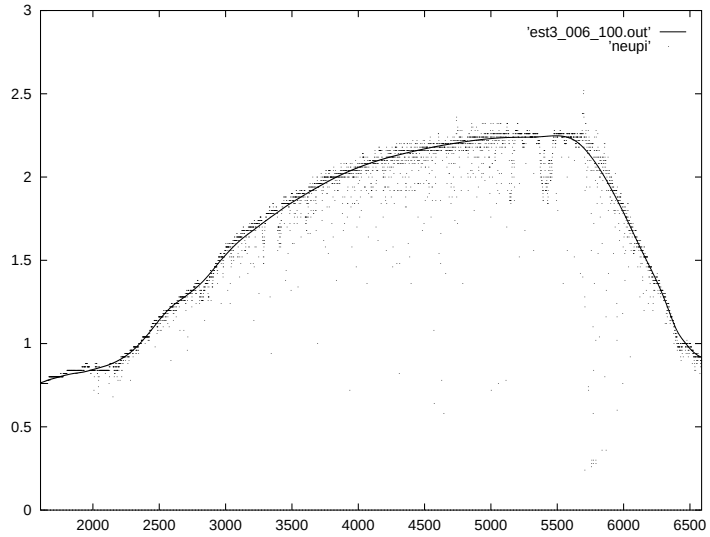

Abbildung 57: Anwendung von Algorithmus 3.5 auf die Ballon-Daten mit $j_0 = 3$ und $\sigma = 0.06$

(J-ten) Levels, sondern einmal die des $J-1$-ten Levels und ein anderes Mal die des $J-2$-ten Levels und kommen so auf die geschätzten Standardabweichungen $\hat{\sigma} = 0.032$ und $\hat{\sigma} = 0.060$.

Um die 150 aufeinanderfolgenden Ausreißer zu eliminieren, wurde $l = 150$ gewählt. Bei der Bestimmung von d mit Algorithmus 3.6 wurde für m nicht das 0.5-Quantil gewählt, da wie oben schon festgestellt, das Rauschen nicht unabhängig verteilt ist. Statt dessen wurde $m = 100$ benutzt und lieferte $d = 7$ bzgl. der Daubechies-Basis der Ordnung 6. Die Abbildungen 57 und 58 zeigen die Schätzfunktionen bezüglich der beiden Standardabweichungen und $j_0 = 3$, in den Abbildungen 59 und 60 sind die Ergebnisse für $j_0 = 6$ dargestellt. Die Schätzkurven sind alle sehr zufriedenstellend.

3.4 Robuste Bestimmung von Wavelet-Koeffizienten

In den vergangenen Abschnitten wurde versucht, Ausreißer in Datensätzen zu eliminieren. Das Grundprinzip war stets, durch Abänderungen von ausreißerverdächtigen Beobachtungen einen neuen Datensatz zu generieren und auf diesen Wavelet-Shrinkage anzuwenden. Die Philosophie dieses Abschnittes ist grundlegend anders: Wir versuchen, die Wavelet-Koeffizienten der Schätzfunktion direkt robust zu bestimmen, indem wir die l^1-Norm minimieren.

Dazu bezeichnen wir zunächst mit $\{w_i\}_{i\in\{1,\dots,n\}}$ die Orthonormalbasis des $\mathbb{R}^n$, die durch die Diskrete Wavelet-Transformation bzgl. einer Daubechies-Basis des $L^2(\mathbb{R})$ induziert wird, d.h. es gilt $w_i = \mathcal{W}^T e_i$, wobei e_i der i-te Einheitsvektor des $\mathbb{R}^n$ sei. Die w_i seien so durchnumeriert, daß $w_{n/2+1}, \dots, w_n$ mit den Waveletkoeffizienten des Levels J korrespondieren, $w_{n/4+1}, \dots, w_{n/2}$ mit den Koeffizienten des Levels $J-1$ usw.

Für eine beliebige (nicht notwendigerweise nicht-leere) Teilmenge $I \subset \{1,\dots,n\}$ sei W^I die quadratische Matrix, deren i-te Spalte für $i \in I$ der Vektor w_i und sonst eine Nullspalte ist.

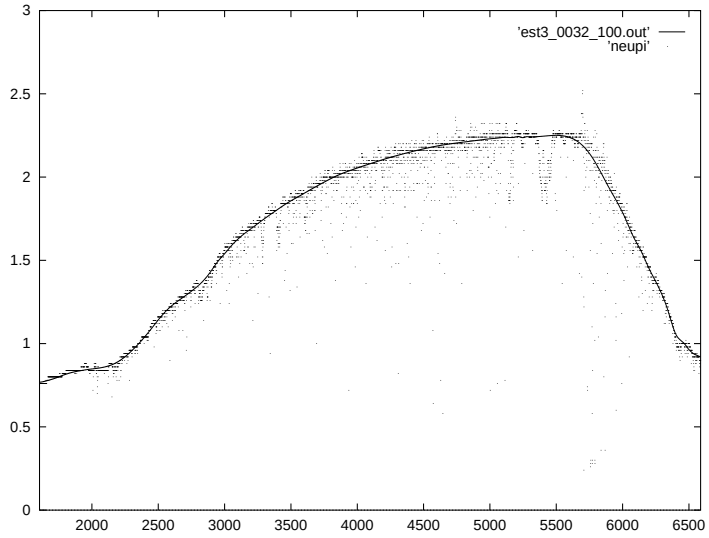

Abbildung 58: Anwendung von Algorithmus 3.5 auf die Ballon-Daten mit $j_0 = 3$ und $\sigma = 0.032$

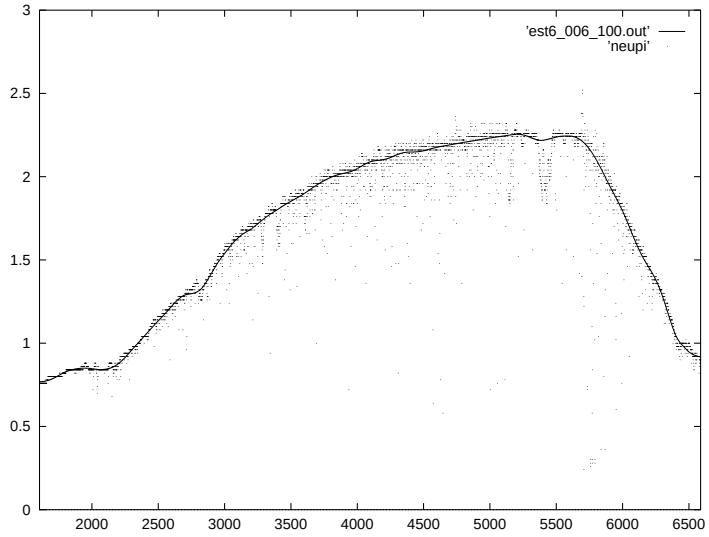

Abbildung 59: Anwendung von Algorithmus 3.5 auf die Ballon-Daten mit $j_0 = 6$ und $\sigma = 0.06$

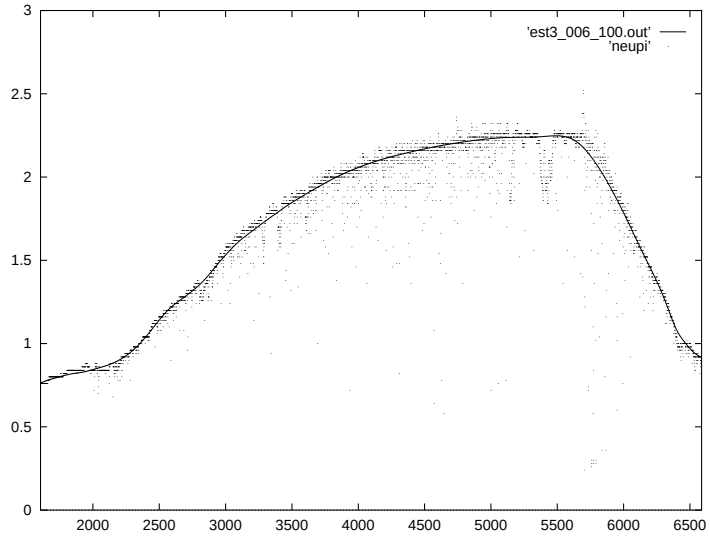

Abbildung 60: Anwendung von Algorithmus 3.5 auf die Ballon-Daten mit $j_0 = 6$ und $\sigma = 0.032$

[28] Wir betrachten dann das Minimierungsproblem

$$||y - W^I a||_1 = \min, \tag{3.16}$$

Eine Lösung a^I von (3.16) ist im allgemeinen nicht eindeutig bestimmt. Um ein a^I in der Praxis zu berechnen, bietet sich der schnelle Algorithmus von I. Barrodale und F. D. K. Roberts an (siehe [2]). Dabei wird das folgende Alternativproblem betrachtet:

$$\text{Minimiere } \sum_{i=1}^{n}(u_i + v_i)$$

$$\text{bezüglich } y_i = \sum_{j \in I}(b_j - c_j)(w_j)_i + u_i - v_i$$

$$\text{und } b_j, c_j, u_i, v_i \geq 0.$$

Dieses Problem läßt sich mit dem Simplex-Verfahren lösen, und durch $a^I = b - c$ ist dann eine Lösung von (3.16) definiert.

Wir interessieren uns natürlich vor allem für $g^I := W^I a^I$. Um zu entscheiden, ob ein so berechnetes g^I ein adäquates Modell für unsere Daten darstellt, betrachten wir wie in Abschnitt 3.1 die maximale Runlänge der Vorzeichen der Residuen $y - g^I$ und verlangen, daß diese kleiner als ein zuvor (theoretisch oder durch Simulation) berechneter Wert l ist. Um nun eine Schätzkurve für $\hat{f}(i/n)$ anzugeben, wählen wir unter allen g^I, die die Run-Bedingung erfüllen, eines mit minimalem $|I|$ aus.

Leider läßt der Rechenaufwand eine direkte Umsetzung dieses Verfahrens in die Praxis nicht zu, denn alleine, um zu überprüfen, daß für keine zwei-elementige Teilmenge I die maximale Runlänge der Vorzeichen der Residuen $y - g^I$ zu groß ist, müßte man bereits bei einem Datensatz

[28] In der Praxis werden die Nullspalten selbstverständlich gestrichen – die weiter unten folgende Definition von Algorithmus 3.7 gewinnt jedoch ein wenig an Übersichtlichkeit, wenn in dem nun folgenden Minimierungsproblem der i-te Eintrag von a^I der i-te Waveletkoeffizient von g^I ist.

der Länge 2048 mehr als zwei Millionenmal das Minimierungsproblem (3.16) lösen. Wir verfolgen daher einen Ansatz, der in jedem Fall zu einem g^I führt, für den die Residuen die gestellte Run-Bedingung erfüllen, aber im allgemeinen nicht mit minimalem $|I|$.

Unser Verfahren, das im Anschluß vorgestellt wird, besteht aus zwei Teilen: Im ersten Teil wird zunächst überprüft, ob der Nullvektor die Run-Bedingung erfüllt. Falls nicht, bestimmen wir das kleinste $j \in \{0, \ldots, J\}$, für das ein $g^{\{1,\ldots,2^j\}}$ die Run-Bedingung erfüllt. Würden wir jetzt abbrechen, erhielten wir wieder die Probleme des zweiten Kapitels, die sich zum Beispiel bei dem „Doppler"-Datensatz äußern würden. Wir versuchen daher im zweiten Teil möglichst viele Koeffizienten aus $I := \{1, \ldots, 2^j\}$ wegzunehmen, ohne daß die Run-Bedingung verletzt wird.

Algorithmus 3.7.

(1) Überprüfe, ob die maximale Runlänge der Vorzeichen von y kleiner oder gleich l ist.

(2) Falls ja, wähle $\hat{f} := 0$, fertig.

(3) Starte mit $j := 0$.

(4) Betrachte $I := \{1, \ldots, 2^j\}$.

(5) Bestimme a^I und g^I.

(6) Überprüfe, ob die maximale Runlänge der Vorzeichen von $y - g^I$ kleiner oder gleich l ist.

(7) Falls nicht, setze $j := j + 1$, weiter bei (4).

(8) Bestimme ein $i \in I$, so daß $|a_i|$ minimal.

(9) Bestimme $a^{I\setminus\{i\}}$ und $g^{I\setminus\{i\}}$.

(10) Überprüfe, ob die maximale Runlänge der Vorzeichen von $y - g^{I\setminus\{i\}}$ kleiner oder gleich l ist.

(11) Falls ja, setze $I := I \setminus \{i\}$, weiter bei (8).

(12) Wähle $\hat{f} := g_I$.

Die Abbildungen 61 bis 65 zeigen die Schätzkurven, die Algorithmus 3.7 liefert, wenn man ihn auf die verschiedenen Datensätze anwendet. Es wurde stets die Daubechies-Basis der Ordnung 6 verwendet. Außerdem war bei den Crash-Daten $l = 5$, bei den Ballon-Daten $l = 200$ und sonst $l = 15$. Der dazu notwendige Rechenaufwand variierte sehr stark mit der Größe des Datensatzes und insbesondere der Komplexität des Modells, gemessen in dem höchsten Level, der zur Minimierung der Runlänge in den Schritten (3) bis (7) benötigt wird. Er betrug bei den Crash-Daten und der Sinus-Kurve ein bis zwei Sekunden, bei den „NoisyBlocks", dem Datensatz „Doppler" und den Ballon-Daten 25 bis 30 Minuten.

Wir wollen nun auf die Wahl von l eingehen. In den betrachteten Beispielen wurde für l stets die nach Vergleichen der Schätzfunktionen für verschiedene l und subjektivem Ermessen beste Wahl getroffen. Bei den Crash-Daten entspricht diese Wahl dem 0.05-Quantil der Verteilung der maximalen Runlänge bei Datensätzen der Länge 128. Bei der Sinus-Kurve, den „NoisyBlocks"

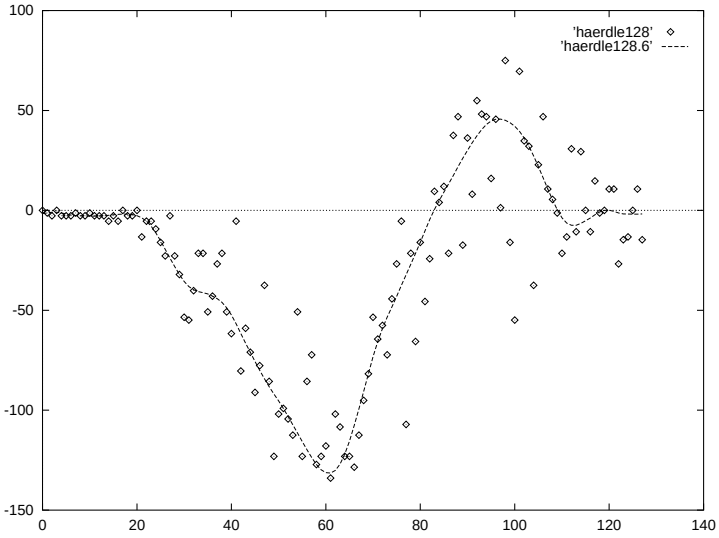

Abbildung 61: Anwendung von Algorithmus 3.7 auf die Crash-Daten mit $l = 5$

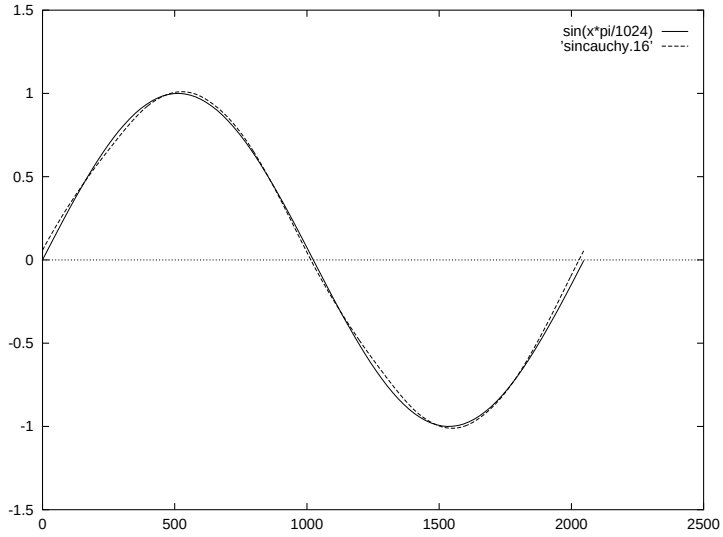

Abbildung 62: Anwendung von Algorithmus 3.7 auf die verrauschte Sinus-Kurve mit $l = 15$

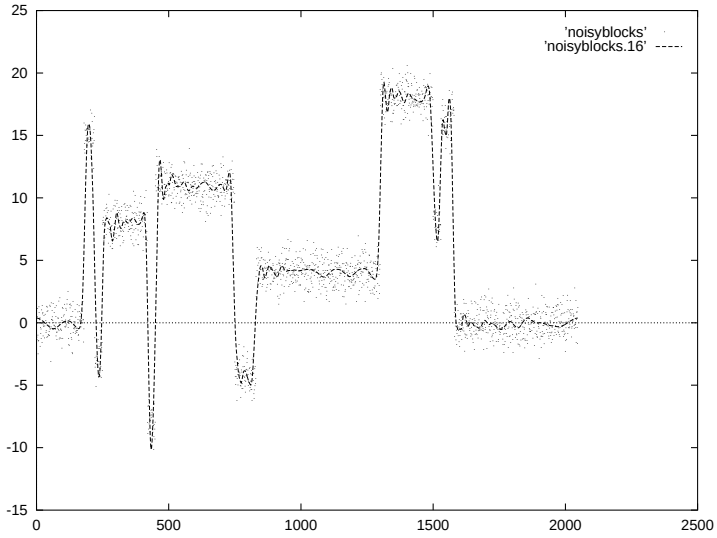

Abbildung 63: Anwendung von Algorithmus 3.7 auf den Datensatz „NoisyBlocks” mit $l = 15$

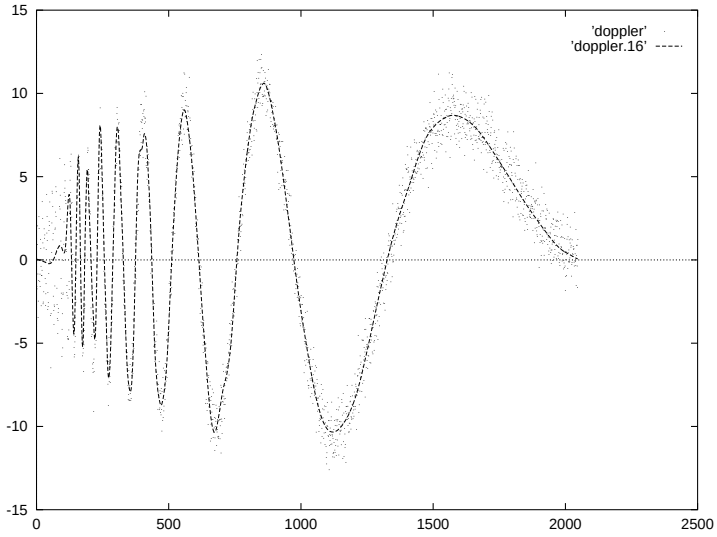

Abbildung 64: Anwendung von Algorithmus 3.7 auf den Datensatz „Doppler” mit $l = 15$

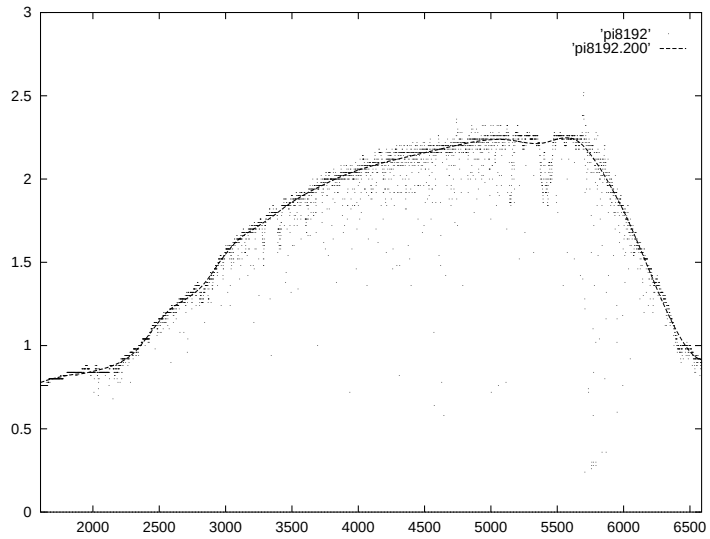

Abbildung 65: Anwendung von Algorithmus 3.7 auf die Ballon-Daten mit $l = 200$

und dem „Doppler"-Datensatz resultierte das 0.95-Quantil und die Wahl bei den Ballon-Daten hat mit den Quantilen nichts zu tun.

Es fällt leider sehr schwer, diese Bestimmung eines adäquaten Wertes für l zu automatisieren, da schon kleine Änderungen in l zum Teil deutlich verschiedene Kurven produzieren. Abbildung 66 zeigt für die Motorrad-Daten die beiden Schätzkurven für (den nur um eins größeren Wert) $l = 6$ sowie für das 0.95-Quantil $l = 11$. Auf der anderen Seite zeigt Abbildung 67 die Kurve für (den nur um eins kleineren Wert) $l = 14$, die man zugleich auch für alle $l \in \{9, \ldots, 14\}$ erhält. Was schließlich die Ballon-Daten betrifft, war aufgrund der besonderen Ausreißer-Situation klar, daß weder das 0.05-Quantil noch das 0.95-Quantil zu einem geeigneten Modell führen würden. Trotzdem hätte man eigentlich erwartet, daß die Wahl $l = 150$ ausreichen würde, da 150 die maximale Anzahl an aufeinanderfolgenden Ausreißern darstellt. Doch statt dessen erhält man die Kurve in Abbildung 68.

Besserung kann man sich auf Kosten einer höheren Rechenzeit verschaffen, indem man den zweiten Teil von Algorithmus derart erweitert, daß man nicht aufhört, sobald es nicht möglich ist, den betragskleinsten Koeffizienten zu entfernen, sondern statt dessen versucht, den zweitkleinsten (dritt-kleinsten, usw.) Koeffizienten zu entfernen, bis man zu einem Modell gelangt, wo Weglassen eines beliebigen Koeffizienten zu einer Kurve führt, bei der die Run-Bedingung verletzt ist. Im Falle des Ballon-Datensatzes führt dieser Weg tatsächlich zum Erfolg und liefert nach weiteren 45 Minuten Rechenzeit die Kurve, die in Abbildung 69 dargestellt ist. Auch bei den NoisyBlocks wird durch Anwenden des erweiterten Algorithmus das Rauschen eliminiert, wie in Abbildung 70 für $l = 10$ dargestellt ist. Allerdings betrug die gesamte Rechenzeit auch vier Stunden.

Im Rahmen dieser Diplomarbeit wurden noch viele andere Möglichkeiten der l^1-Approximation mit Wavelets ausprobiert, die oftmals bei weitaus weniger Rechenzeit bei vielen Datensätzen gute Ergebnisse erzielen konnten. Dabei wurde für einen Vektor $v \in \mathbb{R}^n$ mit $\mathrm{cbr}_l(v)$ die Anzahl der Vorzeichenruns der Länge $l + 1$ von v bezeichnet, genauer:

$$\mathrm{cbr}_l(v) = \#\{i : \forall j \in \{0, \ldots, l\} : v_i v_{i+j} > 0\}.$$

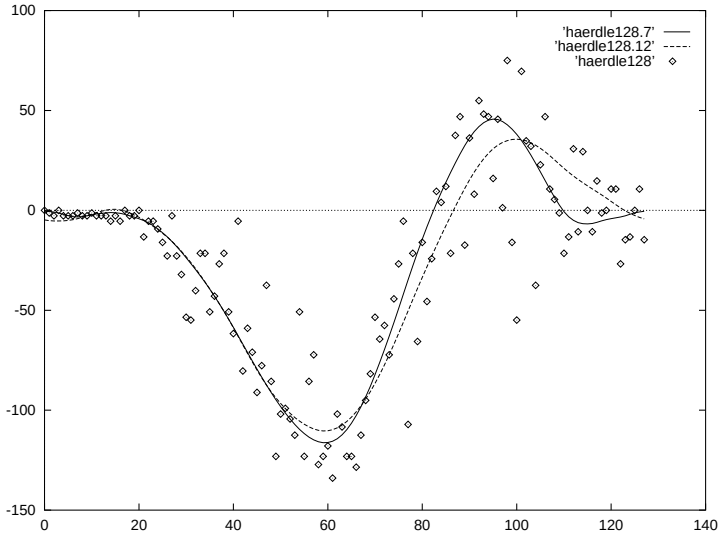

Abbildung 66: Anwendung von Algorithmus 3.7 auf die Crash-Daten mit $l = 6$ und $l = 11$

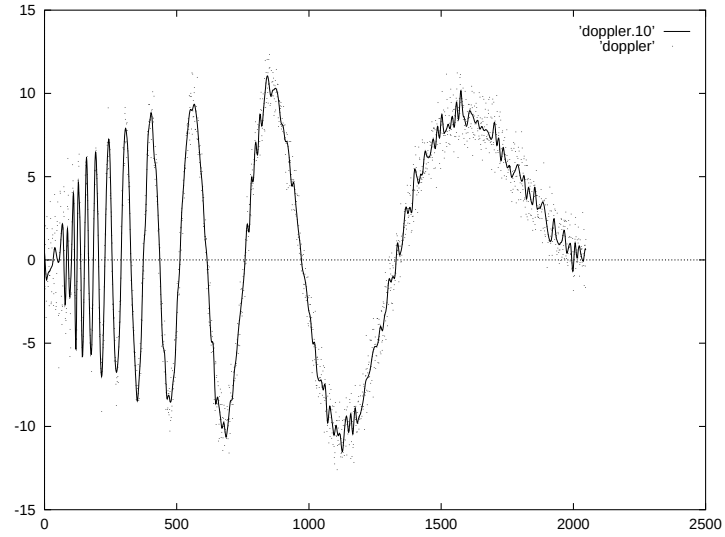

Abbildung 67: Anwendung von Algorithmus 3.7 auf den Datensatz „Doppler" mit $l = 14$

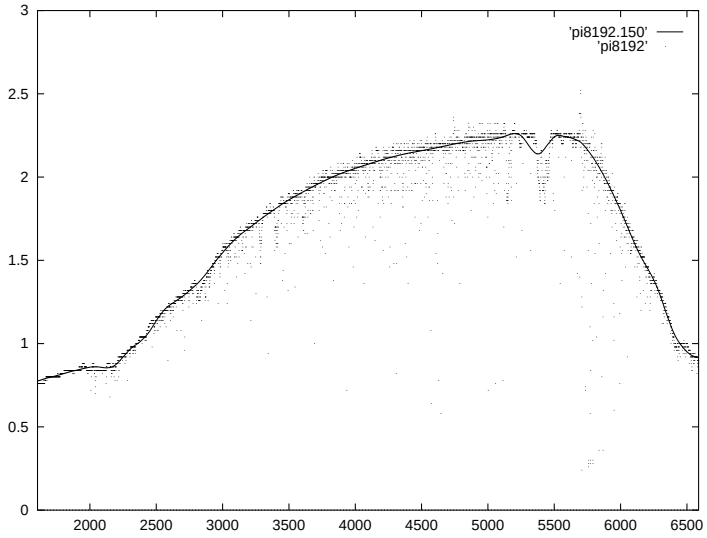

Abbildung 68: Anwendung von Algorithmus 3.7 auf die Ballon-Daten mit $l = 150$

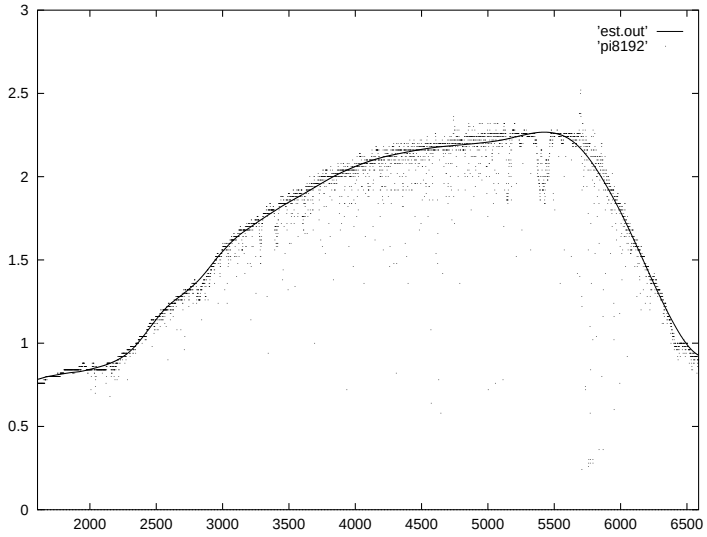

Abbildung 69: Anwendung des erweiterten Algorithmus 3.7 auf die Ballon-Daten mit $l = 150$

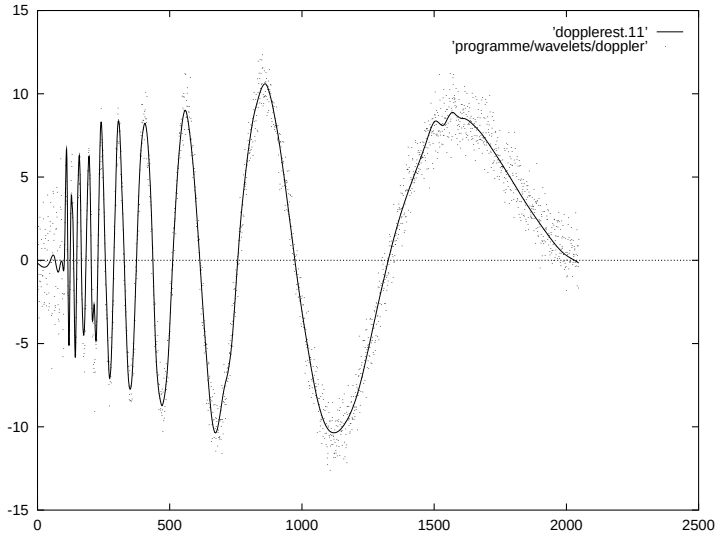

Abbildung 70: Anwendung des erweiterten Algorithmus 3.7 auf den Datensatz „Doppler" mit $l = 10$

Der einfachste Ansatz unter Verwendung von cbr_l sieht zum Beispiel so aus:

Algorithmus 3.8.

(1) Setze $r := y$, $\hat{f} := 0$ *und* $i = 0$.

(2) Falls $\mathrm{cbr}_l(r) = 0$, *fertig.*

(3) Bestimme $\alpha_i \in \mathbb{R}$, *so daß* $||r - \alpha w_i||_1$ *minimal.*

(4) Falls $\mathrm{cbr}_l(r - \alpha w_i) < \mathrm{cbr}_l(r)$, *so setze* $r := r - \alpha w_i$ *und* $\hat{f} := \hat{f} + \alpha w_i$.

(5) Setze $i := i + 1$.

(6) Weiter bei Schritt (2).

Bei anderen Ansätzen wird immer, wenn die Bedingung in (4) erfüllt ist, $i = 0$ gesetzt. Desweiteren kann man das $<$-Zeichen in Bedingung (4) gegen ein $\leq$-Zeichen austauschen, wobei man im $=$-Fall zusätzlich fordert, daß die l_1-Norm der neuen Residuen echt kleiner als die l_1-Norm der alten Residuen ist. Es ergaben sich aber bei diesen und noch einer Reihe von anderen Varianten stets sehr instabile Verfahren, wobei sich die Instabilität zum Beispiel darin äußerte, daß das Ergebnis oftmals stark von der verwendeten Wavelet-Basis abhing und bei Wahl einer bestimmten Basis versagte, während es bei allen anderen Basen zufriedenstellende Ergebnisse lieferte. Desweiteren funktionierte ein Verfahren oft bei den meisten Datensätzen, scheiterte aber an einem anderen Datensatz. Ein Grund dafür dürfte darin liegen, daß $\hat{f}$ im allgemeinen nicht die l^1-Norm minimiert unter allen $f \in \mathbb{R}^n$ mit $(\mathcal{W}f)_i = 0$ für alle $i \in \{1, \ldots, n\}$ mit $(\mathcal{W}\tilde{f})_i = 0$, da im dritten Schritt des Algorithmus stets nur ein Wavelet-Koeffizient neu bestimmt wird.

Dies alleine rechtfertigt allerdings noch nicht die Instabilität, denn auch wenn man in Schritt (3) von Algorithmus 3.8 die l^1-Norm der Residuen bzgl. aller bereits verwendeten Wavelets minimiert wie in Schritt (5) von Algorithmus 3.7, stößt man auf ähnliche Probleme. Offensichtlich

scheint es Fälle zu geben, in denen die Hinzunahme jedes einzelnen von zwei oder mehr Koeffizienten zu keiner Verkleinerung von cbr_l führt, wohl aber die gleichzeitige Hinzunahme aller drei Koeffizienten. Solche Fälle werden durch Anwendung von Algorithmus 3.7 umgangen.

4 Source-Code

4.1 Diskrete Wavelet-Transformation

Die nachfolgenden Prozeduren implementieren die Diskrete Wavelet-Transformation und beruhen auf der Fortran-Implementation von W. Press (vergl. [28]). Vor Ausführung der ersten Wavelet-Transformation muß mit der Funktion `void pwtset(int n)` die zu verwendende Wavelet-Basis bestimmt werden. Die eigentliche Wavelet-Transformation sowie deren Inverse können mit der Funktion `void wt1(double *a, int n, int isign` berechnet werden. `a` enthält bei Aufruf von `wt1` den zu transformierenden Daten- oder Koeffizientenvektor der Länge `n`, `isign` muß positiv sein, falls die DWT berechnet werden soll, und negativ im Falle der inversen DWT.

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>
#define NMAX 8195

double c2[3]={0.0, 0.707106781187,
                   0.707106781187};

double c4[5]={0.0, 0.482962913145,
   0.836516303738,
      0.224143868042,
   0.129409522551};

double c6[7]={0.0, 0.332670552950,
   0.806891509311,
   0.459877502118,
  -0.135011020010,
  -0.085441273882,
   0.035226291882};

double c8[9]={0.0, 0.230377813309,
   0.714846570553,
   0.630880767930,
  -0.027983769417,
  -0.187034811719,
   0.030841381836,
   0.032883011667,
  -0.010597401785};

double c10[11]={0.0, 0.160102397974,
     0.603829269797,
     0.724308528438,
     0.138428145901,
    -0.242294887066,
    -0.032244869585,
     0.077571493840,
    -0.006241490213,
    -0.012580751999,
     0.003335725285};

double c12[13]={0.0, 0.111540743350,
     0.494623890398,
     0.751133908021,
     0.315250351709,
    -0.226264693965,
    -0.129766867567,
     0.097501605587,
     0.027522865530,
    -0.031582039318,
     0.000553842201,
```

```
     0.004777257511,
    -0.001077301085};

double c14[15]={0.0, 0.077852054085,
     0.396539319482,
     0.729132090846,
     0.469782287405,
    -0.143906003929,
    -0.224036184994,
     0.071309219267,
     0.080612609151,
    -0.038029936935,
    -0.016574541631,
     0.012550998556,
     0.000429577973,
    -0.001801640704,
     0.000353713800};

double c16[17]={0.0, 0.054415842243,
     0.312871590914,
     0.675630736297,
     0.585354683654,
    -0.015829105256,
    -0.284015542962,
     0.000472484574,
     0.128747426620,
    -0.017369301002,
    -0.044088253931,
     0.013981027917,
     0.008746094047,
    -0.004870352993,
    -0.000391740373,
     0.000675449406,
    -0.000117476784};

double c18[19]={0.0, 0.038077947364,
     0.243834674613,
     0.604823123690,
     0.657288078051,
     0.133197385825,
    -0.293273783279,
    -0.096840783223,
     0.148540749338,
     0.030725681479,
    -0.067632829061,
     0.000250947115,
     0.022361662124,
    -0.004723204758,
    -0.004281503682,
     0.001847646883,
     0.000230385764,
    -0.000251963189,
     0.000039347320};

double c20[21]={0.0, 0.026670057901,
     0.188176800078,
     0.527201188932,
     0.688459039454,
     0.281172343661,
    -0.249846424327,
    -0.195946274377,
     0.127369340336,
     0.093057364604,
    -0.071394147166,
    -0.029457536822,
     0.033212674059,
     0.003606553567,
    -0.010733175483,
     0.001395351747,
     0.001992405295,
      -0.000685856695,
```

```
      -0.000116466855,
       0.000093588670,
      -0.000013264203};

int ncof, ioff, joff;
double wksp[NMAX+1],cc[NMAX+1],cr[NMAX+1];

void pwt(a, n, isign)
int n,isign;
double a[n+1];
  {
  int i,ii,j,jf,jr,k,n1,ni,nj,nh,nmod;
  double ai,ai1;

  if(n<4) return;
  nmod=ncof*n;
  n1=n-1;
  nh=n/2;
  for(j=1;j<=n;j++)
    wksp[j]=0.0;
  if(isign>=0)
    {
    ii=1;
    for(i=1;i<=n;i+=2)
      {
      ni=i+nmod+ioff;
      nj=i+nmod+joff;
      for(k=1;k<=ncof;k++)
        {
        jf=n1&(ni+k);
        jr=n1&(nj+k);
        wksp[ii]=wksp[ii]+cc[k]*a[jf+1];
        wksp[ii+nh]=wksp[ii+nh]+cr[k]*a[jr+1];
        }
      ii++;
      }
    }
  else
    {
    ii=1;
    for(i=1;i<=n;i+=2)
      {
      ai=a[ii];
      ai1=a[ii+nh];
      ni=i+nmod+ioff;
      nj=i+nmod+joff;
      for(k=1;k<=ncof;k++)
        {
        jf=(n1&(ni+k))+1;
        jr=(n1&(nj+k))+1;
        wksp[jf]=wksp[jf]+cc[k]*ai;
        wksp[jr]=wksp[jr]+cr[k]*ai1;
        }
      ii++;
      }
    }
  for(j=1;j<=n;j++)
    a[j]=wksp[j];
  }

void pwtset(int n)
  {
  int k;
  double sig;

  ncof=n;
  sig=-1.0;
  for(k=1;k<=n;k++)
    {
    if(n<=2) cc[k]=c2[k];
    else if (n<= 4) cc[k]=c4[k];
```

```
    else if (n<= 6) cc[k]=c6[k];
    else if (n<= 8) cc[k]=c8[k];
    else if (n<=10) cc[k]=c10[k];
    else if (n<=12) cc[k]=c12[k];
    else if (n<=14) cc[k]=c14[k];
    else if (n<=16) cc[k]=c16[k];
    else if (n<=18) cc[k]=c18[k];
    else if (n<=20) cc[k]=c20[k];
    else
      {
      puts("unimplemented value n in pwtset");
      exit(FALSE);
      }
    cr[ncof+1-k]=sig*cc[k];
    sig=-sig;
    }
  ioff=-2;
  joff=-2-n;
  }

void wt1(a,n,isign)
int isign,n;
double a[n+1];
  {
  int nn;

  if(n<4) exit(FALSE);
  if(isign>=0)
    {
    nn=n;
    while(nn>=4)
      {
      pwt(a,nn,isign);
      nn=nn/2;
      }
    }
  else
    {
    nn=4;
    while(nn<=n)
      {
      pwt(a,nn,isign);
      nn=nn*2;
      }
    }
  }
```

4.2 Spline-Smoothing

Dieser Algorithmus wendet Spline-Smoothing auf einen Datensatz an und ist eine Portierung des Fortran-Programms von C. De Boor nach C (vergl. [4]).

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <curses.h>

#define MAXPOINTS 3000

double x[MAXPOINTS+1], dx[MAXPOINTS+1], y[MAXPOINTS+1], A[MAXPOINTS+1][5], V[MAXPOINTS+1][8];
int NPOINTS;

void cholid(),splinesmooth(),setupq();


FILE *MyOpenToRead(char *datei) /* Versucht, die Datei datei zum Lesen zu öffnen */
  {
```

```
  FILE *new = fopen(datei,"rb");

  if(new == NULL) printf("Die Datei %s ließ sich nicht öffnen\n",datei);
  return new;
  }

FILE *MyOpenToWrite(char *datei) /* Versucht, die Datei datei zum Schreiben zu öffnen */
  {
  FILE *new = fopen(datei,"wb");

  if(new == NULL) printf("Die Datei %s ließ sich nicht öffnen\n",datei);
  return new;
  }

void readdata(char *name)
  {
  FILE *indatei;

  NPOINTS=1;
  if((indatei = MyOpenToRead(name))==NULL) exit(FALSE);

  while((!feof(indatei))&&(NPOINTS<MAXPOINTS))
    {
    x[NPOINTS]=NPOINTS;
    if(fscanf(indatei,"%lf ",&y[NPOINTS])!=1) break;
    NPOINTS++;
    }
  NPOINTS--;
  fclose(indatei);
  }

void splinesmooth(s)
double s;
  {
  int i;
  double p,sfp,prevp,prevsf,utru,change,smooth,sixp,six1mp;

  setupq();
  if(s<=0.0)
    {
    p=1.0;
    cholid(p);
    sfp=0.0;
    }
  else
    {
    p=0.0;
    cholid(p);
    sfp=0.0;
    for(i=1;i<=NPOINTS;i++)
      sfp+=A[i][1]*A[i][1]*dx[i]*dx[i];
    sfp*=36.0;
    if(sfp>s)
      {
      prevp=0.0;
      prevsf=sfp;
      utru=0.0;
      for(i=2;i<=NPOINTS;i++)
utru+=V[i-1][4]*(A[i-1][3]*(A[i-1][3]+A[i][3])+A[i][3]*A[i][3]);
      p=(sfp-s)/(24.0*utru);

      while(TRUE)
{
cholid(p);
sfp=0.0;
for(i=1;i<=NPOINTS;i++)
  sfp+=A[i][1]*A[i][1]*dx[i]*dx[i];
sfp=sfp*36.0*(1.0-p)*(1.0-p);
if(sfp<=1.01*s) break;
if(sfp>=prevsf)
  {
```

```
  p=1.0;
  cholid(p);
  sfp=0.0;
  break;
  }
change=(p-prevp)/(sfp-prevsf)*(sfp-s);
prevp=p;
p=p-change;
prevsf=sfp;
if(p>=1.0)
  p=1.0-sqrt(s/prevsf)*(1.0-prevp);
}
      }
    }
  smooth=sfp;
  printf("p: %f    %f\n",p,(1-p)/p);
  six1mp=6.0*(1.0-p);
  for(i=1;i<=NPOINTS;i++)
    A[i][1]=y[i]-six1mp*dx[i]*dx[i]*A[i][1];
  sixp=6.0*p;
  for(i=1;i<=NPOINTS;i++)
    A[i][3]=A[i][3]*sixp;
  for(i=1;i<=NPOINTS-1;i++)
    {
    A[i][4]=(A[i+1][3]-A[i][3])/V[i][4];
    A[i][2]=(A[i+1][1]-A[i][1])/V[i][4]-(A[i][3]+A[i][4]/3.0*V[i][4])/2.0*V[i][4];
    }
  }

void setupq()
  {
  int NPM1=NPOINTS-1,i;
  double prev,diff;

  V[1][4]=x[2]-x[1];
  for(i=2;i<=NPM1;i++)
    {
    V[i][4]=x[i+1]-x[i];
    V[i][1]=dx[i-1]/V[i-1][4];
    V[i][2]=-dx[i]/V[i][4]-dx[i]/V[i-1][4];
    V[i][3]=dx[i+1]/V[i][4];
    V[i][5]=V[i][1]*V[i][1]+V[i][2]*V[i][2]+V[i][3]*V[i][3];
    }
  V[NPOINTS][1]=0;
  if(NPM1>=3)
    for(i=3;i<=NPM1;i++)
      V[i-1][6]=V[i-1][2]*V[i][1]+V[i-1][3]*V[i][2];
  V[NPM1][6]=0;
  if(NPM1>=4)
    for(i=4;i<=NPM1;i++)
      V[i-2][7]=V[i-2][3]*V[i][1];
  V[NPM1-1][7]=0;
  V[NPM1][7]=0;
  prev=(y[2]-y[1])/V[1][4];
  for(i=2;i<=NPM1;i++)
    {
    diff = (y[i+1]-y[i])/V[i][4];
    A[i][4]=diff-prev;
    prev=diff;
    }
  }

void cholid(p)
double p;
  {
  int NPM1=NPOINTS-1, NPM2=NPOINTS-2, i;
  double six1mp,twop,ratio,prev;

  six1mp=6.0*(1.0-p);
  twop=2.0*p;
  for(i=2;i<=NPM1;i++)
```

```
    {
    V[i][1]=six1mp*V[i][5]+twop*(V[i-1][4]+V[i][4]);
    V[i][2]=six1mp*V[i][6]+p*V[i][4];
    V[i][3]=six1mp*V[i][7];
    }
  if(NPM2<2)
    {
    A[1][3]=0;
    A[2][3]=A[2][4]/V[2][1];
    A[3][3]=0;
    }
  else
    {
    for(i=2;i<=NPM2;i++)
      {
      ratio = V[i][2]/V[i][1];
      V[i+1][1] = V[i+1][1]-ratio*V[i][2];
      V[i+1][2] = V[i+1][2]-ratio*V[i][3];
      V[i][2]=ratio;
      ratio=V[i][3]/V[i][1];
      V[i+2][1]=V[i+2][1]-ratio*V[i][3];
      V[i][3]=ratio;
      }
    A[1][3] = 0;
    V[1][3] = 0;
    A[2][3] = A[2][4];
    for(i=2;i<=NPM2;i++)
      A[i+1][3]=A[i+1][4]-V[i][2]*A[i][3]-V[i-1][3]*A[i-1][3];
    A[NPOINTS][3]=0;
    A[NPM1][3]=A[NPM1][3]/V[NPM1][1];
    for(i=NPM2;i>=2;i--)
      A[i][3]=A[i][3]/V[i][1]-A[i+1][3]*V[i][2]-A[i+2][3]*V[i][3];
    }
  prev=0;
  for(i=2;i<=NPOINTS;i++)
    {
    A[i][1]=(A[i][3]-A[i-1][3])/V[i-1][4];
    A[i-1][1]=A[i][1]-prev;
    prev=A[i][1];
    }
  A[NPOINTS][1]=-A[NPOINTS][1];
  }

void writesplines(res)
double res;
  {
  double t = 0.0,s;
  FILE *outdatei;
  int n=1;

  if((outdatei = MyOpenToWrite("smooth.RES"))==NULL) exit(FALSE);

  while(n<NPOINTS-1)
    {
    s=t-x[n];
    fprintf(outdatei,"%f %f\n",t,A[n][4]*s*s*s/6.0+A[n][3]*s*s/2.0+A[n][2]*s+A[n][1]);
    t+=res;
    if(t>x[NPOINTS]) break;
    while(t>x[n+1]-0.00001) n++;
    }
  fclose(outdatei);
  }

main()
  {
  int i,j;
  double s;
  char name[80];

  printf("Name des Datensatzes: ");
  scanf("%s",name);
```

```
  printf("Please enter s: ");
  readdata(name);
  scanf("%lf",&s);
  for(i=1;i<=NPOINTS;i++) dx[i]=1.0;
  splinesmooth(s);
  writesplines(1.0);
  }
```

4.3 Hilfsfunktionen

Die hier benutzten Hilfsfunktionen werden von den folgenden Programmen benutzt.

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>

#define NMAX 8195

int dcomp(double *a, double *b)
  {
  if(*a>*b) return 1;
  else if(*a==*b) return 0;
  else return -1;
  }

void dqsort(double *x, int n)
  {
  qsort(x,n,sizeof(double),(void *)(*dcomp));
  }

double MED(double *x,int n)
  {
  double xbak[n];

  memcpy(xbak,x,sizeof(xbak));
  dqsort(xbak,n);
  if(n%2==0)
    return (xbak[n/2-1]+xbak[n/2])/2.0;
  else
    return xbak[(n+1)/2-1];
  }

double MAD(double *x, int n)
  {
  int i;
  double r[n],m;
  m=MED(x,n);
  for(i=0;i<n;i++)
    r[i]=fabs(x[i]-m);
  return MED(r,n);
  }


void readdata(double *a,int *n,char *dateiname)
  {
  FILE *indatei;
  *n=0;

  if((indatei = fopen(dateiname,"rb"))==NULL) exit(FALSE);

  while((!feof(indatei))&&(*n<NMAX))
    {
    if(fscanf(indatei,"%lf ",&a[*n+1])!=1) break;
    (*n)++;
    }

  fclose(indatei);
```

```
  }

void printvector(char *c,double *a,int n)
  {
  FILE *fd;
  int i;

  fd=fopen(c,"w");
  if(fd!=NULL)
    {
    for(i=1;i<=n;i++)
      fprintf(fd,"%f\n",a[i]);
    fprintf(fd,"\n");
    fclose(fd);
    }
  }

int maxrunlength(double *x, int n)
  {
  int SIGN,i,actual=1,max=1;

  if(fabs(x[1])>=0)
    SIGN=1;
  else
    SIGN=-1;
  for(i=2;i<=n;i++)
    {
    if(x[i]>=0)
      if(SIGN==1)
        actual++;
      else
        {
        if(actual>max)
          max=actual;
        actual=1;
        SIGN=1;
        }
    else
      if(SIGN==-1)
        actual++;
      else
        {
        if(actual>max)
          max=actual;
        actual=1;
        SIGN=-1;
        }
    }
  if(actual>max)
    max=actual;
  return(max);
  }
```

4.4 Algorithmus 3.1

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>
#define NMAX 8195

int zwpo[11]={1,2,4,8,16,32,64,128,256,512,1024};

extern int ncof, ioff, joff;
extern double wksp[NMAX+1],cc[NMAX+1],cr[NMAX+1];

extern double MAD(double *x, int n);
extern void readdata(double *a,int *n,char *dateiname);
```

```
extern void printvector(char *c,double *a,int n);
extern void pwtset(int n);
extern void wt1(double *a,int n,int isign);
extern int maxrunlength(double *x, int n);

main()
  {
  double res[8195],a[8195],x[8195],grenze,sigma,t;
  int i,n,nrwav,j0,zaehler;
  char dummy[80],name[200],name2[200];
  double maxres,delta,xorg[8195];

  printf("Datensatz: ");
  scanf("%s",name);
  gets(dummy);
  readdata(x,&n,name);
  printf("%d Daten gelesen!\n",n);
  printf("Welches Wavelet (4,10,12)? ");
  scanf("%d",&nrwav);
  pwtset(nrwav);
  printf("j0: ");
  scanf("%d",&j0);
  printf("delta: ");
  scanf("%lf",&delta);
  for(i=1;i<=n;i++)
    {
    xorg[i]=x[i];
    a[i] = x[i];
    }
  wt1(a,n,1);
  sigma=MAD(&(a[n/2+1]),n/2)/0.6745;
  printf("sigma ist %f\n",sigma);
  grenze=sqrt(2.0*log((double)n))*sigma;
  zaehler=0;
  do
    {
    zaehler++;
    for(i=1;i<=n;i++)
      a[i] = x[i];
    wt1(a,n,1);
    for(i=zwpo[j0]+1;i<=n;i++)
      if(a[i]<-grenze)
        a[i]+=grenze;
      else
      if(a[i]>grenze)
        a[i]-=grenze;
      else
        a[i]=0.0;
    wt1(a,n,-1);
    maxres=0.0;
    for(i=1;i<=n;i++)
      if(fabs(x[i]-a[i])>maxres)
        maxres=fabs(x[i]-a[i]);
    for(i=1;i<=n;i++)
      if(fabs(x[i]-a[i])>0.95*maxres)
        x[i]=a[i];
      for(i=1;i<=n;i++)
        res[i]=a[i]-xorg[i];
    } while(maxres>delta);
  printf("zaehler: %d -- maxres: %f\n",zaehler,maxres);
  printf("Maximale Runlaenge: %d\n",maxrunlength(res,n));
  sprintf(name2,"%s.est",name);
  printvector(name2,a,n);
  sprintf(name2,"%s.neu",name);
  printvector(name2,x,n);
  }
```

4.5 Algorithmus 3.4

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>
#define NMAX 8195

int zwpo[11]={1,2,4,8,16,32,64,128,256,512,1024};
double storeq[1000][20], storer[20][20], storeb[20];

extern int ncof, ioff, joff;
extern double wksp[NMAX+1],cc[NMAX+1],cr[NMAX+1];
extern double MAD(double *x, int n);
extern void readdata(double *a,int *n,char *dateiname);
extern void printvector(char *c,double *a,int n);
extern void pwtset(int n);
extern void wt1(double *a,int n,int isign);

void qrdecomp(double *abak, double *qbak, double *rbak, int n, int k)
  {
  double a[n][k], q[n][k], r[k][k],sum,b[n];
  int i, j, l;

  for(i=0;i<n;i++)
    for(j=0;j<k;j++)
      a[i][j]=abak[i*k+j];

  sum=0.0;
  for(i=0;i<n;i++)
    sum+=a[i][0]*a[i][0];
  r[0][0]=sqrt(sum);
  for(i=1;i<n;i++)
    r[i][0]=0.0;
  for(i=0;i<n;i++)
    q[i][0]=a[i][0]/r[0][0];
  for(j=1;j<k;j++)
    {
    for(i=0;i<=j-1;i++)
      {
      r[i][j]=0.0;
      for(l=0;l<n;l++)
        r[i][j]+=a[l][j]*q[l][i];
      }
    for(i=0;i<n;i++)
      b[i]=a[i][j];
    for(l=0;l<=j-1;l++)
      {
      for(i=0;i<n;i++)
        b[i]-=r[l][j]*q[i][l];
      }
    sum=0.0;
    for(i=0;i<n;i++)
      sum+=b[i]*b[i];
    r[j][j]=sqrt(sum);
    for(i=j+1;i<n;i++)
      r[i][j]=0.0;
    for(i=0;i<n;i++)
      q[i][j]=b[i]/r[j][j];
    }
  for(i=0;i<n;i++)
    for(j=0;j<k;j++)
      qbak[i*k+j]=q[i][j];
  for(i=0;i<k;i++)
    for(j=0;j<k;j++)
      rbak[i*k+j]=r[i][j];
  }

void storeit(double *x, int n, int k)
  {
```

```
  int i,j;
  double q[n][k], r[k][k], b[k];

  qrdecomp(x,q,r,n,k);

  for(i=0;i<k;i++)
    {
    for(j=0;j<k;j++)
      storer[i][j]=r[i][j];
    for(j=0;j<n;j++)
      storeq[j][i]=q[j][i];
    storeb[i]=b[i];
    }
  }

void storedlinreg(double *y, double *beta, int n, int k)
  {
  int i,j;

  for(i=0;i<k;i++)
    {
    storeb[i]=0;
    for(j=0;j<n;j++)
      storeb[i]+=storeq[j][i]*y[j];
    }
  for(i=k-1;i>=0;i--)
    {
    beta[i]=storeb[i];
    for(j=i+1;j<k;j++)
      beta[i]-=beta[j]*storer[i][j];
    beta[i]=beta[i]/storer[i][i];
    }
  }

void compwv(double *wv, int *laenge, int n, int level)
  {
  double wavelet[n+1];
  int i;

  for(i=1;i<=n;i++)
    wavelet[i]=0;
  wavelet[n/zwpo[level]]=1;
  wt1(wavelet,n,-1);
  i=1;
  while(fabs(wavelet[i])>0.000001) i++;
  while(fabs(wavelet[i])<=0.000001) i++;
  *laenge=0;
  while(fabs(wavelet[i])>0.000001)
    {
    (*laenge)++;
    wv[*laenge]=wavelet[i];
    i++;
    if(i>n) i=1;
    }
  }

void swt(double *data, double *coeff, int n, int level)
  {
  double wv[n+1],summe;
  int i,j,laenge;

  compwv(wv,&laenge,n,level);
  for(i=1;i<=n;i++)
    {
    summe=0;
    for(j=1;j<=laenge;j++)
      summe+=wv[j]*data[(i+j-2)%n+1];
    coeff[i]=summe;
    }
  }
```

```
int bigcoeffs(double *a, int n, double thresh)
  {
  int i,zaehler=0;

  for(i=1;i<=n;i++)
    if(fabs(a[i])>thresh+0.00001)
      zaehler++;
  return zaehler;
  }

main()
  {
  char ALLESKLEINER,dummy[80],name[30],FERTIG;
  double AAA[2500],A[1000][20],b[1000];
  double abak[6][8195],a[6][8195],x[8195],grenze,sigma,t;
  double a2[8195],thresh,xbak[8195],bigwv[6][8195],beta[1000];
  int i,n,j0,zaehler,j,laenge[6],N,anzcand,anzcandbak,k,pos,l;
  int index,ii,jj,kmin,kmax,outlierlevel,maxoutlierlevel;
  int maxN,filezaehler,maxdepth,depth;

  printf("Datensatz: ");
  scanf("%s",name);
  gets(dummy);
  readdata(x,&n,name);
  printf("%d Daten gelesen!\n",n);

  printf("maxoutlierlevel: ");
  scanf("%d",&maxoutlierlevel);
  printf("maxN: ");
  scanf("%d",&maxN);
  do
    {
    printf("maxdepth: ");
    scanf("%d",&maxdepth);
    } while(maxdepth>6);

  /* Varianz schätzen */
  sigma=500000000;
  for(N=2;N<=20;N++)
    {
    pwtset(N);
    swt(x,a[0],n,0);
    j=0;
    for(i=1;i<=n;i++)
      if(fabs(a[0][i])>0.0001)
        a2[++j]=a[0][i];
    if(j<n/2)
    puts("Sind die Daten wirklich verrauscht? Mehr als die Hälfte der W-Koeffs ergibt 0");
    if(MAD(&(a2[1]),j)/0.6745<sigma)
      sigma=MAD(&(a2[1]),j)/0.6745;
    }
  printf("sigma: %f\n",sigma);

  /* Threshold bestimmen */
  thresh=2.33*sigma;
  thresh=1.96*sigma;
  thresh=sqrt(2*log((double)n))*sigma;
  printf("Threshold: %f\n",thresh);

  for(N=2;N<=maxN;N+=2) /* N wird unten erhoeht */
    {
    printf("N: %d\n",N);
    pwtset(N);
    for(i=0;i<maxdepth;i++)
      compwv(bigwv[i],&(laenge[i]),n,i);

    for(outlierlevel=1;outlierlevel<=maxoutlierlevel;outlierlevel++)
    for(depth=0;depth<maxdepth;depth++)
      {
      /* Matrix berechnen und QR-Zerlegung speichern */
      for(ii=1;ii<=laenge[depth]+outlierlevel-1;ii++)
```

```
  {
  for(jj=1;jj<=outlierlevel;jj++)
    {
    index=laenge[depth]-ii+jj;
    if((index<1)||(index>laenge[depth]))
      A[ii][jj]=0.0;
    else
      A[ii][jj]=bigwv[depth][index];
    }
  }
j=0;
for(ii=1;ii<=laenge[depth]+outlierlevel-1;ii++)
  for(jj=1;jj<=outlierlevel;jj++)
    AAA[j++]=A[ii][jj];
storeit(AAA,laenge[depth]+outlierlevel-1,outlierlevel);

do
  {
  FERTIG=TRUE;
  pwtset(N);
  anzcand=0;
  for(i=0;i<maxdepth;i++)
    {
    swt(x,a[i],n,i);
    anzcand+=bigcoeffs(a[i],n,thresh);
    }

  printf("depth: %d --- outlevel: %d -- Noch %d grosse Koeffs.\n",
                                              depth,outlierlevel,anzcand);

  for(pos=1;(pos<=n-outlierlevel+1)&&(anzcand>0);pos++)
    {
    /* Prüfe, ob Datenpunkte pos bis pos+outlierlevel-1
       Ausreißer sein könnten, zunächst aber, ob
       überhaupt ein beeinflußter Koeffizient
       groß ist. */
    ALLESKLEINER=TRUE;
    for(j=(pos-laenge[depth])%n+1;ALLESKLEINER&&(j<=pos+outlierlevel-1);j++)
      if(fabs(a[depth][j])>thresh)
        ALLESKLEINER=FALSE;
    if(!ALLESKLEINER)
      {
      for(j=1;j<=n;j++)
        xbak[j]=x[j];

      for(ii=1;ii<=laenge[depth]+outlierlevel-1;ii++)
        {
        b[ii]=0.0;
        for(jj=1;jj<=laenge[depth];jj++)
          {
          l=(pos-laenge[depth]+ii+jj-2+n)%n+1;
          if((l<pos)||(l>=pos+outlierlevel))
            b[ii]-=bigwv[depth][jj]*x[l];
          }
        }
      storedlinreg(&(b[1]),beta,laenge[depth]+outlierlevel-1,outlierlevel);

      for(j=pos;j<=pos+outlierlevel-1;j++)
        xbak[j]=beta[j-pos];
      for(i=0;i<maxdepth;i++)
        swt(xbak,abak[i],n,i);
      ALLESKLEINER=TRUE;
      for(l=0;l<=depth;l++)
        for(j=pos-laenge[l]+1;(j<=pos+outlierlevel-1)&&ALLESKLEINER;j++)
          if((fabs(abak[l][j])>thresh)&&(fabs(a[l][j])<=thresh))
            ALLESKLEINER=FALSE;
      if(ALLESKLEINER)
        {
        anzcandbak=0;
        for(i=0;i<maxdepth;i++)
          anzcandbak+=bigcoeffs(abak[i],n,thresh);
```

```
            }
          if((ALLESKLEINER)&&(anzcandbak<anzcand))
            {
            printf("%d bis %d Ausreißer\n",pos,pos+outlierlevel-1);
            for(j=pos;j<=pos+outlierlevel-1;j++)
              {
              FERTIG=FALSE;
              x[j]=xbak[j];
              }
            for(i=0;i<maxdepth;i++)
              swt(x,a[i],n,i);
            anzcand=anzcandbak;
            printf("anzcand: %d\n",anzcand);
            }
          } /* end if(!ALLESKLEINER) */
        } /* end for pos */

      } while(!FERTIG);
    } /* end for outlierlevel */

  filezaehler++;
  sprintf(name,"robust%d.out",filezaehler);
  printf("%s\n",name);
  printvector(name,x,n);
  } /* end for N */

for(i=1;i<=n;i++)
  xbak[i]=x[i];
do
  {
  printf("j0: ");
  scanf("%d",&j0);
  wt1(x,n,1);
  for(i=zwpo[j0]+1;i<=n;i++)
    if((x[i]<=thresh)&&(-thresh<=x[i]))
      x[i]=0.0;
  wt1(x,n,-1);
  printvector("est.out",x,n);
  for(i=1;i<=n;i++)
    x[i]=xbak[i];
  } while(j0>0);
} /* main */
```

4.6 Algorithmus 3.7

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>
#define NMAX 8195

int zwpo[11]={1,2,4,8,16,32,64,128,256,512,1024};
double storeq[1000][20], storer[20][20], storeb[20];

extern int ncof, ioff, joff;
extern double wksp[NMAX+1],cc[NMAX+1],cr[NMAX+1];
extern double MAD(double *x, int n);
extern void readdata(double *a,int *n,char *dateiname);
extern void printvector(char *c,double *a,int n);
extern void pwtset(int n);
extern void wt1(double *a,int n,int isign);

void qrdecomp(double *abak, double *qbak, double *rbak, int n, int k)
  {
  double a[n][k], q[n][k], r[k][k],sum,b[n];
  int i, j, l;

  for(i=0;i<n;i++)
```

```
    for(j=0;j<k;j++)
      a[i][j]=abak[i*k+j];

  sum=0.0;
  for(i=0;i<n;i++)
    sum+=a[i][0]*a[i][0];
  r[0][0]=sqrt(sum);
  for(i=1;i<n;i++)
    r[i][0]=0.0;
  for(i=0;i<n;i++)
    q[i][0]=a[i][0]/r[0][0];
  for(j=1;j<k;j++)
    {
    for(i=0;i<=j-1;i++)
      {
      r[i][j]=0.0;
      for(l=0;l<n;l++)
        r[i][j]+=a[l][j]*q[l][i];
      }
    for(i=0;i<n;i++)
      b[i]=a[i][j];
    for(l=0;l<=j-1;l++)
      {
      for(i=0;i<n;i++)
        b[i]-=r[l][j]*q[i][l];
      }
    sum=0.0;
    for(i=0;i<n;i++)
      sum+=b[i]*b[i];
    r[j][j]=sqrt(sum);
    for(i=j+1;i<n;i++)
      r[i][j]=0.0;
    for(i=0;i<n;i++)
      q[i][j]=b[i]/r[j][j];
    }
  for(i=0;i<n;i++)
    for(j=0;j<k;j++)
      qbak[i*k+j]=q[i][j];
  for(i=0;i<k;i++)
    for(j=0;j<k;j++)
      rbak[i*k+j]=r[i][j];
  }

void storeit(double *x, int n, int k)
  {
  int i,j;
  double q[n][k], r[k][k], b[k];

  qrdecomp(x,q,r,n,k);

  for(i=0;i<k;i++)
    {
    for(j=0;j<k;j++)
      storer[i][j]=r[i][j];
    for(j=0;j<n;j++)
      storeq[j][i]=q[j][i];
    storeb[i]=b[i];
    }
  }

void storedlinreg(double *y, double *beta, int n, int k)
  {
  int i,j;

  for(i=0;i<k;i++)
    {
    storeb[i]=0;
    for(j=0;j<n;j++)
      storeb[i]+=storeq[j][i]*y[j];
    }
  for(i=k-1;i>=0;i--)
```

```
    {
    beta[i]=storeb[i];
    for(j=i+1;j<k;j++)
      beta[i]-=beta[j]*storer[i][j];
    beta[i]=beta[i]/storer[i][i];
    }
  }

void compwv(double *wv, int *laenge, int n, int level)
  {
  double wavelet[n+1];
  int i;

  for(i=1;i<=n;i++)
    wavelet[i]=0;
  wavelet[n/zwpo[level]]=1;
  wt1(wavelet,n,-1);
  i=1;
  while(fabs(wavelet[i])>0.000001) i++;
  while(fabs(wavelet[i])<=0.000001) i++;
  *laenge=0;
  while(fabs(wavelet[i])>0.000001)
    {
    (*laenge)++;
    wv[*laenge]=wavelet[i];
    i++;
    if(i>n) i=1;
    }
  }

void swt(double *data, double *coeff, int n, int level)
  {
  double wv[n+1],summe;
  int i,j,laenge;

  compwv(wv,&laenge,n,level);
  for(i=1;i<=n;i++)
    {
    summe=0;
    for(j=1;j<=laenge;j++)
      summe+=wv[j]*data[(i+j-2)%n+1];
    coeff[i]=summe;
    }
  }

int bigcoeffs(double *a, int n, double thresh)
  {
  int i,zaehler=0;

  for(i=1;i<=n;i++)
    if(fabs(a[i])>thresh+0.00001)
      zaehler++;
  return zaehler;
  }

main()
  {
  char ALLESKLEINER,dummy[80],name[30],FERTIG;
  double AAA[2500],A[1000][20],b[1000];
  double abak[6][8195],a[6][8195],x[8195],grenze,sigma,t;
  double a2[8195],thresh,xbak[8195],bigwv[6][8195],beta[1000];
  int i,n,j0,zaehler,j,laenge[6],N,anzcand,anzcandbak,k,pos,l;
  int index,ii,jj,kmin,kmax,outlierlevel,maxoutlierlevel;
  int maxN,filezaehler,maxdepth,depth;

  printf("Datensatz: ");
  scanf("%s",name);
  gets(dummy);
  readdata(x,&n,name);
  printf("%d Daten gelesen!\n",n);
```

```
printf("maxoutlierlevel: ");
scanf("%d",&maxoutlierlevel);
printf("maxN: ");
scanf("%d",&maxN);
do
  {
  printf("maxdepth: ");
  scanf("%d",&maxdepth);
  } while(maxdepth>6);

/* Varianz schätzen */
sigma=500000000;
for(N=2;N<=20;N++)
  {
  pwtset(N);
  swt(x,a[0],n,0);
  j=0;
  for(i=1;i<=n;i++)
    if(fabs(a[0][i])>0.0001)
      a2[++j]=a[0][i];
  if(j<n/2)
  puts("Sind die Daten wirklich verrauscht? Mehr als die Hälfte der W-Koeffs ergibt 0");
  if(MAD(&(a2[1]),j)/0.6745<sigma)
    sigma=MAD(&(a2[1]),j)/0.6745;
  }
printf("sigma: %f\n",sigma);

/* Threshold bestimmen */
thresh=2.33*sigma;
thresh=1.96*sigma;
thresh=sqrt(2*log((double)n))*sigma;
printf("Threshold: %f\n",thresh);

for(N=2;N<=maxN;N+=2) /* N wird unten erhoeht */
  {
  printf("N: %d\n",N);
  pwtset(N);
  for(i=0;i<maxdepth;i++)
    compwv(bigwv[i],&(laenge[i]),n,i);

  for(outlierlevel=1;outlierlevel<=maxoutlierlevel;outlierlevel++)
  for(depth=0;depth<maxdepth;depth++)
    {
    /* Matrix berechnen und QR-Zerlegung speichern */
    for(ii=1;ii<=laenge[depth]+outlierlevel-1;ii++)
      {
      for(jj=1;jj<=outlierlevel;jj++)
        {
        index=laenge[depth]-ii+jj;
        if((index<1)||(index>laenge[depth]))
          A[ii][jj]=0.0;
        else
          A[ii][jj]=bigwv[depth][index];
        }
      }
    j=0;
    for(ii=1;ii<=laenge[depth]+outlierlevel-1;ii++)
      for(jj=1;jj<=outlierlevel;jj++)
        AAA[j++]=A[ii][jj];
    storeit(AAA,laenge[depth]+outlierlevel-1,outlierlevel);

    do
      {
      FERTIG=TRUE;
      pwtset(N);
      anzcand=0;
      for(i=0;i<maxdepth;i++)
        {
        swt(x,a[i],n,i);
        anzcand+=bigcoeffs(a[i],n,thresh);
        }
```

```
    printf("depth: %d --- outlevel: %d -- Noch %d grosse Koeffs.\n",
                                        depth,outlierlevel,anzcand);

    for(pos=1;(pos<=n-outlierlevel+1)&&(anzcand>0);pos++)
      {
      /* Prüfe, ob Datenpunkte pos bis pos+outlierlevel-1
         Ausreißer sein könnten, zunächst aber, ob
         überhaupt ein beeinflußter Koeffizient
         groß ist. */
      ALLESKLEINER=TRUE;
      for(j=(pos-laenge[depth])%n+1;ALLESKLEINER&&(j<=pos+outlierlevel-1);j++)
        if(fabs(a[depth][j])>thresh)
          ALLESKLEINER=FALSE;
      if(!ALLESKLEINER)
        {
        for(j=1;j<=n;j++)
          xbak[j]=x[j];

        for(ii=1;ii<=laenge[depth]+outlierlevel-1;ii++)
          {
          b[ii]=0.0;
          for(jj=1;jj<=laenge[depth];jj++)
            {
            l=(pos-laenge[depth]+ii+jj-2+n)%n+1;
            if((l<pos)||(l>=pos+outlierlevel))
              b[ii]-=bigwv[depth][jj]*x[l];
            }
          }
        storedlinreg(&(b[1]),beta,laenge[depth]+outlierlevel-1,outlierlevel);

        for(j=pos;j<=pos+outlierlevel-1;j++)
          xbak[j]=beta[j-pos];
        for(i=0;i<maxdepth;i++)
          swt(xbak,abak[i],n,i);
        ALLESKLEINER=TRUE;
        for(l=0;l<=depth;l++)
          for(j=pos-laenge[l]+1;(j<=pos+outlierlevel-1)&&ALLESKLEINER;j++)
            if((fabs(abak[l][j])>thresh)&&(fabs(a[l][j])<=thresh))
              ALLESKLEINER=FALSE;
        if(ALLESKLEINER)
          {
          anzcandbak=0;
          for(i=0;i<maxdepth;i++)
            anzcandbak+=bigcoeffs(abak[i],n,thresh);
          }
        if((ALLESKLEINER)&&(anzcandbak<anzcand))
          {
          printf("%d bis %d Ausreißer\n",pos,pos+outlierlevel-1);
          for(j=pos;j<=pos+outlierlevel-1;j++)
            {
            FERTIG=FALSE;
            x[j]=xbak[j];
            }
          for(i=0;i<maxdepth;i++)
            swt(x,a[i],n,i);
          anzcand=anzcandbak;
          printf("anzcand: %d\n",anzcand);
          }
        } /* end if(!ALLESKLEINER) */
      } /* end for pos */

    } while(!FERTIG);
  } /* end for outlierlevel */

filezaehler++;
sprintf(name,"robust%d.out",filezaehler);
printf("%s\n",name);
printvector(name,x,n);
} /* end for N */
```

```
  for(i=1;i<=n;i++)
    xbak[i]=x[i];
  do
    {
    printf("j0: ");
    scanf("%d",&j0);
    wt1(x,n,1);
    for(i=zwpo[j0]+1;i<=n;i++)
      if((x[i]<=thresh)&&(-thresh<=x[i]))
        x[i]=0.0;
    wt1(x,n,-1);
    printvector("est.out",x,n);
    for(i=1;i<=n;i++)
      x[i]=xbak[i];
    } while(j0>0);
  } /* main */
```

4.7 Der Alternativ-Algorithmus zu Algorithmus 3.7

Dies ist der Alternativ-Algorithmus zu Algorithmus 3.7, der sich aus Satz 3.4 ergibt. Außerdem wird der Parameter mit dem Algorithmus 3.6 automatisch bestimmt.

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>
#define NMAX 8195

int zwpo[11]={1,2,4,8,16,32,64,128,256,512,1024};
extern int ncof, ioff, joff;
extern double wksp[NMAX+1],cc[NMAX+1],cr[NMAX+1];
extern int maxrunlength(double *x,int n);
extern double MED(double *x,int n);
extern double MAD(double *x, int n);
extern void readdata(double *a,int *n,char *dateiname);
extern void printvector(char *c,double *a,int n);
extern void pwtset(int n);
extern void wt1(double *a,int n,int isign);

void qrdecomp(double *a, double *q, double *r, int n, int k)
  {
  double sum;
  int i, j, l;
  double b[n];

  sum=0.0;
  for(i=0;i<n;i++)
    sum+=a[i*k]*a[i*k];
  r[0]=sqrt(sum);
  for(i=1;i<k;i++)
    r[i*k]=0.0;
  for(i=0;i<n;i++)
    q[i*k]=a[i*k]/r[0];
  for(j=1;j<k;j++)
    {
    for(i=0;i<=j-1;i++)
      {
      r[i*k+j]=0.0;
      for(l=0;l<n;l++)
        r[i*k+j]+=a[l*k+j]*q[l*k+i];
      }
    for(i=0;i<n;i++)
      b[i]=a[i*k+j];
    for(l=0;l<=j-1;l++)
      {
      for(i=0;i<n;i++)
        b[i]-=r[l*k+j]*q[i*k+l];
```

```
      }
    sum=0.0;
    for(i=0;i<n;i++)
      sum+=b[i]*b[i];
    r[j*k+j]=sqrt(sum);
    for(i=j+1;i<k;i++)
      r[i*k+j]=0.0;
    for(i=0;i<n;i++)
      q[i*k+j]=b[i]/r[j*k+j];
    }
  }

void linreg(double *x, double *y, double *beta, int n, int k)
  {
  double *q, *r, b[n];
  int i,j;

  q=(void *)calloc(n*k,sizeof(double));
  r=(void *)calloc(k*k,sizeof(double));

  if((q==NULL)||(r==NULL))
    {
    puts("OUT OF MEMORY!!");
    exit(0L);
    }

  qrdecomp(x,q,r,n,k);

  for(i=0;i<k;i++)
    {
    b[i]=0;
    for(j=0;j<n;j++)
      b[i]+=q[j*k+i]*y[j];
    }
  for(i=k-1;i>=0;i--)
    {
    beta[i]=b[i];
    for(j=i+1;j<k;j++)
      beta[i]-=beta[j]*r[i*k+j];
    beta[i]=beta[i]/r[i*k+i];
    }
  }


main()
  {
  char dummy[80],name[30],outlier[8195];
  double *A,*b;
  double xbak[8195],res[8195],a[8195],x[8195],sigma,t;
  double med,thresh,beta[1000],zeile[8195];
  int d,i,j,k,l,n,j0,N,pos;
  int maxN,fensterbreite,anzkoeffs2;
  int maxmrl,anzoutlier=0;

  printf("Datensatz: ");
  scanf("%s",name);
  gets(dummy);
  readdata(x,&n,name);
  printf("%d Daten gelesen!\n",n);

  printf("maxN: ");
  scanf("%d",&maxN);
  printf("fensterbreite: ");
  scanf("%d",&fensterbreite);

  for(i=1;i<=n;i++)
    outlier[i]=FALSE;

  /* (1) Varianz schätzen */
  sigma=500000000;
  for(N=2;N<=20;N++)
```

```
  {
  printf("N: %d\n",N);
  pwtset(N);
  for(i=1;i<=n;i++)
    xbak[i]=(a[i]=x[i]);
  wt1(a,n,1);
  if(MAD(&(a[n/2+1]),n/2)/0.6745<sigma)
    sigma=MAD(&(a[n/2+1]),n/2)/0.6745;
  }
printf("sigma: %f\n",sigma);

/* (2), (3) Ausreißer mit laufendem Median identifizieren */
for(i=1;i<=n;i++)
  {
  k=i-fensterbreite;
  if(k<1)
    k=1;
  l=i+fensterbreite;
  if(l>n)
    l=n;
  med=MED(&(x[k]),l-k+1);
  if(fabs(x[i]-med)>1.96*sigma)
    {
    outlier[i]=TRUE;
    printf("%d ist Ausreißer\n",i);
    anzoutlier++;
    }
  }

printf("%d Ausreißer gefunden\n",anzoutlier);


/* (4) Daten abändern */

pwtset(maxN);

printf("maxmrl: ");
scanf("%d",&maxmrl);

if(anzoutlier>0)
  {
  anzkoeffs2=1;
  do
    {
    A=(double *)calloc((n-anzoutlier)*(anzkoeffs2),sizeof(double));
    b=(double *)calloc(n-anzoutlier,sizeof(double));
    if((A==NULL)||(b==NULL))
      {
      puts("Out of memory!");
      exit(0L);
      }
    for(i=0;i<anzkoeffs2;i++)
      {
      for(j=1;j<=n;j++)
        zeile[j]=0;
      zeile[i+1]=1;
      wt1(zeile,n,-1);
      pos=0;
      for(j=1;j<=n;j++)
        if(!outlier[j])
          {
          A[pos*(anzkoeffs2)+i]=zeile[j];
          pos++;
          }
      }
    pos=0;
    for(j=1;j<=n;j++)
      if(!outlier[j])
        {
        b[pos]=x[j];
        pos++;
```

```
        }
      linreg(A,b,beta,n-anzoutlier,anzkoeffs2);
      for(j=1;j<=anzkoeffs2;j++)
        zeile[j]=beta[j-1];
      for(j=anzkoeffs2+1;j<=n;j++)
        zeile[j]=0;
      wt1(zeile,n,-1);
      pos=1;
      for(j=1;j<=n;j++)
        if(!outlier[j])
          {
          res[pos]=x[j]-zeile[j];
          pos++;
          }
      anzkoeffs2*=2;
      d--;
      } while(maxrunlength(res,n-anzoutlier)>maxmrl);
    for(i=1;i<=n;i++)
      if(outlier[i])
        x[i]=zeile[i];
    free(A);
    free(b);
    }

  printvector("robust.out",x,n);

  /* (5) Hard-Thresholding anwenden */
  thresh=sqrt(2*log((double)n))*sigma;
  printf("Threshold: %f\n",thresh);
  for(i=1;i<=n;i++)
    xbak[i]=x[i];
  for(j0=3;j0<=6;j0++)
    {
    wt1(x,n,1);
    for(i=zwpo[j0]+1;i<=n;i++)
      if((x[i]>-thresh)||(x[i]<thresh))
        x[i]=0.0;
    wt1(x,n,-1);
    sprintf(name,"est%d.out",j0);
    printvector(name,x,n);
    for(i=1;i<=n;i++)
      x[i]=xbak[i];
    }
  } /* main */
```

4.8 Diskrete lineare l^1-Approximation

Dieser Algorithmus von Ian Barrodale berechnet eine l^1-Lösung des linearen Regressionsproblemes wie in [2] und [3] beschrieben.

```
      SUBROUTINE CL1(K, L, M, N, KLMD, KLM2D, NKLMD, N2D,              CL1   10
     * Q, KODE, TOLER, ITER, X, RES, ERROR, CU, IU, S)
C THIS SUBROUTINE USES A MODIFICATION OF THE SIMPLEX
C METHOD OF LINEAR PROGRAMMING TO CALCULATE AN L1 SOLUTION
C TO A K BY N SYSTEM OF LINEAR EQUATIONS
C             AX=B
C SUBJECT TO L LINEAR EQUALITY CONSTRAINTS
C             CX=D
C AND M LINEAR INEQUALITY CONSTRAINTS
C             EX.LE.F.
C DESCRIPTION OF PARAMETERS
C K       NUMBER OF ROWS OF THE MATRIX A (K.GE.1).
C L       NUMBER OF ROWS OF THE MATRIX C (L.GE.0).
C M       NUMBER OF ROWS OF THE MATRIX E (M.GE.0).
C N       NUMBER OF COLUMNS OF THE MATRICES A,C,E (N.GE.1).
C KLMD    SET TO AT LEAST K+L+M FOR ADJUSTABLE DIMENSIONS.
C KLM2D   SET TO AT LEAST K+L+M+2 FOR ADJUSTABLE DIMENSIONS.
C NKLMD   SET TO AT LEAST N+K+L+M FOR ADJUSTABLE DIMENSIONS.
```

```
C N2D    SET TO AT LEAST N+2 FOR ADJUSTABLE DIMENSIONS
C Q      TWO DIMENSIONAL REAL ARRAY WITH KLM2D ROWS AND
C        AT LEAST N2D COLUMNS.
C        ON ENTRY THE MATRICES A,C AND E, AND THE VECTORS
C        B,D AND F MUST BE STORED IN THE FIRST K+L+M ROWS
C        AND N+1 COLUMNS OF Q AS FOLLOWS
C             A B
C         Q = C D
C             E F
C        THESE VALUES ARE DESTROYED BY THE SUBROUTINE.
C KODE   A CODE USED ON ENTRY TO, AND EXIT
C        FROM, THE SUBROUTINE.
C        ON ENTRY, THIS SHOULD NORMALLY BE SET TO 0.
C        HOWEVER, IF CERTAIN NONNEGATIVITY CONSTRAINTS
C        ARE TO BE INCLUDED IMPLICITLY, RATHER THAN
C        EXPLICITLY IN THE CONSTRAINTS EX.LE.F, THEN KODE
C        SHOULD BE SET TO 1, AND THE NONNEGATIVITY
C        CONSTRAINTS INCLUDED IN THE ARRAYS X AND
C        RES (SEE BELOW).
C        ON EXIT, KODE HAS ONE OF THE
C        FOLLOWING VALUES
C             0- OPTIMAL SOLUTION FOUND,
C             1- NO FEASIBLE SOLUTION TO THE
C                CONSTRAINTS,
C             2- CALCULATIONS TERMINATED
C                PREMATURELY DUE TO ROUNDING ERRORS,
C             3- MAXIMUM NUMBER OF ITERATIONS REACHED.
C TOLER  A SMALL POSITIVE TOLERANCE. EMPIRICAL
C        EVIDENCE SUGGESTS TOLER = 10**(-D*2/3),
C        WHERE D REPRESENTS THE NUMBER OF DECIMAL
C        DIGITS OF ACCURACY AVAILABLE. ESSENTIALLY,
C        THE SUBROUTINE CANNOT DISTINGUISH BETWEEN ZERO
C        AND ANY QUANTITY WHOSE MAGNITUDE DOES NOT EXCEED
C        TOLER. IN PARTICULAR, IT WILL NOT PIVOT ON ANY
C        NUMBER WHOSE MAGNITUDE DOES NOT EXCEED TOLER.
C ITER   ON ENTRY ITER MUST CONTAIN AN UPPER BOUND ON
C        THE MAXIMUM NUMBER OF ITERATIONS ALLOWED.
C        A SUGGESTED VALUE IS 10*(K+L+M). ON EXIT ITER
C        GIVES THE NUMBER OF SIMPLEX ITERATIONS.
C X      ONE DIMENSIONAL REAL ARRAY OF SIZE AT LEAST N2D.
C        ON EXIT THIS ARRAY CONTAINS A
C        SOLUTION TO THE L1 PROBLEM. IF KODE=1
C        ON ENTRY, THIS ARRAY IS ALSO USED TO INCLUDE
C        SIMPLE NONNEGATIVITY CONSTRAINTS ON THE
C        VARIABLES. THE VALUES -1, 0, OR 1
C        FOR X(J) INDICATE THAT THE J-TH VARIABLE
C        IS RESTRICTED TO BE .LE.0, UNRESTRICTED,
C        OR .GE.0 RESPECTIVELY.
C RES    ONE DIMENSIONAL REAL ARRAY OF SIZE AT LEAST KLMD.
C        ON EXIT THIS CONTAINS THE RESIDUALS B-AX
C        IN THE FIRST K COMPONENTS, D-CX IN THE
C        NEXT L COMPONENTS (THESE WILL BE =0),AND
C        F-EX IN THE NEXT M COMPONENTS. IF KODE=1 ON
C        ENTRY, THIS ARRAY IS ALSO USED TO INCLUDE SIMPLE
C        NONNEGATIVITY CONSTRAINTS ON THE RESIDUALS
C        B-AX. THE VALUES -1, 0, OR 1 FOR RES(I)
C        INDICATE THAT THE I-TH RESIDUAL (1.LE.I.LE.K) IS
C        RESTRICTED TO BE .LE.0, UNRESTRICTED, OR .GE.0
C        RESPECTIVELY.
C ERROR  ON EXIT, THIS GIVES THE MINIMUM SUM OF
C        ABSOLUTE VALUES OF THE RESIDUALS.
C CU     A TWO DIMENSIONAL REAL ARRAY WITH TWO ROWS AND
C        AT LEAST NKLMD COLUMNS USED FOR WORKSPACE.
C IU     A TWO DIMENSIONAL INTEGER ARRAY WITH TWO ROWS AND
C        AT LEAST NKLMD COLUMNS USED FOR WORKSPACE.
C S      INTEGER ARRAY OF SIZE AT LEAST KLMD, USED FOR
C        WORKSPACE.
C IF YOUR FORTRAN COMPILER PERMITS A SINGLE COLUMN OF A TWO
C DIMENSIONAL ARRAY TO BE PASSED TO A ONE DIMENSIONAL ARRAY
C THROUGH A SUBROUTINE CALL, CONSIDERABLE SAVINGS IN
C EXECUTION TIME MAY BE ACHIEVED THROUGH THE USE OF THE
```

```
C FOLLOWING SUBROUTINE, WHICH OPERATES ON COLUMN VECTORS.
C     SUBROUTINE COL(V1, V2, XMLT, NOTROW, K)
C THIS SUBROUTINE ADDS TO THE VECTOR V1 A MULTIPLE OF THE
C VECTOR V2 (ELEMENTS 1 THROUGH K EXCLUDING NOTROW).
C     DIMENSION V1(K), V2(K)
C     KEND = NOTROW - 1
C     KSTART = NOTROW + 1
C     IF (KEND .LT. 1) GO TO 20
C     DO 10 I=1,KEND
C        V1(I) = V1(I) + XMLT*V2(I)
C  10 CONTINUE
C     IF(KSTART .GT. K) GO TO 40
C  20 DO 30 I=KSTART,K
C       V1(I) = V1(I) + XMLT*V2(I)
C  30 CONTINUE
C  40 RETURN
C     END
C SEE COMMENTS FOLLOWING STATEMENT LABELLED 440 FOR
C INSTRUCTIONS ON THE IMPLEMENTATION OF THIS MODIFICATION.
      DOUBLE PRECISION SUM
      DOUBLE PRECISION DBLE
      REAL Q, X, Z, CU, SN, ZU, ZV, CUV, RES, XMAX, XMIN,
     * ERROR, PIVOT, TOLER, TPIVOT
      REAL ABS
      INTEGER I, J, K, L, M, N, S, IA, II, IN, IU, JS, KK,
     * NK, N1, N2, JMN, JPN, KLM, NKL, NK1, N2D, IIMN,
     * IOUT, ITER, KLMD, KLM1, KLM2, KODE, NKLM, NKL1,
     * KLM2D, MAXIT, NKLMD, IPHASE, KFORCE, IINEG
      INTEGER IABS
      DIMENSION Q(KLM2D,N2D), X(N2D), RES(KLMD),
     * CU(2,NKLMD), IU(2,NKLMD), S(KLMD)
C
C INITIALIZATION.
C
      MAXIT = ITER
      N1 = N + 1
      N2 = N + 2
      NK = N + K
      NK1 = NK + 1
      NKL = NK + L
      NKL1 = NKL + 1
      KLM = K + L + M
      KLM1 = KLM + 1
      KLM2 = KLM + 2
      NKLM = N + KLM
      KFORCE = 1
      ITER = 0
      JS = 1
      IA = 0
C SET UP LABELS IN Q.
      DO 10 J=1,N
         Q(KLM2,J) = J
   10 CONTINUE
      DO 30 I=1,KLM
         Q(I,N2) = N + I
         IF (Q(I,N1).GE.0.) GO TO 30
         DO 20 J=1,N2
            Q(I,J) = -Q(I,J)
   20    CONTINUE
   30 CONTINUE
C SET UP PHASE 1 COSTS.
      IPHASE = 2
      DO 40 J=1,NKLM
         CU(1,J) = 0.
         CU(2,J) = 0.
         IU(1,J) = 0
         IU(2,J) = 0
   40 CONTINUE
      IF (L.EQ.0) GO TO 60
      DO 50 J=NK1,NKL
         CU(1,J) = 1.
```

```
         CU(2,J) = 1.
         IU(1,J) = 1
         IU(2,J) = 1
   50 CONTINUE
      IPHASE = 1
   60 IF (M.EQ.0) GO TO 80
      DO 70 J=NKL1,NKLM
         CU(2,J) = 1.
         IU(2,J) = 1
         JMN = J - N
         IF (Q(JMN,N2).LT.0.) IPHASE = 1
   70 CONTINUE
   80 IF (KODE.EQ.0) GO TO 150
      DO 110 J=1,N
         IF (X(J)) 90, 110, 100
   90    CU(1,J) = 1.
         IU(1,J) = 1
         GO TO 110
  100    CU(2,J) = 1.
         IU(2,J) = 1
  110 CONTINUE
      DO 140 J=1,K
         JPN = J + N
         IF (RES(J)) 120, 140, 130
  120    CU(1,JPN) = 1.
         IU(1,JPN) = 1
         IF (Q(J,N2).GT.0.0) IPHASE = 1
         GO TO 140
  130    CU(2,JPN) = 1.
         IU(2,JPN) = 1
         IF (Q(J,N2).LT.0.0) IPHASE = 1
  140 CONTINUE
  150 IF (IPHASE.EQ.2) GO TO 500
C COMPUTE THE MARGINAL COSTS.
  160 DO 200 J=JS,N1
         SUM = 0.D0
         DO 190 I=1,KLM
            II = Q(I,N2)
            IF (II.LT.0) GO TO 170
            Z = CU(1,II)
            GO TO 180
  170       IINEG = -II
            Z = CU(2,IINEG)
  180       SUM = SUM + DBLE(Q(I,J))*DBLE(Z)
  190    CONTINUE
         Q(KLM1,J) = SUM
  200 CONTINUE
      DO 230 J=JS,N
         II = Q(KLM2,J)
         IF (II.LT.0) GO TO 210
         Z = CU(1,II)
         GO TO 220
  210    IINEG = -II
         Z = CU(2,IINEG)
  220    Q(KLM1,J) = Q(KLM1,J) - Z
  230 CONTINUE
C DETERMINE THE VECTOR TO ENTER THE BASIS.
  240 XMAX = 0.
      IF (JS.GT.N) GO TO 490
      DO 280 J=JS,N
         ZU = Q(KLM1,J)
         II = Q(KLM2,J)
         IF (II.GT.0) GO TO 250
         II = -II
         ZV = ZU
         ZU = -ZU - CU(1,II) - CU(2,II)
         GO TO 260
  250    ZV = -ZU - CU(1,II) - CU(2,II)
  260    IF (KFORCE.EQ.1 .AND. II.GT.N) GO TO 280
         IF (IU(1,II).EQ.1) GO TO 270
         IF (ZU.LE.XMAX) GO TO 270
```

```
         XMAX = ZU
         IN = J
  270    IF (IU(2,II).EQ.1) GO TO 280
         IF (ZV.LE.XMAX) GO TO 280
         XMAX = ZV
         IN = J
  280 CONTINUE
      IF (XMAX.LE.TOLER) GO TO 490
      IF (Q(KLM1,IN).EQ.XMAX) GO TO 300
      DO 290 I=1,KLM2
         Q(I,IN) = -Q(I,IN)
  290 CONTINUE
      Q(KLM1,IN) = XMAX
C DETERMINE THE VECTOR TO LEAVE THE BASIS.
  300 IF (IPHASE.EQ.1 .OR. IA.EQ.0) GO TO 330
      XMAX = 0.
      DO 310 I=1,IA
         Z = ABS(Q(I,IN))
         IF (Z.LE.XMAX) GO TO 310
         XMAX = Z
         IOUT = I
  310 CONTINUE
      IF (XMAX.LE.TOLER) GO TO 330
      DO 320 J=1,N2
         Z = Q(IA,J)
         Q(IA,J) = Q(IOUT,J)
         Q(IOUT,J) = Z
  320 CONTINUE
      IOUT = IA
      IA = IA - 1
      PIVOT = Q(IOUT,IN)
      GO TO 420
  330 KK = 0
      DO 340 I=1,KLM
         Z = Q(I,IN)
         IF (Z.LE.TOLER) GO TO 340
         KK = KK + 1
         RES(KK) = Q(I,N1)/Z
         S(KK) = I
  340 CONTINUE
  350 IF (KK.GT.0) GO TO 360
      KODE = 2
      GO TO 590
  360 XMIN = RES(1)
      IOUT = S(1)
      J = 1
      IF (KK.EQ.1) GO TO 380
      DO 370 I=2,KK
         IF (RES(I).GE.XMIN) GO TO 370
         J = I
         XMIN = RES(I)
         IOUT = S(I)
  370 CONTINUE
      RES(J) = RES(KK)
      S(J) = S(KK)
  380 KK = KK - 1
      PIVOT = Q(IOUT,IN)
      II = Q(IOUT,N2)
      IF (IPHASE.EQ.1) GO TO 400
      IF (II.LT.0) GO TO 390
      IF (IU(2,II).EQ.1) GO TO 420
      GO TO 400
  390 IINEG = -II
      IF (IU(1,IINEG).EQ.1) GO TO 420
  400 II = IABS(II)
      CUV = CU(1,II) + CU(2,II)
      IF (Q(KLM1,IN)-PIVOT*CUV.LE.TOLER) GO TO 420
C BYPASS INTERMEDIATE VERTICES.
      DO 410 J=JS,N1
         Z = Q(IOUT,J)
         Q(KLM1,J) = Q(KLM1,J) - Z*CUV
```

```
         Q(IOUT,J) = -Z
  410 CONTINUE
      Q(IOUT,N2) = -Q(IOUT,N2)
      GO TO 350
C GAUSS-JORDAN ELIMINATION.
  420 IF (ITER.LT.MAXIT) GO TO 430
      KODE = 3
      GO TO 590
  430 ITER = ITER + 1
      DO 440 J=JS,N1
         IF (J.NE.IN) Q(IOUT,J) = Q(IOUT,J)/PIVOT
  440 CONTINUE
C IF PERMITTED, USE SUBROUTINE COL OF THE DESCRIPTION
C SECTION AND REPLACE THE FOLLOWING SEVEN STATEMENTS DOWN
C TO AND INCLUDING STATEMENT NUMBER 460 BY..
C     DO 460 J=JS,N1
C        IF(J .EQ. IN) GO TO 460
C        Z = -Q(IOUT,J)
C        CALL COL(Q(1,J), Q(1,IN), Z, IOUT, KLM1)
C 460 CONTINUE
      DO 460 J=JS,N1
         IF (J.EQ.IN) GO TO 460
         Z = -Q(IOUT,J)
         DO 450 I=1,KLM1
            IF (I.NE.IOUT) Q(I,J) = Q(I,J) + Z*Q(I,IN)
  450    CONTINUE
  460 CONTINUE
      TPIVOT = -PIVOT
      DO 470 I=1,KLM1
         IF (I.NE.IOUT) Q(I,IN) = Q(I,IN)/TPIVOT
  470 CONTINUE
      Q(IOUT,IN) = 1./PIVOT
      Z = Q(IOUT,N2)
      Q(IOUT,N2) = Q(KLM2,IN)
      Q(KLM2,IN) = Z
      II = ABS(Z)
      IF (IU(1,II).EQ.0 .OR. IU(2,II).EQ.0) GO TO 240
      DO 480 I=1,KLM2
         Z = Q(I,IN)
         Q(I,IN) = Q(I,JS)
         Q(I,JS) = Z
  480 CONTINUE
      JS = JS + 1
      GO TO 240
C TEST FOR OPTIMALITY.
  490 IF (KFORCE.EQ.0) GO TO 580
      IF (IPHASE.EQ.1 .AND. Q(KLM1,N1).LE.TOLER) GO TO 500
      KFORCE = 0
      GO TO 240
C SET UP PHASE 2 COSTS.
  500 IPHASE = 2
      DO 510 J=1,NKLM
         CU(1,J) = 0.
         CU(2,J) = 0.
  510 CONTINUE
      DO 520 J=N1,NK
         CU(1,J) = 1.
         CU(2,J) = 1.
  520 CONTINUE
      DO 560 I=1,KLM
         II = Q(I,N2)
         IF (II.GT.0) GO TO 530
         II = -II
         IF (IU(2,II).EQ.0) GO TO 560
         CU(2,II) = 0.
         GO TO 540
  530    IF (IU(1,II).EQ.0) GO TO 560
         CU(1,II) = 0.
  540    IA = IA + 1
         DO 550 J=1,N2
            Z = Q(IA,J)
```

```
            Q(IA,J) = Q(I,J)
            Q(I,J) = Z
  550    CONTINUE
  560 CONTINUE
      GO TO 160
  570 IF (Q(KLM1,N1).LE.TOLER) GO TO 500
      KODE = 1
      GO TO 590
  580 IF (IPHASE.EQ.1) GO TO 570
C PREPARE OUTPUT.
      KODE = 0
  590 SUM = 0.D0
      DO 600 J=1,N
         X(J) = 0.
  600 CONTINUE
      DO 610 I=1,KLM
         RES(I) = 0.
  610 CONTINUE
      DO 640 I=1,KLM
         II = Q(I,N2)
         SN = 1.
         IF (II.GT.0) GO TO 620
         II = -II
         SN = -1.
  620    IF (II.GT.N) GO TO 630
         X(II) = SN*Q(I,N1)
         GO TO 640
  630    IIMN = II - N
         RES(IIMN) = SN*Q(I,N1)
         IF (II.GE.N1 .AND. II.LE.NK) SUM = SUM +
     *    DBLE(Q(I,N1))
  640 CONTINUE
      ERROR = SUM
      RETURN
      END
```

4.9 Algorithmus 3.7

```
#include <math.h>
#include <stdio.h>
#include <curses.h>
#include <stdlib.h>
#define NMAX 8195

int zwpo[15]={1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384};
extern int ncof, ioff, joff;
extern double wksp[NMAX+1],cc[NMAX+1],cr[NMAX+1];

extern void cl1_();
void MAIN__()
  {
  }

extern void pwtset(int n);
extern void wt1(double *a,int n,int isign);
extern void readdata(double *a,int *n,char *dateiname);
extern void printvector(char *c,double *a,int n);

int countbigruns(double *res, int n, int laenge)
  {
  int i,j,count=0,neu_n=0;
  char RUN;
  double neures[n+1];

  for(i=1;i<=n;i++)
    if(fabs(res[i])>0.001)
      neures[++neu_n]=res[i];
  for(i=1;i<=neu_n-laenge+1;i++)
```

```
    {
    if(neures[i]>0.001)
      {
      RUN=TRUE;
      for(j=i+1;j<=i+laenge-1;j++)
        if(neures[j]<=0.001)
          RUN=FALSE;
      }
    else
    if(neures[i]<-0.001)
      {
      RUN=TRUE;
      for(j=i+1;j<=i+laenge-1;j++)
        if(neures[j]>=-0.001)
          RUN=FALSE;
      }
    else
      RUN=FALSE;
    if(RUN)
      {
      count++;
      }
    }
  return count;
  }

main()
  {
  double x[8195],t;
  double wv[8195],est[8195],res[8195],p;
  double m,actl1norm;
  double l1normlinks,l1normrechts;
  double minimum,oldminimum,l1norm,unteregrenze;
  char dummychar, dummy[80],name[200],dabei[8195];
  int klinks,krechts,zaehler=0;
  int pos,ii,i2max,imax=1,i,k,n,nrwav,j,level,i2;
  int mmrl,testcbg,cbg,wvlength,s[8195],iu[17000];
  int cl1k,cl1l,cl1m,cl1n,klmd,klm2d,nklmd,n2d,kode,iter;
  float toler=0.0001, error,resmax,*Q,cl1x[8195],cl1xbak[8195],cl1res[8195],cu[17000];
  int pos2,nochdabei,mini,mmm,mm;

  printf("Datensatz: ");
  scanf("%s",name);
  gets(dummy);
  readdata(x,&n,name);
  printf("%d Daten gelesen!\n",n);
  cl1k=n;
  cl1l=0;
  cl1m=0;
  klmd=n;
  klm2d=n+2;
  kode=0;
  printf("Welches Wavelet (4,10,12)? ");
  scanf("%d",&nrwav);
  printf("l: ");
  scanf("%ld",&mmrl);
  pwtset(nrwav);
  mmrl++;
  actl1norm=0;
  for(i=1;i<=n;i++)
    {
    res[i]=x[i];
    est[i]=0;
    actl1norm+=fabs(res[i]);
    }
  do
    {
    cl1n=imax;
    printf("imax ist %d\n",imax);
    nklmd=imax+n;
    n2d=imax+2;
```

```
  Q=(float *)malloc(klm2d*n2d*sizeof(float));
  if(Q==NULL)
    {
    puts("Not enough memory");
    exit(0L);
    }
  pos=0;
  for(ii=1;ii<=imax;ii++)
    {
    for(j=1;j<=n;j++)
      wv[j]=0;
    wv[ii]=1;
    wt1(wv,n,-1);
    for(j=1;j<=n;j++)
      Q[pos++]=wv[j];
    Q[pos++]=0.0;
    Q[pos++]=0.0;
    }
  for(j=1;j<=n;j++)
    Q[pos++]=x[j];
  iter=20*n;

  cl1_(&cl1k,&cl1l,&cl1m,&cl1n,&klmd,&klm2d,&nklmd,&n2d,Q,&kode,
                           &toler,&iter,cl1x,cl1res,&error,cu,iu,s);
        for(j=0;j<nochdabei;j++)
          cl1xbak[j]=cl1x[j];
  free(Q);

  for(j=1;j<=n;j++)
    {
    res[j]=cl1res[j-1];
    est[j]=x[j]-res[j];
    }

  cbg=countbigruns(res,n,mmrl);
  printf("testcbg: %d\n",cbg);
  printvector("est.out",est,n);
  imax*=2;
  } while(cbg>0);
imax=imax/2;
for(i=1;i<=imax;i++)
  dabei[i]=TRUE;
nochdabei=imax;
mmm=0;
while(mmm<=nochdabei)
  {
  do
    {
    /* Finde kleinsten Koeffizienten */
    unteregrenze=0.0;
    for(mm=0;mm<=mmm;mm++)
    {
    minimum=1000000000000.0;
    for(i=0;i<nochdabei;i++)
      {
      if((fabs(cl1x[i])<minimum)&&(unteregrenze+0.000001<fabs(cl1x[i])))
        {
        mini=i;
        minimum=fabs(cl1x[i]);
        }
      }
    unteregrenze=minimum;
    }
    pos=0;
    i=-1;
    do
      {
      pos++;
      if(dabei[pos])
        i++;
```

```
      } while(i<mini);
    nochdabei--;
    dabei[pos]=FALSE;

    printf("I will try to eliminate number %d\n",pos);

    cl1n=nochdabei;
    printf("nochdabei ist %d\n",nochdabei);
    nklmd=nochdabei+n;
    n2d=nochdabei+2;

    Q=(float *)malloc(klm2d*n2d*sizeof(float));
    if(Q==NULL)
      {
      puts("Not enough memory");
      exit(0L);
      }
    pos2=0;
    for(ii=1;ii<=imax;ii++)
      {
      if(dabei[ii])
        {
        for(j=1;j<=n;j++)
          wv[j]=0;
        wv[ii]=1;
        wt1(wv,n,-1);
        for(j=1;j<=n;j++)
          Q[pos2++]=wv[j];
        Q[pos2++]=0.0;
        Q[pos2++]=0.0;
        }
      }
    for(j=1;j<=n;j++)
      Q[pos2++]=x[j];
    iter=20*n;

    cl1_(&cl1k,&cl1l,&cl1m,&cl1n,&klmd,&klm2d,&nklmd,&n2d,Q,&kode,
                            &toler,&iter,cl1x,cl1res,&error,cu,iu,s);
    free(Q);

    for(j=1;j<=n;j++)
      {
      res[j]=cl1res[j-1];
      est[j]=x[j]-res[j];
      }

    cbg=countbigruns(res,n,mmrl);
    printf("testcbg: %d\n",cbg);
    if(cbg==0)
      {
      printvector("est.out",est,n);
      mmm=0;
      for(j=0;j<nochdabei;j++)
        cl1xbak[j]=cl1x[j];
      }
    else
      {
      mmm++;
      dabei[pos]=TRUE;
      nochdabei++;
      for(j=0;j<nochdabei;j++)
        cl1x[j]=cl1xbak[j];
      }
    } while(cbg==0);
  } /* while(mmm<=nochdabei) */
} /* main */
```

Literatur

[1] F. Abramovich and Y. Benjamini, *Thresholding of wavelet coefficients as multiple hypotheses testing procedure*, Wavelets and Statistics (A. Antoniadis and G. Oppenheim, eds.), Springer, Heidelberg, 1995, pp. 6–14.

[2] I. Barrodale and F. D. K. Roberts, *An improved algorithm for discrete l_1 linear approximation*, SIAM J. Numer. Anal. **10** (1973), no. 5, 839–848.

[3] ———, *Solution of an overdetermined system of equations in the l^1-norm*, Comm. of the ACM **17** (1974), 319–320.

[4] C. De Boor, *A practical guide to splines*, Springer-Verlag, New York, 1978.

[5] C. K. Chui, *An introduction to wavelets*, Academic Press, Inc., San Diego, 1992.

[6] R. R. Coifman and D. L. Donoho, *The stationary wavelet transform and some statistical applications*, Wavelets and Statistics (A. Antoniadis and G. Oppenheim, eds.), Springer, Heidelberg, 1995, pp. 6–14.

[7] I. Daubechies, *Orthonormal bases of compactly supported wavelets*, Comms Pure Appl. Math. **41** (1988), 909–996.

[8] ———, *Ten lectures on wavelets*, SIAM, Philadelphia, 1992.

[9] P. L. Davies, *Data features*, Statistica Neerlandica **49** (1995), 185–245.

[10] P. L. Davies and U. Gather, *The identification of multiple outliers*, J. Amer. Statist. Soc. **88** (1993), 782–801.

[11] B. Delyon and A. Juditsky, *Estimating wavelet coefficients*, Wavelets and Statistics (A. Antoniadis and G. Oppenheim, eds.), Springer, Heidelberg, 1995, pp. 151–167.

[12] D. L. Donoho, *Interpolating wavelet transforms*, Tech. report, Department of Statistics, Stanford University, Stanford, 1992.

[13] ———, *Nonlinear wavelet methods for recovery of signals, images, and densities from noisy and incomplete data*, Different Perspectives on Wavelets (I. Daubechies, ed.), American Mathematical Society, 1993, pp. 173–205.

[14] ———, *De-noising via soft-thresholding*, IEEE Trans. Info. Thry (1995).

[15] D. L. Donoho and I. M. Johnstone, *Ideal spatial adaption by wavelet shrinkage*, Biometrika **81** (1994), 425–455.

[16] ———, *Adapting to unknown smoothness via wavelet shrinkage*, J. Am. Statist. Ass. (1995), to appear.

[17] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard, *Wavelet shrinkage: asymptopia?*, J. Roy. Statist. Soc. Ser. B **57** (1995), 371–394.

[18] R.L. Eubank, *Spline smoothing and nonparametric regression*, Marcel Dekker Inc., New York, 1988.

[19] J. Fan and I. Gijbels, *Data-driven bandwidth selection in local polynomial fitting: Variable bandwidth and spatial adaption*, J. Roy. Statist. Soc. Ser. B **57** (1995), 301–370.

[20] W. Feller, *An introduction to probability theory and its applications*, vol. 1, New York, 1967.

[21] Th. Gasser and H. G. Müller, *Kernel estimation of regression functions*, Smoothing Techniques for Curve Estimation (Th. Gasser and M. Rosenblatt, eds.), Springer, Heidelberg, 1979, pp. 23–68.

[22] A. Haar, *Zur theorie der orthogonalen funktionensysteme*, Math. Ann. **69** (1910), 331–371.

[23] Y. Meyer, *Wavelets and operators*, Cambridge University Press, Cambridge, 1992.

[24] G. Nason and B. W. Silverman, *An introduction to wavelets and their statistical uses*, Vortrag bei einer Konferenz der Royal Statistical Society im September 1994 in Newcastle, September 1994.

[25] G. P. Nason, *Wavelet function estimation using cross-validation*, Zur Veröffentlichung eingereicht (1994).

[26] G. P. Nason and B. W. Silverman, *The discrete wavelet transform in s*, Journal of Computational and Graphical Statistics **3** (1994), 163–191.

[27] ———, *The stationary wavelet transform and some statistical applications*, Wavelets and Statistics (A. Antoniadis and G. Oppenheim, eds.), Springer, Heidelberg, 1995, pp. 6–14.

[28] W. H. Press, *Wavelet transforms*, Harvard-Smithsonian Center for Astrophysics Preprint (1991), no. 3184.

[29] B. W. Silverman, *Some aspects of the spline smoothing approach to non-parametric regression curve fitting (with discussion)*, J. Roy. Statist. Soc. Ser. B **47** (1982), 1–52.

[30] M. J. T. Smith and S. L. Eddins, *Analysis/synthesis techniques for subband image coding*, IEEE Transactions on Acoustics, Speech and Signal Processing **38** (1990), 1446–1456.

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

[32] W. Swelden, *The construction and application of wavelets in numerical analysis*, Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, 1995.

[33] B. Vidakovich, *Nonlinear wavelet shrinkage with bayes rules and bayes factors*, Zur Veröffentlichung eingereicht (1994).