Universität Stuttgart, Geodätisches Institut

Diplomarbeit

Wavelet Toolbox in JAVA

Matthias Wengert
Matr.-Nr.: 1740071

Prüfer und Betreuer: Prof. Dr.-Ing. W.Keller

März 2002

Inhaltsverzeichnis

Abbildungsverzeichnis

Notationen

N	Menge der natürlichen Zahlen
Z	Menge der ganzen Zahlen
R	Menge der reellen Zahlen
$\mathbf{R}^*$	$\mathbf{R}\backslash\{0\}$
C	Körper der komplexen Zahlen
$L^2(R)$	$L^2(R) = \{ f : R \to C \mid \int_R \lvert f(x) \rvert^2 \, dx < \infty \}$
$G[f](\omega,t)$	gefensterte Fourier-Transformation
$W_\psi[f](a,b)$	Wavelet-Transformierte zum Wavelet ψ
$< \cdot , \cdot >, \lVert \cdot \rVert$	Skalarprodukt bzw. Norm
$\hat{f}$	Fourier-Transformierte von f
δ_{kj}	Kronecker-Tensor $\delta_{kj} = \begin{cases} 1, \forall k = j \\ 0, \forall k \neq j \end{cases}$

Kapitel 1

Einleitung

Die Theorie der Wavelets ist ein verhältnismäßig junges Forschungsgebiet. Aus ganz verschiedenen Anwendungsbereichen wuchs zu Beginn der 1980er Jahre das Interesse an Transformationen, mit denen Signale in geeigneter Weise analysiert und manipuliert werden können. Das Ziel einer jeden Transformation ist es, Daten in eine andere Repräsentation zu bringen, um daraus Vorteile für anschließende Operationen zu gewinnen. Transformationen sollen Informationen aus einem Signal extrahieren, die aus der ursprünglichen Amplituden-Zeit-Darstellung des Signals nicht unmittelbar zu erkennen sind. Welche Transformation für eine spezielle Anwendung geeignet ist, hängt von der Art der Information ab, an der man interessiert ist.

Das klassische Verfahren der Frequenzanalyse ist die Fourier-Transformation. Sie zerlegt Signale in Sinus- und Kosinusschwingungen, die unendlich lang mit derselben Periode schwingen. Daraus resultieren wesentliche Nachteile in signaltheoretischer Hinsicht. Ein Mangel der Fourier-Transformation liegt darin, dass sie nur unzureichend die lokalen Eigenschaften eines Signals berücksichtigt. Die getrennte Darstellung von Zeit und Frequenz erweist sich als weiterer Nachteil der Fourier-Transformation. Die Fourier-Transformiert gibt Auskunft über die im Signal enthaltenen Frequenzen. Wann diese vorkommen, vermag sie allerdings nicht zu deuten.
Die Wavelet-Transformation verwendet keine unendlich langen Wellen als analysierende Funktionen, sondern endliche, kleine Wellen – die sogenannten Wavelets. Die erwähnten Nachteile der Fourier-Transformation können dadurch weitestgehend behoben werden. In der Literatur war das Prinzip der kontinuierliche Wavelet-Transformation schon längere Zeit bekannt, doch für praktische Anwendungen kam der Durchbruch erst mit der diskreten Variante.

Hauptbestandteil dieser Diplomarbeit sind zwei Computerprogramme, mit denen Signale mittels Wavelet-Transformation analysiert und auch manipuliert werden können. Das eine verwendet die kontinuierliche, das andere die diskrete Wavelet-Transformation. Hierfür wurde die objektorientierte Programmiersprache Java verwendet. Obwohl Java erst 1996 auf den Markt kam, wurde es binnen kurzer Zeit sehr bekannt. Java ist plattformunabhängig und wurde dadurch zu einer Sprache des Internets. Kleine Java-Programme – sogenannte Applets – können in Internetseiten eingebunden werden, und geben so dem Anwender die Möglichkeit mit ihnen interaktiv zu agieren.

Kapitel 2

Hilbert-Räume

Da man sich einen unendlichdimensionalen Vektorraum - wie der Hilbert-Raum einer ist – nur schwer vorstellen kann, wird hier an den dreidimensionalen Vektorraum erinnert. Die wichtigsten Eigenschaften sind kurz zusammengefasst:

- ein Vektor im Raum ist durch drei Komponenten gegeben $\vec{v} = (v_1, v_2, v_3)$
- es ist ein Skalarprodukt zwischen zwei Vektoren definiert: $\langle \vec{v}, \vec{w} \rangle = v_1 w_1 + v_2 w_2 + v_3 w_3$
- die Länge $|\vec{v}|$ eines Vektors kann aus dem Skalarprodukt abgeleitet werden $|\vec{v}| = \sqrt{\langle \vec{v}, \vec{v} \rangle}$
- drei Vektoren $\vec{e}_1, \vec{e}_2, \vec{e}_3$ bilden eine Orthonormalbasis, wenn sie alle die Länge 1 haben und paarweise orthogonal sind, d.h. $\langle \vec{e}_i, \vec{e}_j \rangle = \delta_{ij}$
- jeder Vektor kann als Linearkombination der drei Basisvektoren dargestellt werden $\vec{v} = v_1 \vec{e}_1 + v_2 \vec{e}_2 + v_3 \vec{e}_3$
- als Koordinaten eines Vektors werden die Projektionen auf die jeweiligen Basisvektoren bezeichnet $v_i = \langle \vec{v}, \vec{e}_i \rangle$

Die Theorie der Hilbert-Räume verallgemeinert die Eigenschaften des dreidimensionalen Raums. Die Elemente des Hilbert-Raums müssen keine Vektoren mehr sein und die Dimension ist i.a. nicht mehr endlich.
Ein für die Signalverarbeitung wichtiger Raum ist der Raum der quadratisch integrierbaren Funktionen. Man spricht auch von Signalen mit endlicher Energie und verwendet für die Menge dieser Signale die Bezeichnung $L^2(\mathbf{R})$. $L^2(\mathbf{R})$ ist ein Hilbert-Raum, dessen Elemente die Signale sind. Im Gegensatz zum dreidimensionalen Raum ist der $L^2(\mathbf{R})$ ein unendlichdimensionaler Raum. Eine Basis des $L^2(\mathbf{R})$ hat deshalb unendlich viele Elemente. Ein Signal $f \in L^2(\mathbf{R})$ lässt sich als Vektor mit unendlich vielen Komponenten f(t), $t \in \mathbf{R}$ auffassen. Für die Definition des Skalarprodukts zweier Signale f und g folgt deshalb

$$\langle f(t), g(t) \rangle = \int_R f(t) g(t) dt .$$

Die folgenden Definitionen und Sätze sollen die wichtigsten Eigenschaften der Hilbert-Räume kurz und knapp festhalten.

Definition (Vektorraum):
Ein Tripel $(V,+,\cdot)$, bestehend aus einer Menge V, einer Abbildung $+: V \times V \to V$, $(x,y) \to x+y$, und einer Abbildung $\bullet: \mathbf{C} \times V \to V$, $(\alpha,x) \to \alpha x$, heißt *komplexer Vektorraum*, wenn die folgenden acht Axiome gelten:

(1) $(x+y)+z = x+(y+z)$, $x,y,z \in V$

(2) $x+y = y+x$, $x,y \in V$

(3) Es gibt ein Element $0 \in V$, so dass für alle $x \in V$ gilt: $x+0 = x$

(4) Zu jedem $x \in V$ gibt es ein inverses Element $-x \in V$ mit $x + (-x) = 0$

(5) $\alpha(\beta x) = (\alpha\beta)x, \qquad \alpha, \beta \in \mathbf{C}, x \in V$

(6) $1x = x, \qquad x \in V$

(7) $\alpha(x+y) = \alpha x + \beta x, \qquad \alpha \in \mathbf{C}, x, y \in V$

(8) $(\alpha + \beta)x = \alpha x + \beta x, \quad \alpha, \beta \in \mathbf{C}, x \in V$

Definition (Prä-Hilbert-Raum):
Ein komplexer Vektorraum V zusammen mit einem Skalarprodukt heißt Prä-Hilbert-Raum

$$\langle \cdot,\cdot \rangle : V \times V \to \mathbf{C}.$$

Definition (Skalarprodukt):
Eine Abbildung $\langle \cdot,\cdot \rangle : V \times V \to \mathbf{C}$ heißt *Skalarprodukt*, wenn es die folgenden Bedingungen erfüllt:

$$\langle x,y \rangle = \langle y,x \rangle, \qquad x,y \in V$$

$$\langle x+y,z \rangle = \langle x,y \rangle + \langle x,z \rangle, \quad x,y,z \in V$$

$$\langle \alpha x,y \rangle = \alpha \langle x,y \rangle, \quad x,y \in V, \alpha \in \mathbf{C}$$

$\langle x,x \rangle \geq 0$, und es gilt $\langle x,x \rangle = 0 \Leftrightarrow x=0$

Definition (Norm):
Die Abbildung $\| \cdot \| : V \to \mathbf{C}$ ist definiert durch

$$||x|| := \sqrt{\langle x, x \rangle}$$

und wird als *Norm* von bezeichnet.

Satz (Ungleichung von Cauchy-Schwarz):
In einem Prä-Hilbert-Raum V gilt für das Skalarprodukt zweier Elemente $x,y \in V$ die Abschätzung von Cauchy-Schwarz

$$|\langle x,y \rangle|^2 \leq \|x\|^2 \, \|y\|^2.$$

Definition (Konvergenz):
Eine Folge $\{x_n\} \subset V$ *konvergiert* gegen $x \in V$, wenn für jedes $\varepsilon > 0$ ein Index $n_0 = n_0(\varepsilon)$ existiert, mit

$$||x_n - x|| < \varepsilon, \quad \forall n > n_o .$$

Definition (Cauchy-Folge):
Eine Folge $\{x_n\} \subset V$ wird *Cauchy-Folge* genannt, wenn für jedes $\varepsilon > 0$ ein Index $n_0 = n_0(\varepsilon)$ existiert, mit

$$||x_m - x_n|| < \varepsilon, \forall m, n \leq n_o .$$

Definition (Vollständigkeit):
Ein Prä-Hilbert-Raum wird als *vollständig* bezeichnet, wenn jede Cauchy-Folge konvergiert, und der Grenzwert ein Element des Prä-Hilbert-Raums ist.

Definition (Hilbert-Raum):
Ein Prä-Hilbert-Raum heißt *Hilbert-Raum*, wenn der Vektorraum V bezüglich der induzierten Norm

$$\|x\| := \sqrt{\langle x, x \rangle}$$

vollständig ist.

Definition (lineare Abhängigkeit):
Die Vektoren $\{x_1,...,x_n\} \in V$ heißen *linear unabhängig*, wenn eine Linearkombination von $\{x_1,...,x_n\}$ nur dann Null sein kann, wenn alle „Koeffizienten" verschwinden, d.h. wenn gilt:

$$\lambda_1 x_1 + ... + \lambda_n x_n = 0 \Rightarrow \lambda_1 = ... = \lambda_n = 0 .$$

Definition (Dimension):
Die maximale Anzahl von linear unabhängigen Elementen eines Vektorraums V wird *Dimension* von V genannt.

„Da man in einem Hilbert-Raum über das Skalarprodukt „Winkel messen" kann, lässt sich auch der Begriff der Orthogonalität einführen."

Definition (Orthogonaliotät, Orthonormalität):
Eine Familie $(x_i)_{i \in I}$ von Elementen eines Hilbert-Raumes H heißt ein *Orthogonal-System*, wenn alle Elemente paarweise aufeinander senkrecht stehen, d.h. für alle $i \neq j \in I$ gilt

$$< x_i , x_j >= 0 .$$

Eine Familie $(x_i)_{i \in I}$ heißt *Orthonormal-System*, wenn zusätzlich alle Elemente auf die Länge 1 normiert sind, d.h. für alle $i,j \in I$ gilt

$$< x_i , x_j >= \delta_{ij} .$$

Definition (vollständiges Orthonormal-System):
Ein *Orthonormal-System* $(x_i)_{i \in I}$ heißt *vollständig*, wenn für jedes Element z eines Hilbert-Raums folgendes gilt

$$0 = \langle z, x_i \rangle \qquad \Leftrightarrow \qquad z = 0.$$

Definition (Fourier-Koeffizienten bzgl. eines Orthonormal-System):
In einem Hilbert-Raum H sind die *Fourier-Koeffizienten* eines Elementes $x \in H$ bezüglich eines Orthonormal-Systems $(x_i)_{i \in I}$ definiert als die Familie der Skalarprodukte

$$\left(\langle x, x_i \rangle\right)_{i \in I} .$$

Definition (Hilbert-Basis):
Ein Orthonormalsystem $(e_i)_{i \in I}$ von Elementen eines Hilbert-Raumes H heißt *Hilbert-Basis*, wenn jedes Element $x \in H$ die Fourier-Entwicklung hat

$$x = \sum_{i \in I} \langle x, e_i \rangle e_i .$$

Der Hilbert-Raum H heißt *separabel*, wenn er eine abzählbare Hilbert-Basis hat.

Bezeichnung (Räume intgrierbarer Funktionen)
Wir bezeichnen mit $L^2 = L^2(\mathbf{R})$ den Vektorraum der Klassen von messbaren Funktionen

$$f: \mathbf{R} \to \mathbf{C}$$

mit

$$\int_R |f(t)|^2 \, dt < \infty ,$$

versehen mit dem Skalarprodukt

$$< f, g >= \int_R f(t) g(t) dt .$$

Satz:
Der Vektorraum $L^2(\mathbf{R})$ ist vollständig, und damit ein Hilbert-Raum.

Kapitel 3

Die Fourier-Transformation

Die Fourier-Transformation wurde Anfang des 19. Jahrhunderts von Jean Baptiste Joseph Fourier entdeckt. Sie überführt eine zeit- (oder auch orts-)abhängige Funktion f in eine frequenzabhängige Funktion $\hat{f}$, die sogenannte Fourier-Transformierte. Die Fourier-Transformation wirkt wie eine Art mathematisches Prisma. Sie zerlegt die Funktion in die in ihr enthaltenen Frequenzen, wie in untenstehender Abbildung gezeigt wird.

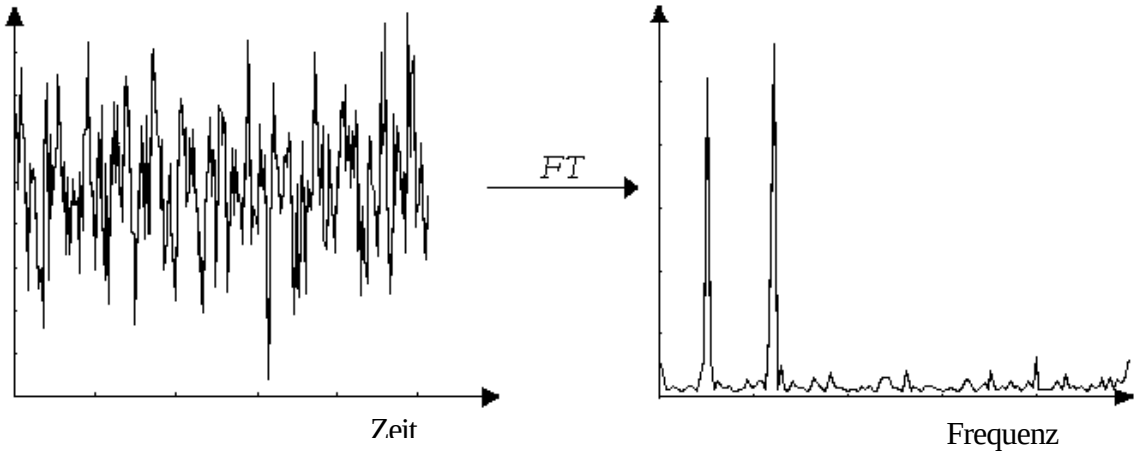


Abbildung 3.1: Signal und Fourier-Transformierte

3.1 Die Fourier-Reihe

Jede T-periodische Funktion lässt sich als Summe von Sinus- und Kosinusschwingungen darstellen

$$f(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\{a_n\cos(\frac{n\pi x}{T})+b_n\sin(\frac{n\pi x}{T})\} \tag{3.1.1}$$

mit den Fourier-Koeffizienten

$$a_n=\frac{1}{T}\int_0^{2T} f(x)\cos(\frac{n\pi x}{T})dx \tag{3.1.2}$$

$$b_n=\frac{1}{T}\int_0^{2T} f(x)\sin(\frac{n\pi x}{T})dx\,. \tag{3.1.3}$$

Im allgemeinen ist die Fourier-Reihe komplex[1] und wird durch folgende Formel definiert

$$f(x)=\sum_{n\in Z} c_n e^{i2\frac{\pi n x}{T}} \tag{3.1.4}$$

[1] Die komplexe Darstellung lässt sich mit Hilfe der Euler'schen Formel

$$e^{i\alpha}=\cos\alpha+i\sin\alpha$$

herleiten.

mit den Fourier-Koeffizienten

$$c_n = \frac{1}{2T}\int_{-T}^{T} f(x)e^{-i2\frac{\pi nx}{T}}dx. \tag{3.1.5}$$

3.2 Die Fourier-Transformation

Es können auch nichtperiodische Signale f(x) $\in$ $L^2(\mathbf{R})$ aus andauernden Schwingungen aufgebaut werden. Dazu denkt man sich die Periode ins Unendliche ausgedehnt. Dies führt zu einer Grenzwertaufgabe mit folgendem Ergebnis

$$f(x) = \int_R \hat{f}(\omega)e^{i\omega x}d\omega \tag{3.2.1}$$

$$\hat{f}(\omega) = \frac{1}{2\pi}\int_R f(x)e^{-i\omega x}dx \tag{3.2.2}$$

Ein wesentlicher Nachteil der Fourier-Transformation liegt darin, das sie Signale unter dem Aspekt der „Ewigkeit“ analysiert werden. Die Fourier-Transformation zerlegt Signale in Sinus- und Kosinusschwingungen, die unendlich lang mit derselben Periode schwingen. Daraus ergeben sich im wesentlichen zwei Nachteile. Die unendlich langen Schwingungen haben keinen lokalen Charakter – wann im Signal welche Frequenzen auftreten bleibt also unbekannt. Aus den Fourier-Koeffizienten lassen sich Schwingungen ableiten, die zusammengesetzt das Signal rekonstruieren. Tritt in einem der Koeffizienten ein Fehler auf, so wirkt sich der Fehler – aufgrund der unendlich langen Schwingung – auf das ganze Signal aus.

3.3 Die gefensterte Fourier-Transformation

Ein wesentlicher Nachteil der Fourier-Transformation ist das fehlen der zeitlichen Lokalisierung. Die Fourier-Transformierte zeigt zwar mit welchem Gewicht die einzelnen Frequenzen im Signal enthalten sind, doch wann die Frequenzen auftreten vermag sie nicht sichtbar zu machen. Das wäre gerade so, als wenn man die Noten eines Musikstücks der Tonhöhe nach sortiert. Man wüsste zwar welche Töne das Musikstück enthält, aber wann sie zu spielen sind wäre nicht bekannt. Eine Partitur gibt glücklicherweise an, wann welche Töne zu spielen sind. Zur Analyse von Signalen wäre eine Transformation, die ein Signal in eine partiturähnliche Darstellung überführt optimal. Die gefensterte Fourier-Transformation und die Wavelet-Transformation sind Transformtionen, die sowohl die Zeit- als auch die Frequenzinformation extrahieren.
Die gefensterte Fourier-Transformation bedient sich eines einfachen Tricks. Das Signal wird in zeitlich begrenzte Abschnitte unterteilt, welche dann nacheinander mittels Fourier-Transformation analysiert werden. Auf diese Weise kann man zumindest den Zeitraum eingrenzen innerhalb dessen sich etwas ereignet.
Ein Zeitfenster ist eine Funktion g(t), die nur in einer kleinen Umgebung von t=0 von Null verschieden ist. Normalerweise verwendet man gerade Funktionen. Die gefensterte Fourier-Transformation wird auch Gabor-Transformation genannt, wenn das Zeitfenster die Form der Gauß’schen Glockenkurve hat

$$g(t) = \pi^{-1/4}e^{-\frac{t^2}{2}}. \tag{3.3.1}$$

Die gefensterte Fourier-Transformation ist ein Funktion von zwei veränderlichen – der Frequenz ω und der Zeit t

$$G[f](\omega,t)=\frac{1}{\sqrt{2\pi}}\int_R f(u)g(u-t)e^{-iu\omega}du\,. \tag{3.3.2}$$

Die gefensterte Fourier-Transformation hat eine Inverse der Form

$$f(t)=\frac{1}{\sqrt{2\pi}}\int_R\int_R G[f](\omega,t)g(u-t)e^{i\omega t}d\omega dt\,. \tag{3.3.3}$$

Ein schwerwiegender Nachteil der gefensterte Fourier-Transformation ist, dass die Zeit- und die Frequenzauflösung nicht gleichzeitig beliebig klein gemacht werden können. Variiert man die Fensterbreite, so hat man entweder eine gute Frequenzauflösung oder eine gute Zeitauflösung. Je schmaler das Fenster ist, desto besser lassen sich plötzliche Änderungen wie scharfe Peaks erfassen. Gleichzeitig wird das Verfahren aber immer unempfindlicher gegenüber den niedrigen Signalfrequenzen, die nicht mehr in das schmale Fenster „passen“. Wählt man ein größeres Fenster, werden zwar die niedrigeren Frequenzen besser erfasst, dafür ist die zeitliche Auflösung aber schlechter.

Kapitel 4

Die Kontinuierliche Wavelet-Transformation

Ein wesentlicher Nachteil der gefensterten Fourier-Transformation liegt darin, dass bedingt durch die konstante Fensterbreite die zeitliche Auflösung für alle Frequenzen dieselbe ist. Man muss sich entscheiden zwischen schmaler Fensterbreite und guter zeitlicher Auflösung oder breiter Fensterbreite und guter Frequenzauflösung. Beides zusammen ließe sich nur erreichen, wenn das Signal mit unterschiedlichen Fensterbreiten mehrfach abgetastet wird. Damit steigt aber auch der Rechenaufwand erheblich.
Die Wavelet-Transformation löst dieses Dilemma sehr elegant. Anstatt wie bei der gefensterten Fourier-Transformation die Fensterbreite konstant zu halten wird die Fensterbreite variiert. Dagegen bleibt die Anzahl der Oszillationen im Fenster konstant. Wird die Fensterbreite verändert, so wird das Wavelet gedehnt oder gestaucht wie ein Akkordeon. Gleichzeitig werden auch die Oszillationen gedehnt oder gestaucht, so dass auch zwangsläufig deren Frequenz ab- oder zunimmt. Untenstehende Abbildung zeigt eine gedehnte und eine gestauchte Version des Wavelets auf der linken Seite.

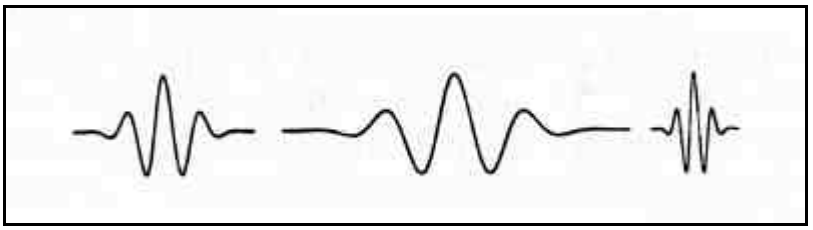


Abbildung 4.1: Dilatierte[2] Wavelets

Abbildung 4.2 soll die Wirkungsweise der Wavelet-Transformation erläutern und zeigen, wie die Wavelet-Koeffizienten die Korrelation zwischen Wavelet und Signalausschnitt charakterisieren. (a) stellt die zu analysierende Funktion dar, (b) das Wavelet. Dieses wird sukzessive mit verschiedenen Abschnitten der Funktion verglichen. Das Produkt des Abschnittes mit dem Wavelet ergibt eine neue Funktion, und die von dieser Funktion eingeschlossene Fläche ist der entsprechende Wavelet-Koeffizient. Funktionsabschnitte, die dem Wavelet „ähneln“ führen zu großen Wavelet-Koeffizienten was in (c) bzw. (d) zu sehen ist. Geringe Ähnlichkeit führt dagegen zu kleinen Wavelet-Koeffizienten ((e) bzw. (f)). Unter Verwendung von gedehnten Versionen des Wavelets (b) können auch die niederfrequenteren Anteile in (e) erfasst werden.
Eine wesentliche Eigenschaft der Wavelets ist, dass sie nur Veränderungen im Signal wahrnehmen. Ist ein Signal in einem Abschnitt konstant, so ist aufgrund von (4.5) sein Wavelet-Koeffizient gleich Null. Diese Eigenschaft macht man sich beispielsweise bei der Bilddatenkomprimierung zunutze. Bilder sind oft sehr homogene Signale mit wenig Signalsschwankungen. Die daraus resultierenden Wavelet-Koeffizienten sind sehr klein und können vernachlässigt werden.
Ein Nachteil der Fourier-Transformation ist, dass sie sehr fehleranfällig ist. Ein Fehler in einem Fourier-Koeffizienten wirkt sich bei der Rekonstruktion auf das gesamte Signal aus. Dies liegt daran, dass die analysierenden Funktionen unendlich ausgedehnte Sinus- und Kosinusschwingungen sind. Bei der Wavelet-Transformation wird das zu analysierende

[2] In der Literatur werden das Dehnen und das Stauchen unter dem Begriff Dilatation (<lat.> Aufschub) zusammengefasst.

Signal mit zeitlich lokalisierten „kurzen Wellen“ gescannt, wodurch auch bei der Rekonstruktion nur lokal begrenzte Fehler auftreten.

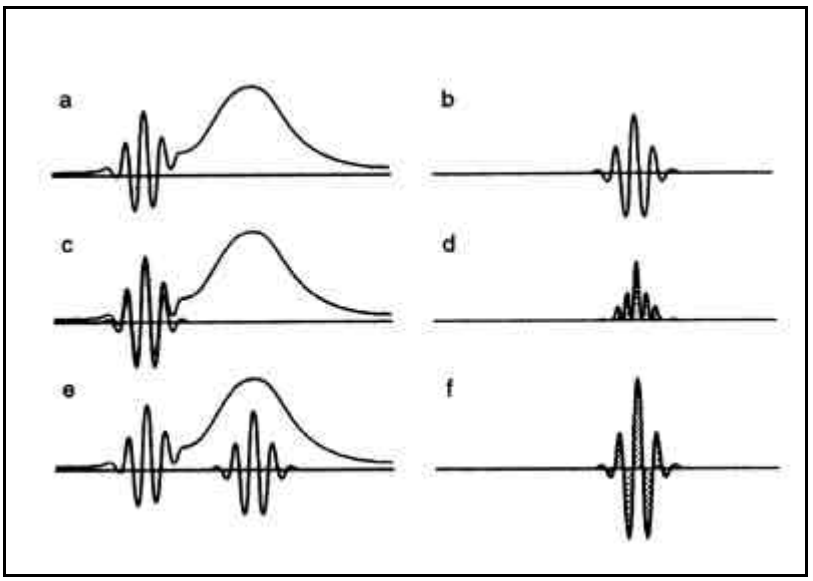


Abbildung 4.2: Korrelation zwischen Signal und Wavelet

Definition (Wavelet, Wavelet-Transformierte):
Eine Funktion $\psi \in L^2(\mathbf{R})$, welche die Zulässigkeitsbedingung

$$0 < c_\psi := 2\pi \int_R \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty \quad , \tag{4.1}$$

erfüllt, heißt *Wavelet*. Die *Wavelet-Transformierte* einer Funktion $f \in L^2(\mathbf{R})$ zum Wavelet ψ ist durch

$$W_\psi[f](a,b) = \frac{1}{\sqrt{c_\psi}} |a|^{-1/2} \int_R f(t) \psi\left(\frac{t-b}{a}\right) dt \tag{4.2}$$

$$= \frac{1}{\sqrt{c_\psi}} \int f(t) \psi_{a,b}(t) dt \tag{4.3}$$

mit

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \tag{4.4}$$

$a \in \mathbf{R}\backslash\{0\}$, $b \in \mathbf{R}$, definiert.

Aus der Definition eines Wavelets können bereits die wichtigsten Eigenschaften der Wavelets abgeleitet werden. Die Zulässigkeitsbedingung impliziert

$$0 = \hat{\psi}(0) = (2\pi)^{-1/2} \int_R \psi(t) dt \,, \tag{4.5}$$

d.h. der Mittelwert des Wavelets ist gleich Null. Wenn das Wavelet einen verschwindenden Mittelwert hat, dann muss es zwischen positiven und negativen Werten oszillieren. Aus der Bedingung $\psi \in L^2(\mathbf{R})$ folgt, dass ψ quadratisch integrierbar ist. Daher muss ψ für $t \to \infty$

gegen Null gehen. Diese Eigenschaften werden bereits durch den Namen „Wavelets“ suggeriert. Der Name ist von „little Waves“ abgeleitet

Satz (Wavelet-Transformation als Isometrie):
Für ein Wavelet $\psi \in L^2(\mathbf{R})$ ist die Wavelet-Transformation W_ψ eine *Isometrie*.

Satz (Umkehrformel der Wavelet-Transformation):
Für die Wavelet-Transformation W_ψ mit einem Wavelet $\psi \in L^2(\mathbf{R})$ gilt die *Umkehrformel*

$$f(t) = \frac{1}{\sqrt{c_\psi}} \int_R \int_R W_\psi[f](a,b) |a|^{-1/2} \psi\left(\frac{t-b}{a}\right) \frac{da db}{a^2} . \tag{4.6}$$

4.1 Zeit-Frequenz-Auflösung

Im Folgenden soll erläutert werden, in wie weit einem Zeitpunkt t eine Frequenz zugeordnet werden kann. Es stellt sich die Frage, wie man überhaupt über Frequenzen in einem gewissen Zeitpunkt sprechen kann, wenn diese stets eine gewisse Zeit zum Oszillieren benötigen? Sicherlich ist eine eindeutige Zeit-Frequenz-Zerlegung eines Signals nicht möglich – man muss demnach mit einer gewissen Unschärfe der Zuordnung rechnen. Dieser Sachverhalt kann anhand von Wavelets gut klar gemacht werden. Niedrige Frequenzen werden von breiten Wavelets analysiert. Diese breiten Wavelets vermögen zwar die Frequenz gut aufzulösen, sind aber aufgrund ihrer breite zeitlich nur ungenau zu bestimmen. Bei hohen Frequenzen stellt sich das umgekehrte Problem. Schmale Wavelets sind zwar bezüglich der Zeitauflösung sehr gut, nicht aber bezüglich der Frequenz. Je schmaler die Wavelets werden, desto stärker entziehen sie sich den im Signalabschnitt enthaltenen Frequenzen.
Die Zeit- und Frequenzauflösung können nicht gleichzeitig verbessert werden, was durch die Unschärferelation ausgedrückt wird

$$\sigma_{\psi_{a,b}} \cdot \sigma_{\hat{\psi}_{a,b}} \geq \frac{1}{4} \tag{4.1.1}$$

mit $\sigma_{\psi_{a,b}}$ ... Maß für die Genauigkeit der Zeitauflösung

$\sigma_{\hat{\psi}_{a,b}}$ ... Maß für die Genauigkeit der Frequenzauflösung.

Die Zeit- und die Frequenzauflösung hängt vom Skalenparameter a ab. Mit zunehmender Frequenz, d.h. für |a| abnehmend, wird die Auflösung im Zeitbereich besser. Dies wird in Abbildung 4.3 anhand von zwei Skalenparametern a_1 und a_2 demonstriert ($a_1 > a_2$). Die durch die Rechtecke dargestellt Fläche entspricht $\sigma_{\psi_{a,b}} \cdot \sigma_{\hat{\psi}_{a,b}}$.

Die Unschärferelation konnte auch schon bei der gefensterten Fourier-Transformation beobachtet werden. Allerdings war hier die Zeit- als auch die Frequenzauflösung für jeden Zeitpunkt und für jede Frequenz gleich. Die Wavelet-Transformation gestaltet sich diesbezüglich wesentlich flexibler. Einen Extremfall stellt im übrigen die Fourier-Transformation dar. Hier hat man die Qual der Wahl. Entscheidet man sich für die Zeitdarstellung f(t), so hat man exakte Zeitangaben, aber keinerlei Information über das Frequenzverhalten. Dagegen sind bei der Fourier-Transformierten $\hat{f}(\omega)$ die Frequenzen exakt definiert auf Kosten jeglicher Zeitinformation.

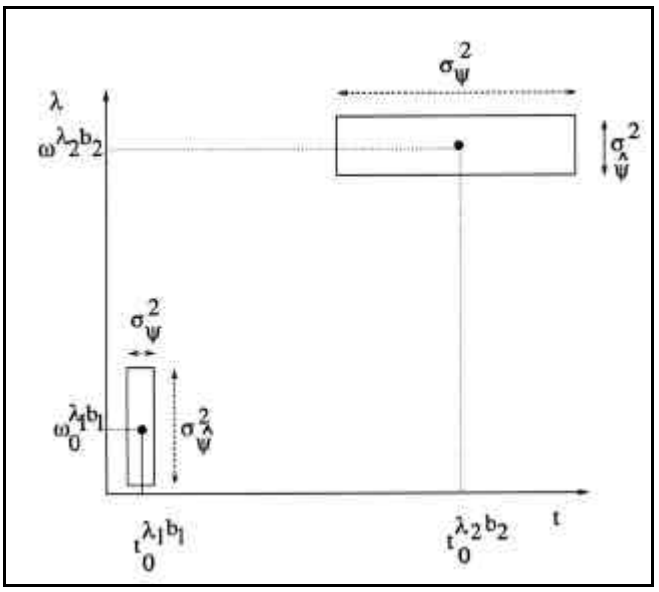


Abbildung 4.3: Zeit-Frequenz-Auflösung

4.2 Redundanz der kontinuierlichen Wavelet-Transformation

Um ein Signal rekonstruieren zu können muss das Signal bei allen möglichen Auflösungen a und Positionen b des Wavelets analysiert werden. Da die Position $b \in \mathbf{R}$ ist, sind unendlich viele Wavelet-Koeffizienten zu bestimmen, bei deren Berechnung das Wavelet kontinuierlich über das Signal hinweggleitet. Nach dieser, im Prinzip unendlich lange währenden Berechnung wird das Wavelet infinitesimal gestaucht, und die Berechnungen beginnen erneut. Diese unendlich vielen Berechnungen werden in der Praxis durch eine große, aber diskrete Anzahl von Koeffizienten ersetzt. Im Allgemeinen ist eine exakte Rekonstruktion des Signals in diesem Fall jedoch nicht mehr gewährleistet. Um jedoch eine Aussage über das Frequenzverhalten des Signals zu machen ist eine diskrete Anzahl von Koeffizienten in der Regel ausreichend.

Theoretisch ist die kontinuierliche Wavelet-Transformation unendlich redundant und somit eine Vergeudung von Ressourcen. Die Wavelets überlappen sich, und demzufolge ist die meiste in einem herausgegriffenen Koeffizienten enthaltene Information auch in dessen Nachbarkoeffizienten enthalten. Um die Wavelet-Transformation auch in der Praxis einsetzen zu können gilt es die Redundanz zu vermeiden. Erst dann ist eine effektive Analyse und Synthese des Signals, aber auch die Komprimierung des Signals möglich. Die Orthogonaltransformationen kodieren die Information nur einfach und sind somit redundantfrei. Es stellt sich allerdings die Frage, ob es Wavelets gibt, die die Anforderungen einer Orthogonaltransformation erfüllen.

Kapitel 5

Die diskrete Wavelet-Transformation

Für praktische Anwendungen ist die kontinuierliche Wavelet-Transformation nicht effizient genug, da für Analyse und Synthese ein Einfach- bzw. ein Doppelintegral gelöst werden müssen. Dies führt zu langwierigen Berechnungen. Der Durchbruch der Wavelets kam erst mit der diskreten Variante der Wavelet-Transformation.

2.1 Wavelet-Frames

Wie im vorigen Kapitel erwähnt wurde, ist die kontinuierliche Wavelet-Transformation unendlich redundant. Es drängt sich folglich die Frage auf, ob die Wavelet-Transformation $W_\psi[f](a,b)$ wirklich an jedem Punkt $(a,b) \in \mathbf{R}^* \times \mathbf{R}$ bekannt sein muss, um f exakt rekonstruieren zu können. Oder anders gefragt: Reicht es nicht aus, das Signal - ohne Informationsverlust - nur an diskreten Stellen abzutasten und dabei nur diskrete Skalen zu verwenden? Die Antwort lautet ja. Doch für welche Wavelets ψ und für welche diskreten Teilmengen $(a_m,b_n) \subset (a,b)$ enthält die Wavelet-Transformierte die vollständige Information eines beliebigen Signals?
Zunächst sollen die Skalierungs- und Translationsparameter definiert werden. Während bei der kontinuierliche Wavelet-Transformation die Wavelet-Koeffizienten $W_\psi[f](a,b)$ eine überabzählbare Menge ergeben, beschränkt sich die diskrete Wavelet-Transformation auf eine abzählbare Menge von Koeffizienten $W_\psi[f](a_m,b_n)$ $m,n \in \mathbf{Z}^2$. Dies liegt daran, dass Skalierungs- und Translationsparameter nur noch diskrete Werte annehmen.

Definition (Skalierungs- und Translationsparameter):
Es werden der feste Zoom-Faktor $a_0 > 1$ und die feste Translationsdistanz $b_0 > 0$ gewählt. Die Folge von *Skalierungsparametern* ergibt sich aus den Potenzen von a_0

$$a_m := a_0^{\,m},\ m \in \mathbf{Z}. \tag{5.1.1}$$

Die Folge von *Tranlationen* ergibt sich zu jedem festen Skalierungsparameter a_m in folgender Weise

$$b_{m,n} := n \cdot b_0 \cdot a_m,\ m, n \in \mathbf{Z}. \tag{5.1.2}$$

Durch die Abhängigkeit von a_m wird sichergestellt, dass die Translationsschritte zu verschiedenen Skalen ineinander enthalten sind. In Abbildung 5.1 ist die Verteilung der Punkte für $a_0 = 2$ und $b_0 = 1$ dargestellt. Die Punkte sind nicht gleichmäßig verteilt. Sie liegen bei kleinen Skalen näher zusammen, als bei großen Skalen. Dies ist offensichtlich logisch, wenn man sich überlegt, dass kleine Skalen auch „schmale“ Wavelets bedeuten.
Für ein Wavelet ψ ergibt sich analog zur Schreibweise (4.4) folgende „Wavelet-Familie“:

$$\psi_{m,n}^{(a_0,b_0)}(t) = \frac{1}{\sqrt{a_0^{\,m}}}\psi\left(\frac{t - n \cdot b_0 \cdot a_0^{\,m}}{a_0^{\,m}}\right),(m, n) \in \mathbf{Z}^2. \tag{5.1.3}$$

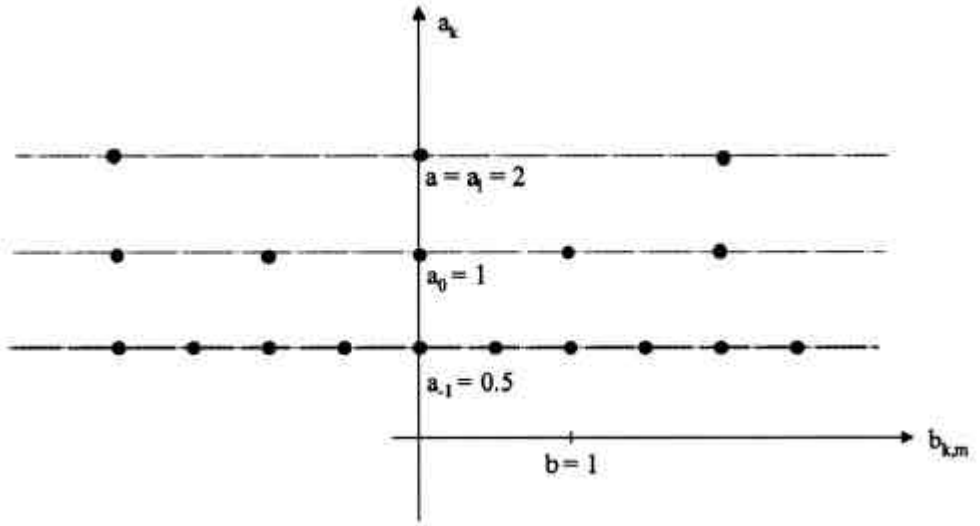


Abbildung 5.1: Verteilung der Skalenraumpunkte

Im Folgenden soll nun erklärt werden, unter welchen Bedingungen an das Wavelet ψ und an die diskrete Menge der Dilatationen und Translationen eine exakte Rekonstruktion von f aus $W_\psi[f](a_m, b_n)$ gewährleistet ist.

Definition (Wavelet-Frame):
Seien $a_0 > 1$, $b_0 > 0$ und $\psi \in L^2(\mathbf{R})$. Das Funktionensystem $\{\gamma_{m,n}^{(a_0,b_0)} \mid m,n \in Z\}$ bildet einen *Wavelet-Frame* für $L^2(\mathbf{R})$, falls es Konstanten A, B > 0 gibt, so dass

$$A\|f\|^2 \le \sum_{m\in Z}\sum_{n\in Z}\left|\left\langle \gamma_{m,n}^{(a_0,b_0)}, f\right\rangle\right|^2 \le B\|f\|^2 \tag{5.1.4}$$

gilt. Man sagt, das Tripel (ψ, a_0, b_0) erzeugt den Frame. Die Konstanten A und B werden als Schranken des Frames bezeichnet. Der Frame heißt *straff*, falls A = B ist.

Satz (Frame-Operator):
Jedem Frame (ψ, a_0, b_0) kann der Operator $T : L^2(\mathbf{R}) \to l^2(\mathbf{Z^2})$, $(Tf)_{m,n} := \left\langle \gamma_{m,n}^{(a_0,b_0)}, f\right\rangle$, zugeordnet werden, der

$$\sqrt{A}\|f\| \le \|Tf\| \le \sqrt{B}\|f\| \tag{5.1.5}$$

erfüllt. Der Operator ist somit stetig, $\|T\| \le B^{1/2}$, und auf seinem Bild stetig invertierbar, $\left\|T^{-1}|_{range(T)}\right\| \le A^{-1/2}$, d.h. f kann aus den diskreten Werten $(Tf)_{m,n} = W_\gamma[f](a_0^m, nb_0a_0^m)$ zurückgewonnen werden. Der Frame-Operator ist also begrenzt und besitzt einen begrenzten inversen Frame-Operator T^{-1}.
Diese Eigenschaft des Frames ist sehr wichtig, da sie eine eineindeutige Beziehung zwischen dem Signal f und seiner Wavelet-Transformierten garantiert. Zusammenfassend lässt sich sagen, dass das Signal f vollständig durch seine Wavelet-Koeffizienten $W_\psi[f](a_m,b_n)$ beschrieben werden kann, wenn (ψ, a_0, b_0) einen Wavelet-Frame bilden. Doch wie lässt sich das Signal aus den Koeffizienten wieder zurückgewinnen? Bilden (ψ, a_0, b_0) einen straffen Frame, so kann das Signal sehr einfach rekonstruiert werden.

Lemma:
Ein straffer Frame (ψ, a_0, b_0) mit den Schranken A = B = 1 erzeugt eine *vollständige Orthonormalbasis* des $L^2(\mathbf{R})$, falls ψ normiert ist.

Dies ist sehr bedeutend für die Rekonstruktion von f. Bildet das Funktionensystem $\{\gamma_{m,n}^{(a_0,b_0)} \mid m,n \in Z\}$ nämlich eine Orthonormalbasis, dann kann das Signal f durch eine Fourier-Entwicklung rekonstruiert werden

$$f(t) = \sum_{m \in Z} \sum_{n \in Z} < \gamma_{m,n}^{(a_0,b_0)}, f > \gamma_{m,n}^{(a_0,b_0)}(t). \tag{5.1.6}$$

Man bezeichnet (ψ, a_0, b_0) auch als orthogonale Wavelet-Basis. Die daraus abgeleiteten Wavelets werden auch als orthogonale Wavelets bezeichnet, was allerdings mathematisch nicht ganz korrekt ist, da ein einzelnes Wavelet nicht orthogonal sein kann. Sie können lediglich paarweise zueinander orthogonal sein.

5.2 Multi-Skalen-Analyse

Das von Stéphane Mallat und Yves Meyer entwickelte Konzept der Multi-Skalen-Analyse (MSA) schlägt eine Brücke von den orthogonalen Wavelets zu den in der Signalverarbeitung verwendeten Filtern. Ähnlich wie durch die Verwendung von verschiedenen Bandpassfiltern wird ein Signal durch eine Multi-Skalen-Analyse in verschiedene Frequenzbänder zerlegt.
Die MSA ist von großer Bedeutung, da sie eine schnelle und stabile Wavelet-Analyse und –Synthese erlaubt. Zudem entwickelte sich aus der Theorie der MSA ein Konzept zur Erzeugung von orthogonalen Wavelet-Basen.

Bemerkung:
Im Folgenden werden nur Wavelet-Frames der Form (ψ,2,1) betrachtet. O.B.d.A seien $a_0 = 2$ und $b_0 = 1$. Für das Wavelets ψ ergeben sich die dilatierten und translatierten Versionen $\psi_{m,k}$ nach Formel (5.1.3) zu

$$\gamma_{m,k}(t) = \frac{1}{\sqrt{2^m}} \gamma(2^{-m} t - k). \tag{5.2.1}$$

Das folgende Beispiel soll die Wirkungsweise einer MSA verdeutlichen. Es zeigt, wie ein Signal in Unterräume und Detailräume zerlegt wird.
Das Signal f aus einem Unterraum V_{-1} des $L^2(\mathbf{R})$ soll in seinen hoch- und niederfrequenten Anteil aufgespaltet werden. Der glatte (niederfrequente) Anteil wird durch die orthogonale Projektion $P_0 f$ auf einem kleineren Unterraum V_0 beschreiben. Dieser enthält die niederfrequenten Funktionen von V_{-1}. Das orthogonale Komplement von V_0 in V_{-1} ist der Raum W_0. Er beinhaltet die hochfrequenten Anteile des Signals f, und wird deswegen auch Detailraum genannt. Die Projektion von f auf W_0 sei $Q_0 f$, dann ist

$$f = P_0 f + Q_0 f,$$

$$V_{-1} = V_0 \oplus W_0, \qquad V_0 \perp W_0.$$

In gleicher Weise kann nun $P_0 f$ in „glatte“ und „raue“ Elemente aufgeteilt werden. Dies entspricht einer Zerlegung des Unterraums V_0 in den Unterraum V_1 und den Detailraum W_1. Die dazugehörigen Projektoren heißen P_1 und Q_1, so dass folgt

$$P_0 f = P_1 f + Q_1 f,$$

bzw.

$$f = P_1 f + Q_1 f + Q_0 f. \tag{5.2.2}$$

Mit jeder Stufe der MSA wird das Signal in zwei Komponenten zerlegt, die eine enthält die Hauptinformation, die andere die Detailinformation. Mit der jeweils nächsten Stufe wird nur die Hauptinformation weiter aufgelöst. Die zweistufige Zerlegung aus (5.2.2) wird durch Abbildung 5.2 demonstriert. Es ist zu beachten, dass die angedeutete Addition nicht die Addition von Funktionen ist. Das Pluszeichen bedeutet hier, dass die Funktionen auf der linken Seite durch die beiden Funktionen der rechten Seite rekonstruiert werden können.

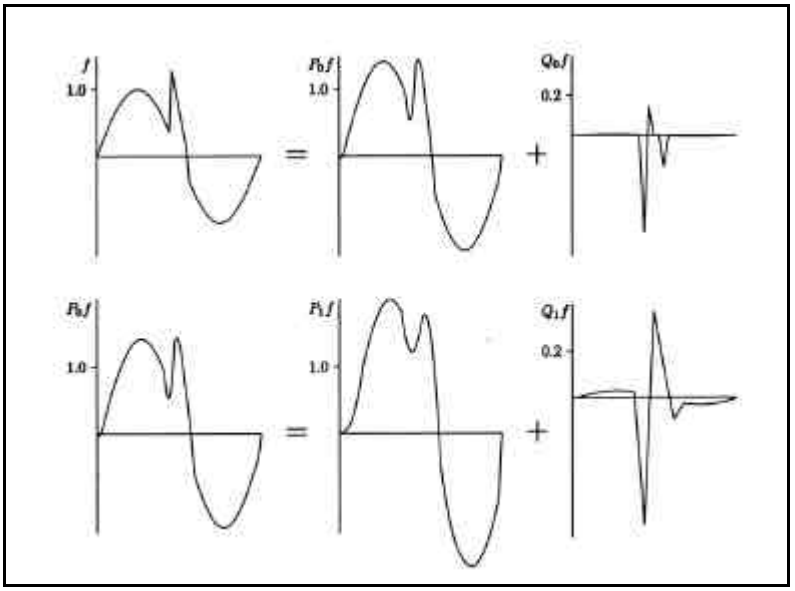


Abbildung 5.2: Zweistufige Zerlegung des Signals f

Rekursiv kann das Signal weiter zerlegt werden, was durch Abbildung 5.3 deutlich gemacht werden soll.

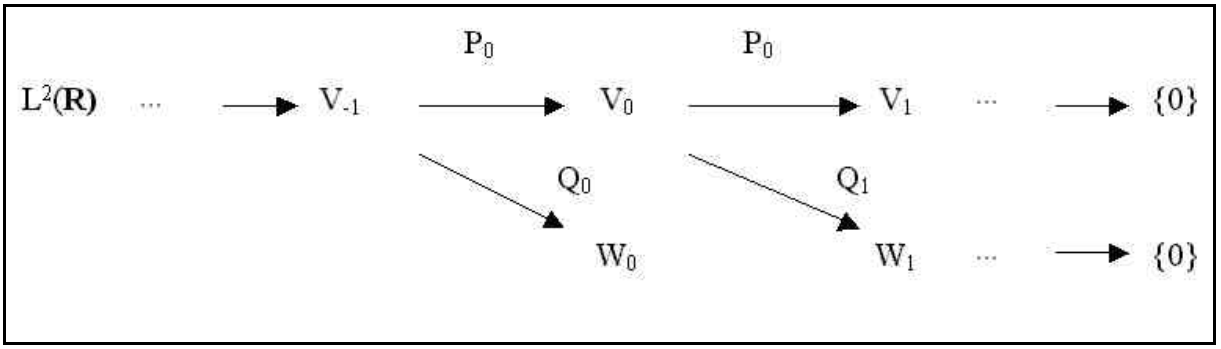


Abbildung 5.3: Hierarchie der Unterräume V_m und der Detailräume W_m

Mathematisch gesehen stellt sich der gesamte Prozess als eine sukzessive Projektion des Signals f auf abgeschlossene Unterräume des Hilbert-Raumes $L^2(\mathbf{R})$ dar.

Definition (Multi-Skalen-Analyse):
Eine *Multi-Skalen-Analyse* des $L^2(\mathbf{R})$ ist eine aufsteigende Folge abgeschlossener Unterräume $V_m \subset L^2(\mathbf{R})$

$$\{0\} \subset \ldots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \ldots \subset L^2(\mathbf{R}),$$

so dass gilt:

$$\overline{\bigcup_{m\in Z} V_m} = L^2(\mathbf{R}), \tag{5.2.3}$$

$$\bigcap_{m\in Z} V_m = \{0\}, \tag{5.2.4}$$

$$f(\cdot)\in V_m \Leftrightarrow f(2^m\cdot)\in V_0 . \tag{5.2.5}$$

Es gibt eine Funktion $\varphi \in L^2(\mathbf{R})$ mit

$$V_0 = \overline{span\{\boldsymbol{j}(\cdot - k)\,|\, k \in Z\}}$$

und

$$A\sum_{k\in Z} c_k^2 \le \left\| \sum_{k\in Z} c_k \boldsymbol{j}(\cdot - k) \right\|^2 \le B\sum_{k\in Z} c_k^2 \tag{5.2.6}$$

für alle $\{c_k\}_{k\in \mathbf{Z}} \in l^2(\mathbf{Z})$. A und B sind positive Konstanten.

Bemerkungen:

(a) Die Forderungen (5.2.3) und (5.2.4) werden von vielen Familien $\{V_m\}_{m\in \mathbf{Z}}$ erfüllt. Gleichung (5.2.3) besagt, dass jede Funktion $f \in L^2(\mathbf{R})$ mit beliebiger Genauigkeit approximiert werden kann. Bedingung (5.2.4) bedeutet, dass das einzige Objekt, das allen Räumen V_m gemein ist, die Funktion 0 ist.

(b) Die Eigenschaft (5.2.5) ist charakteristisch für eine MSA: Die Räume V_m sind skalierte Versionen des Grundraumes V_0, der durch verschobene Funktionen φ aufgespannt wird. φ stellt die sogenannte *Skalierungsfunktion* dar (5.2.6).
Für $m \to \infty$ werden die Funktionen V_m immer breiter und der Informationsgehalt nimmt immer mehr ab. Für $m \to -\infty$ enthalten die Räume V_m dagegen immer feinere Strukturen.

(c) Aus den Bedingungen (5.2.5) und (5.2.6) lässt sich ableiten, dass der Raum V_m von den Funktionen

$$\boldsymbol{j}_{m,k}(t) = \frac{1}{\sqrt{2^m}}\boldsymbol{j}(2^{-m}t - k) \tag{5.2.7}$$

aufgespannt wird:

$$V_m = \overline{span\{\boldsymbol{j}_{m,k}\,|k \in Z\}} . \tag{5.2.8}$$

Die Detailräume W_m werden als orthogonale Komplemente von V_m in V_{m-1} definiert,

$$V_{m-1} = W_m \oplus V_m, \qquad V_m \perp W_m \tag{5.2.9}$$

und die Operatoren Q_m als orthogonale Projektoren von $L^2(\mathbf{R})$ in W_m,

$$P_{m-1} = Q_m + P_m.$$

Offensichtlich ist

$$V_m = \bigoplus_{j \geq m+1} W_j \tag{5.2.10}$$

und damit

$$L^2(\mathbf{R}) = \bigoplus_{j \in Z} W_j \tag{5.2.11}$$

Die Räume W_m erben die Skalierungseigenschaft der Räume V_m (5.2.5)

$$f \in W_m \quad \Leftrightarrow \quad f(2^m \cdot) \in W_0 .$$

Eine Funktion $f \in L^2(\mathbf{R})$ lässt sich zerlegen durch

$$f \overset{(5.2.11)}{=} \sum_{j \in Z} Q_j f = \sum_{j \geq m+1} Q_j f + \sum_{j \leq m} Q_j f$$

$$\overset{(5.2.10)}{=} P_m f + \sum_{j=-\infty}^{m} Q_j f . \tag{5.2.12}$$

Dies ist die Verallgemeinerung der Gleichung (5.2.2). Üblicherweise sagt man, dass $P_m f$ die Details des Signals f bis zur Größe 2^m enthält. Das entspricht der Anwendung eines Tiefpassfilters auf das Signal, der mit wachsendem m einen kleineren Durchlassbereich hat. Der verbleibende Rest aus dem Hochfrequenzbereich wird in seine Anteile zu verschiedenen Frequenzbändern $Q_j f$, $-\infty$ j m, aufgeteilt. Dabei enthält $Q_j f$ nur die Details, die $P_{j-1} f$ von $P_j f$ unterscheiden, $Q_j f = P_j f - P_{j-1} f$.

Eingangs wurde erwähnt, dass mit Hilfe der MSA orthogonale Wavelet-Basen konstruiert werden können. Hierfür ist die im Folgenden Lemma festgehaltene Eigenschaften der Skalierungsfunktion wichtig. Darin liegt der Schlüssel zur Konstruktion sowohl orthogonaler Wavelet-Basen als auch schneller Algorithmen.

Lemma (Skalierungsgleichung):
Die Skalierungsfunktion φ erfüllt eine *Skalierungsgleichung*, d.h. es gibt eine Folge $\{h_k\}_{k \in \mathbf{Z}}$ reeller Zahlen mit

$$\varphi(x) = \sqrt{2} \sum_{k \in Z} h_k \varphi(2x - k) . \tag{5.2.13}$$

Ein Beweis lässt sich [LoMaRi] S.113 finden.
Aus der Forderung (5.2.6) der MSA folgt, dass die Skalierungsfunktion φ eine Orthogonalbasis des Raumes V_0 ist. Außerdem ist bekannt, dass der Detailraum W_0 das orthogonale Komplement von V_0 in V_{-1} ist. Demnach muss auch jede Basisfunktion ψ von W_0 orthogonal zu jedem Element φ des Raumes V_0 sein. Zur Erzeugung der Basisfunktion ψ

macht man sich die Orthogonalitätsbedingung zunutze. Die Funktion ψ lässt sich wie folgt darstellen:

$$\psi(x)=\sqrt{2}\sum_{m\in Z} g_m \varphi(2x-m)\,. \tag{5.2.14}$$

Mit Hilfe der Orthogonalitätsbedingung

$$0 = <\psi,\varphi>.$$

lässt sich eine Lösung für die Folge $\{g_m\}_{m\in \mathbf{Z}}$ finden

$$g_m = (-1)^m \cdot h_{1-m}.$$

Die translatierten Versionen von ψ bilden eine Orthogonalbasis des Detailraums W_0. Außerdem ist ψ ein Wavelet. Die Erzeugung eines Wavelets ψ aus einer Skalierungsfunktion φ wird im folgenden Satz zusammengefasst.

Satz:
Sei $\{V_m\}_{m\in \mathbf{Z}}$ eine MSA, die von der orthogonalen Skalierungsfunktion $\varphi \in V_0$ erzeugt wird. Die Funktion $\psi \in V_{-1}$, definiert durch

$$\psi(x)=\sqrt{2}\sum_{k\in Z} g_k \varphi(2x-k)=\sum_{k\in Z} g_k \varphi_{-1,k}(x), \tag{5.2.15}$$

$$g_k=(-1)^k h_{1-k}\,, \tag{5.2.16}$$

wobei $\{h_k\}_{k\in \mathbf{Z}}$ die Koeffizienten der Skalierungsgleichung (5.2.13) sind, besitzt die folgenden Eigenschaften

(i) $\{\psi_{m,k}(\cdot)=2^{-m/2}\psi(2^{-m}\cdot-k)\,|\,k\in Z\}$ ist eine Orthonormalbasis für W_m,

(ii) $\{\psi_{m,k}\,|\,m,k\in Z\}$ ist eine Orthonormalbasis für $L^2(\mathbf{R})$,

(iii) ψ ist ein Wavelet mit $c_\psi = 2\pi\int |\omega|^{-1}|\hat{\psi}(\omega)|^2\,d\omega = 2\ln 2$.

Ein Beweis lässt sich [LoMaRi] S.122/123 finden.
Der obige Satz zeigt, wie aus einer Skalierungsfunktion φ, die die Bedingungen einer MSA erfüllt ein straffer Wavelet-Frame konstruiert werden kann. Ist also die Skalierungsfunktion φ bekannt, so kann – nach Berechnung der Skalierungskoeffizienten $\{h_k\}_{k\in \mathbf{Z}}$ - das sogenannte „Mutter-Wavelet“ konstruiert werden. Die Skalierungsfunktion wird auch als „Vater-Wavelet“ bezeichnet. Die dilatierten und translatierten Versionen des Mutter-Wavelets werden konsequenterweise auch als „Baby-Wavelets“ bezeichnet.
Aus der Eigenschaft (i) es obigen Satzes wird zudem klar, dass die Detailräume W_m von den Wavelets $\psi_{m,k}$,$\in \mathbf{Z}$ aufgespannt werden. Wavelets kodieren die Differenz zwischen den in zwei aufeinanderfolgenden Räumen V_{m-1} und V_m enthaltenen Informationen. Oder anders ausgedrückt sie enthalten die Details, die man zu einem Raum V_m hinzunehmen muss, um den Raum V_{m-1} mit der doppelten Auflösung zu erhalten.
Die Konstruktion der Skalierungsfunktion gestaltet sich dagegen deutlich schwerer. Lange Zeit kannte man nur die Haar’sche Skalierungsfunktion und es war ungewiss, ob es noch

weitere Funktionen gibt, die die Bedingungen (5.2.3) bis (5.2.6) erfüllen. Es ist ein Verdienst der Belgierin Ingrid Daubechies, die Mitte der 1980er Jahre Algorithmen zur Konstruktion orthogonaler Wavelet-Basen entdeckte. Auf die umfangreiche und komplexe Theorie der nach ihr benannten Daubechies-Wavelets wird hier nicht näher eingegangen. Es soll jedoch kurz erwähnt werden, dass die Daubechies-Wavelets sich nicht auf analytische Weise herleiten lassen, sondern sich aus dem Verlauf eines Iterationsprozesses ergeben. Einige Abbildungen dieser Wavelets sind in Kapitel 5.4 zu sehen.

5.3 Die schnelle Wavelet-Transformation

Die schnelle Wavelet-Transformation ist ein einfaches und rasches Verfahren zur Berechnung der diskreten Wavelet-Transformation. Die Algorithmen lassen sich direkt aus der Multi-Skalen-Analyse und den Skalierungsgleichungen herleiten. Wie bisher ausgeführt lässt sich die Wavelet-Transformierte als Skalarprodukt des Signals f und des Wavelets $\psi_{m,k}$ berechnen

$$W_y[f](2^m, k2^m) = \langle f, \psi_{m,k} \rangle \quad m,k \in \mathbf{Z}$$

mit

$$\psi_{m,k} = \frac{1}{\sqrt{2^m}} \psi(2^{-m} \cdot -k)$$

Die numerische Berechnung der Skalarprodukte ist sehr aufwendig. Mit Hilfe der von Stéphane Mallat entwickelten Algorithmen der schnellen Wavelet-Transformation lassen sich die Koeffizienten der diskreten Wavelet-Transformation aus den Koeffizienten der nächstgrößeren Auflösung bestimmen.
Es werden die folgenden Bezeichnungen eingeführt

$$d_k^m = \langle f, \psi_{m,k} \rangle \tag{5.3.1}$$

$$c_k^m = \langle f, \varphi_{m,k} \rangle . \tag{5.3.2}$$

V_0 sei der Grundraum einer Multi-Skalen-Analyse zu einer orthogonalen Skalierungsfunktion φ. Die Funktion $f \in V_0$ besitzt aufgrund der Definition der MSA eine Entwicklung

$$f(x) = \sum_{k \in Z} c_k^0 \varphi(x-k) \tag{5.3.3}$$

mit den Entwicklungskoeffizienten

$$c^0 = \{c_k^0 \mid k \in Z\} .$$

Mit Hilfe der Skalierungsgleichungen (5.2.13) und(5.2.15) lassen sich die Koeffizienten d_k^m und c_k^m wie folgt berechnen

$$d_k^m = \langle f, \psi_{m,k} \rangle \overset{(5.2.15)}{=} \sum_{l \in Z} g_l \langle f, \varphi_{m-1,2k+l} \rangle = \sum_{l \in Z} g_{l-2k} c_l^{m-1} \tag{5.3.4}$$

$$c_k^m = \langle f, \varphi_{m,k} \rangle \overset{(5.2.13)}{=} \sum_{l \in Z} h_l \langle f, \varphi_{m-1,2k+l} \rangle = \sum_{l \in Z} h_{l-2k} c_l^{m-1} \tag{5.3.5}$$

Die Gleichungen (5.3.4)und (5.3.5) bilden den sogenannten Mallat-Algorithmus. Ausgehend von der Folge c^0 lassen sich mit Gleichung (5.3.3) die Koeffizienten c_k^1 der orthogonalen

Projektion $P_1 f$ von f in dem Raum V_1 berechnen. Die Projektion $P_1 f$ lässt sich demnach wie folgt darstellen

$$P_1 f = \sum_{k \in Z} c_k^1 \boldsymbol{j}_{1,k} \ . \tag{5.3.6}$$

Die mit (5.3.4) berechneten Koeffizienten d_k^1 sind bereits ein Teil des Endergebnisses der diskreten Wavelet-Transformation. Sie sind die Koeffizienten des Wavelet-Spektrums von f auf der Skala 1. Die Projektion $Q_1 f$ lässt sich mit Hilfe der Koeffizienten d_k^1 darstellen

$$Q_1 f = \sum_{k \in Z} d_k^1 \boldsymbol{y}_{1,k} \ . \tag{5.3.7}$$

Aus den nunmehr bekannten Koeffizienten c_k^1 lassen sich mit (5.3.4) und (5.3.5) wiederum die Koeffizienten d_k^2 und c_k^2 der Skala 2 berechnen. Auf diese Weise kann das ganze Wavelet-Spektrum d_k^m rekursiv berechnet werden. Es müssen keine Integrale mehr gelöst werden.
Üblicherweise wird der Rechenvorgang kürzer mit Hilfe der Zerlegungsoperatoren H und G ausgedrückt

$$\begin{aligned} \text{H:} \quad & l^2(\mathbf{Z}) \rightarrow l^2(\mathbf{Z}) \\ & c \rightarrow Hc = \left\{ (Hc)_k = \sum_{l \in Z} h_{l-2k} c_l \right\}, \end{aligned} \tag{5.3.8}$$

$$\begin{aligned} \text{G:} \quad & l^2(\mathbf{Z}) \rightarrow l^2(\mathbf{Z}) \\ & c \rightarrow Gc = \left\{ (Gc)_k = \sum_{l \in Z} g_{l-2k} c_l \right\}. \end{aligned} \tag{5.3.9}$$

Der Algorithmus startet mit der Folge c^0. Diese könnte mit der Gleichung (5.3.2) aus Skalarprodukten berechnet werden. Um den daraus resultierenden großen Rechenaufwand zu vermeiden, wird die Folge c^0 durch Abtastwerte generiert. Dies ist nur eine Näherung der Skalarprodukte. Allerdings ist die Näherung gut, wenn die Abtastschrittweite im Vergleich zu den im Signal enthaltenen Frequenzen klein ist. Dies wird durch das Shannon'sche Abtasttheorem präzisiert, welches sagt, dass ein periodisches Signal nur dann aus seinen Abtastwerten richtig rekonstruiert werden kann, wenn es pro Wellenlänge mindestens zweimal abgetastet wird. Das Abtastintervall T_A hängt also von der größten im Signal vorkommenden Frequenz f_N ab,

$$T_A \leq \frac{1}{2 f_N} .$$

Die Grenzfrequenz f_N wird als Nyquist-Frequenz bezeichnet.
Die gesamte Vorgehensweise wird in Abbildung 5.4 zusammengefasst.

Schnelle Wavelet-Transformation

Eingabe: $c^0 = \{c_k^{\ 0} \mid k \in Z\}$

M Zerlegungstiefe (Anzahl der Skalen)

Berechne: für $m = 1,...,M$

$d^m = Gc^{m-1}$

$c^m = Hc^{m-1}$

Ausgabe: c^M

d^m , $m = 1,...,M$

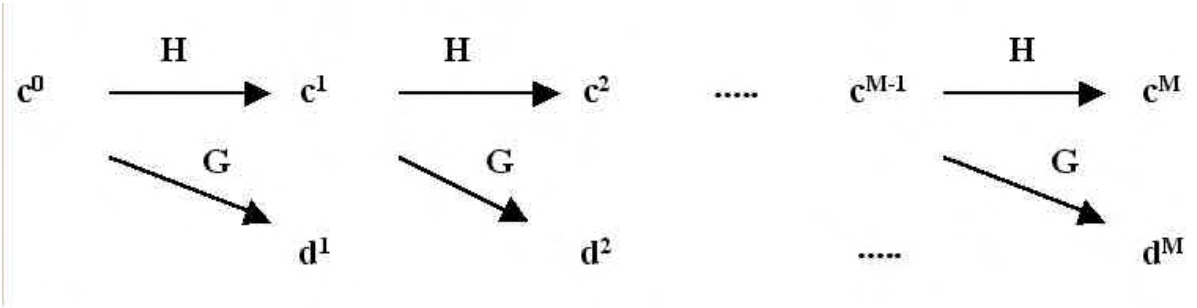


Abbildung 5.4: Ein Schema für die Berechnung der schnellen Wavelet-Transformation

Die Rekonstruktion entspricht der Berechnung der Ausgangsfolge c^0 aus den Koeffizientenfolgen $\{c^M, d^m \mid m = 1,...,M\}$. Auch die Rekonstruktion lässt sich über einen rekursiven Vorgang berechnen. Zunächst wird der letzte Schritt, die Rekonstruktion der Folge c^0 aus den Folgen c^1 und d^1, betrachtet. Die dazugehörigen Räume sind der Raum V_0 und die Unterräume V_1 und W_1, wobei V_1 und W_1 die orthogonale Zerlegung von V_0 sind (vgl. (5.2.9)). Aufgrund der Gleichung (5.2.12) und der Skalierungsgleichungen (5.2.13) und (5.2.15) gilt folgendes

$$\sum_{k\in Z} c_k^0 \boldsymbol{j}_{0,k} = \sum_{j\in Z} c_j^1 \boldsymbol{j}_{1,j} + \sum_{j\in Z} d_j^1 \boldsymbol{y}_{1,j}$$

$$= \sum_{j\in Z} c_j^1 \sum_{l\in Z} h_l \boldsymbol{j}_{0,2j+l} + \sum_{j\in Z} d_j^1 \sum_{l\in Z} g_l \boldsymbol{j}_{0,2j+l}$$

Ein Koeffizientenvergleich ergibt

$$c_k^0 = \sum_{l\in Z} c_l^1 h_{k-2l} + \sum_{l\in Z} d_l^1 g_{k-2l} \,. \tag{5.3.10}$$

Analog kann ausgehend von d^M und c^M zunächst c^{M-1} rekonstruiert werden. Rekursiv können dann die Zerlegungskoeffizienten d^m auf den Skalen M-1,...,1 eingearbeitet werden. Mit den adjungierten Operatoren H^* und G^* von H und G lässt sich der Rekonstruktionsalgorithmus wie folgt schreiben

H^*: $l^2(\mathbf{Z}) \rightarrow l^2(\mathbf{Z})$

$$c \rightarrow H^*c = \left\{ (H^*c)_k = \sum_{l \in Z} h_{2l-k} c_l \right\}, \tag{5.3.11}$$

G^*: $l^2(\mathbf{Z}) \rightarrow l^2(\mathbf{Z})$

$$c \rightarrow Gc = \left\{ (Gc)_k = \sum_{l \in Z} g_{2l-k} c_l \right\}. \tag{5.3.12}$$

Schnelle Wavelet-Rekonstruktion

Eingabe: M Zerlegungstiefe (Anzahl der Skalen)
c^M, d^m , $m = 1,...,M$

Berechne: für $m = M,...,1$
$c^{m-1} = H^* c^m + G^* d^m$

Ausgabe: $c^0 = \{c_k^{\ 0} \mid k \in Z\}$

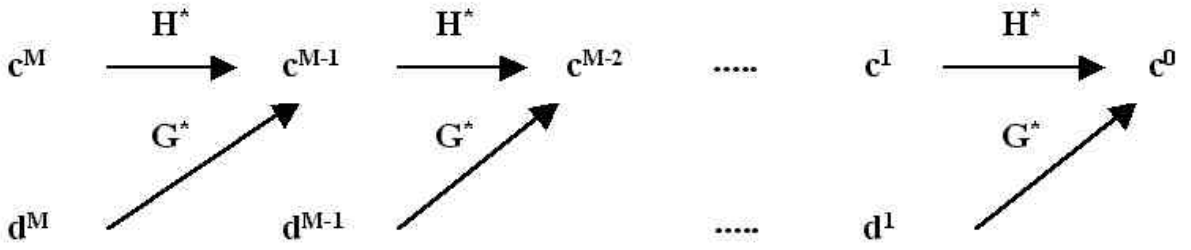


Abbildung 5.5: Ein Schema für die Berechnung der schnellen Wavelet-Rekonstruktion

5.4 Beispiel

Die Theorie der Multi-Skalen-Analyse und der schnellen Wavelet-Transformation soll am Beispiel des Haar-Wavelets noch einmal illustriert werden. Das Haar-Wavelet eignet sich hierfür aufgrund seiner Einfachkeit besonders gut.

Die einfachste Multi-Skalen-Analyse lässt sich mit der Haar'schen Skalierungsfunktion definieren

$$\varphi(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & sonst \end{cases}.$$

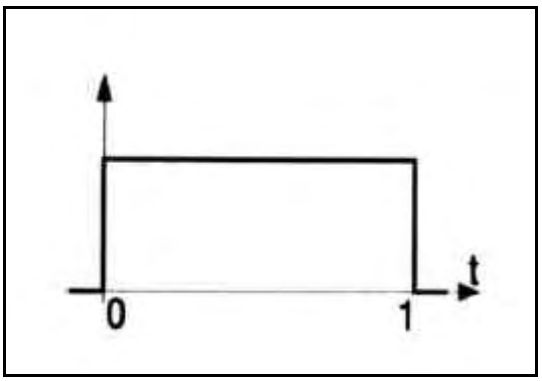


Abbildung 5.6: Haar'sche Skalierungsfunktion

Der Grundraum $V_0 = \overline{span\{\varphi_{0,k} \mid k \in Z\}}$ enthält die Funktionen, die auf den Intervallen[k,k+1[konstant sind. Für allgemeines m gilt

$$V_m = \overline{span\{\varphi_{m,k} \mid k \in Z\}} = \{f \in L^2(\mathbf{R}) : f \text{ ist konstant auf } [2^m k, 2^m(k+1)[\text{ für alle } k \in \mathbf{Z}\}.$$

Die so definierte Familie $\{V_m\}_{m \in \mathbf{Z}}$ erfüllt offensichtlich die Bedingungen (5.2.3) bis(5.2.6) einer Multi-Skalen-Analyse.

Die Skalierungskoeffizienten lassen sich sehr leicht bestimmen. Abbildung 5.7 zeigt, dass sich die Haar'sche Skalierungsfunktion durch gestauchte und verschobene Rechteckfunktionen darstellen lässt.

$$\begin{aligned} \varphi(t) &= \varphi_{1,0}(t) + \varphi_{1,1}(t) \\ &= \varphi(2t) + \varphi(2t+1) \\ &= \sqrt{2}\left(\tfrac{1}{\sqrt{2}}\varphi(2t+0) + \tfrac{1}{\sqrt{2}}\varphi(2t+1)\right) \end{aligned}$$

Mit Gleichung (5.2.13) folgt für die Skalierungskoeffizienten

$$h_0 = h_1 = \frac{1}{\sqrt{2}}, \qquad h_k = 0 \text{ sonst}$$

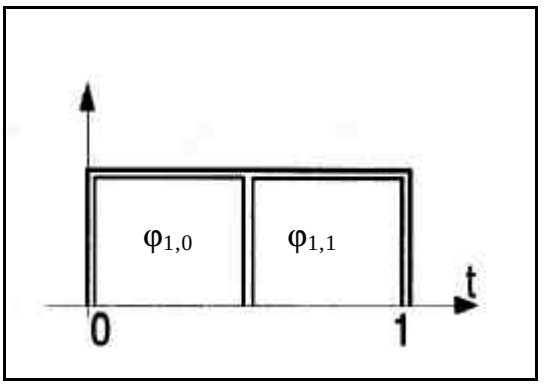


Abbildung 5.7: Darstellung der Haar-Skalierungsfunktion φ durch die Versionen $\varphi_{1,0}$ und $\varphi_{1,1}$

Aus den Skalierungskoeffizienten lässt sich mit Gleichung (5.2.15) leicht das Haar-Wavelet berechnen

$$\psi(t) = \sqrt{2}\left(h_1 \varphi_{1,0}(t) - h_0 \varphi_{1,1}(t)\right) = \begin{cases} 1 & , \quad 0 \le t < 0.5 \\ -1 & , \quad 0.5 \le t < 1 \\ 0 & , \quad \textit{sonst} \end{cases}$$

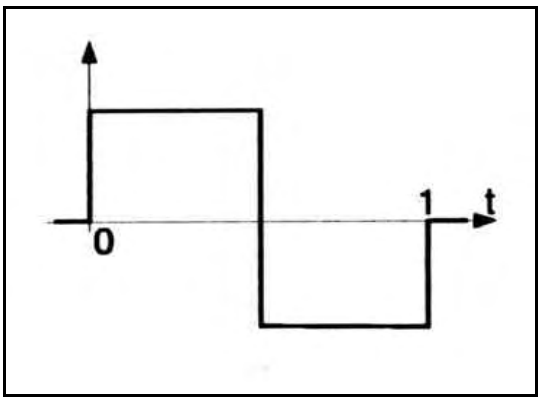


Abbildung 5.8: Haar-Wavelet

Die Mehrfachauflösung soll anhand des Signals in Abbildung 5.9 veranschaulicht werden. Das konstante Signal f wird durch verschobene Rechtecke approximiert (Signal f_0), die den Raum V_0 aufspannen. Mit jeder Stufe der Multi-Skalen-Analyse werden Skalierungsfunktionen mit größerer zeitlicher Auflösung (Rechtecke doppelter Breite) verwendet, die immer größere werdende Approximationen zur Folge haben. Der Fehler, der durch die Approximationen mit jeder Stufe entsteht, wird durch die Detailräume kompensiert, die durch verschobene Wavelets unterschiedlicher Ausdehnung aufgespannt werden. Überlagert man schließlich den am gröbsten auflösenden Approximationsraum V_3 mit den Detailräumen W_1, W_2, W_3, kann man die Genauigkeit des Raumes V_0 zurückgewinnen. Dies folgt aus Gleichung (5.2.9)

$$V_0 = V_3 \oplus W_1 \oplus W_2 \oplus W_3 .$$

Die Räume V_m und W_m werden von den verschobenen Skalierungsfunktionen $\varphi_{m,k}$, $k \in \mathbf{Z}$ und den verschobenen Wavelets $\psi_{m,k}$, $k \in \mathbf{Z}$ aufgespannt. Diese sind in der Abbildung 5.10 zu sehen. Es lässt sich auch leicht erkennen, dass die Räume V_m und die Detailräume W_m orthogonal zueinander sind.

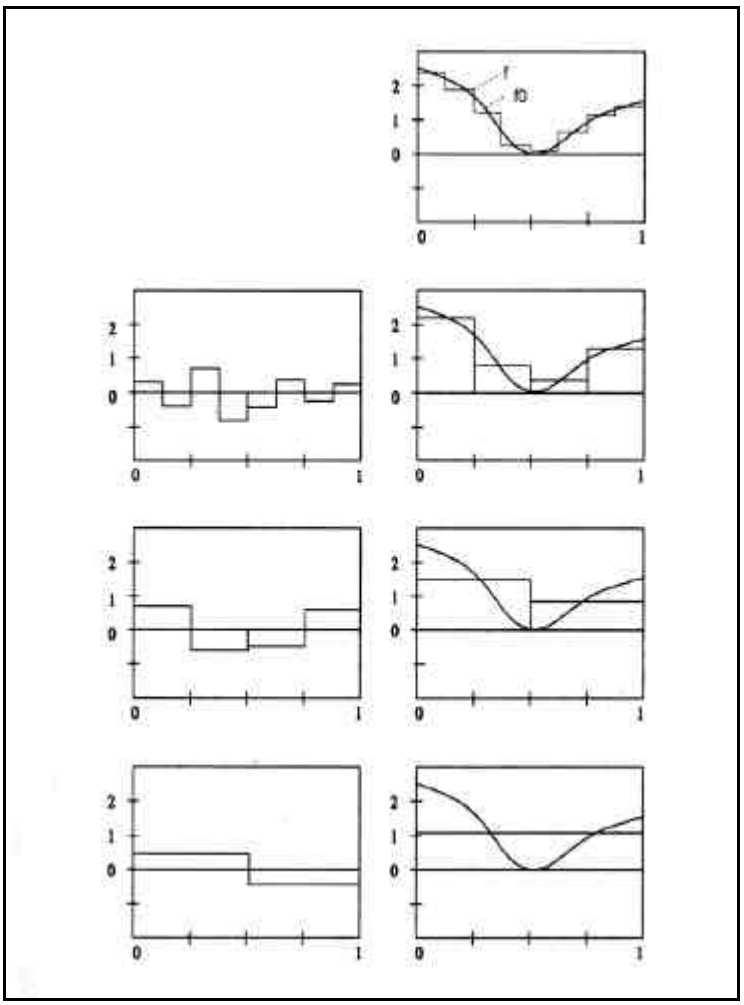


Abbildung 5.9: Mehrfachauflosung der Funktion f_0

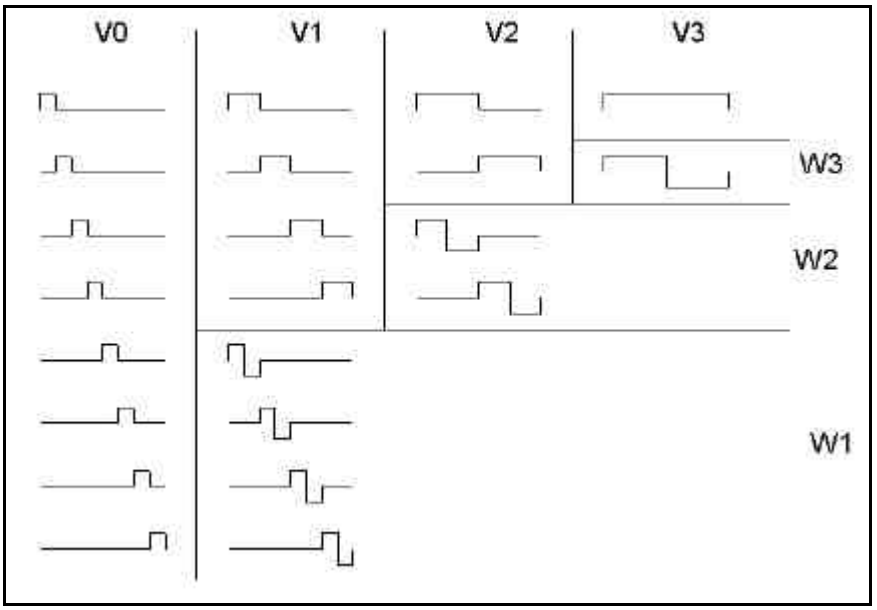


Abbildung 5.10: Die Funktionen der Unter- und der Detailräume

Die Funktion f_0 kann beispielsweise durch eine Abtastung der Funktion f entstanden sein. Aus den Abtastwerten folgt für die Folge c^0 = {2,5; 1,9; 1,0; 0,2; 0,1; 0,5; 1,0; 1,2}. Mit den

Formeln (5.3.4) und (5.3.5) lassen sich die Koeffizienten der diskreten Wavelet-Transformation berechnen. Für eine Zerlegungstiefe M=3 ergeben sich nach dem Schema in Abbildung 5.4 folgende Wavelet-Koeffizienten.

c^0	2,5	1,9	1,0	0,2	0,1	0,5	1,0	1,2
c^1	$\frac{1}{\sqrt{2}} * 4{,}4$		$\frac{1}{\sqrt{2}} * 1{,}2$		$\frac{1}{\sqrt{2}} * 0{,}6$		$\frac{1}{\sqrt{2}} * 2{,}2$	
d^1	**$\frac{1}{\sqrt{2}} * 0{,}6$**		**$\frac{1}{\sqrt{2}} * 0{,}8$**		**$\frac{1}{\sqrt{2}} * (-0{,}4)$**		**$\frac{1}{\sqrt{2}} * (-0{,}2)$**	
c^2	$\frac{1}{2} * 5{,}6$				$\frac{1}{2} * 2{,}8$			
d^2	**$\frac{1}{2} * 3{,}2$**				**$\frac{1}{2} * (-1{,}6)$**			
c^3	$\frac{1}{\sqrt{2}} * \frac{1}{2}$ z$* 8{,}4$							
d^3	**$\frac{1}{\sqrt{2}} * \frac{1}{2} * 2{,}8$**							

Abbildung 5.11: Numerisches Beispiel

Kapitel 6

Eigenschaften von Wavelets

Im Gegensatz zur Fourier-Transformation gibt es bei der Wavelet-Transformation unendlich viele analysierende Funktionen – die Wavelets. Diese gilt es anhand ihrer Eigenschaften einzuteilen.
Möchte man ein Signal analysieren oder es beispielsweise komprimieren, so hängt das Ergebnis von der Wahl des Wavelets ab. Die im Folgenden diskutierten Eigenschaften bilden dabei die Entscheidungskriterien für die Wahl des Wavelets. es sei jedoch angemerkt, dass es für eine Aufgabenstellung nicht eine optimale Wavelet-Basis gibt, da sich die wünschenswerten und die weniger wünschenswerten Eigenschaften gegenseitig bedingen. Man muss abwägen, welche Eigenschaften besonders wichtig sind und mit welchen Nachteilen man leben kann.

Kompakter Träger

Die zeitliche Ausdehnung der Wavelets und der Skalierungsfunktionen wird durch die von Null verschiedenen Koeffizienten g_k und h_k aus den Gleichungen (5.2.15) und (5.2.13) definiert. Um eine gute zeitliche Auflösung zu bekommen, ist man grundsätzlich an Wavelet-Basen mit schmalen kompakten Träger interessiert. Zudem bedeuten wenige Koeffizienten g_k und h_k auch weniger Rechenarbeit bei der schnellen Wavelet-Transformation (vgl. Gleichungen (5.3.2) und (5.3.3)). Leider geht eine Verbesserung der Regularität und des Frequenz-Verhaltens immer einher mit der Vergrößerung des Trägers.

Regularität

Die Wavelet-Transformation eignet sich zur Komprimierung von Signalen. Dabei werden betragsmäßig kleine Wavelet-Koeffizienten auf Null gesetzt. Bei der Rekonstruktion des Signals aus den verbleibenden Koeffizienten ist der verursachte Fehler stark abhängig von der Form der Skalierungsfunktion und des Wavelets. Abbildung 6.1 veranschaulicht die Approximationseigenschaft des Haar-Wavelets (links) und des einfachsten Daubechies-Wavelets (rechts) anhand einer gedämpften Schwingung. Das Resultat trägt deutlich den Stempel dieser Funktionen. Bei der Kodierung einer stetigen Funktion mit einer unstetigen Funktion wie dem Haar-Wavelet entsteht ein unstetiges Abbild, und man erhält Kanten, wo im Original überhaupt keine zu sehen sind. Um dies zu vermeiden hätte man statt des Haar-Wavelets eine reguläre (glatte) Funktion einsetzen müssen. Der Regularitätsgrad ist die Anzahl der stetigen Ableitungen eines Wavelets.

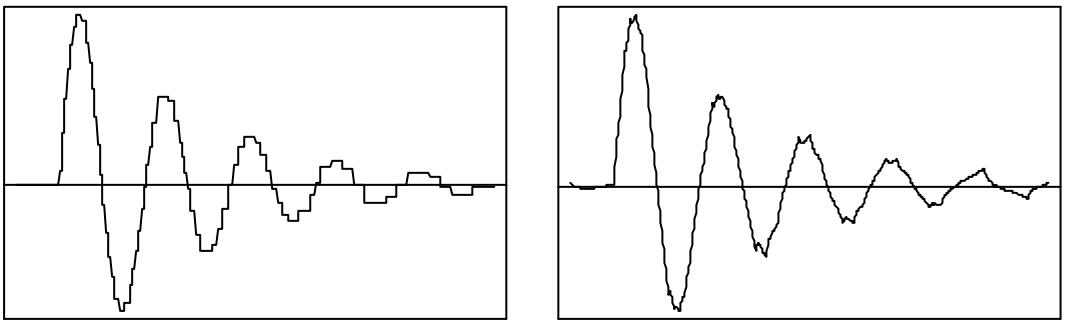


Abbildung 6.1: Komprimierung einer gedämpften Schwingung

Verschwindende Momente

Definition (*k*te Moment):
Das *k*te Moment einer Funktion f ist das Integral über das Produkt dieser Funktion und der *k*ten Potenz der unabhängigen Variablen, also

$$m_k = \int_R f(x) x^k dx .$$

Mit dem Integral verschwindet auch das entsprechende Moment.
Die Anzahl der verschwindenden Momente eines Wavelets steht in gewissem Zusammenhang mit der Anzahl seiner Oszillationen – je mehr verschwindende Momente ein Wavelet besitzt, desto stärker oszilliert es. Der praktische Effekt verschwindender Momente besteht darin, dass die Information in einer relativ kleinen Anzahl von Koeffizienten konzentriert wird. Dies ist beispielsweise sinnvoll bei der Komprimierung von Signalen
Wie auch die Regularität ist auch die Anzahl der verschwindenden Momente stark mit der Differenzierbarkeit des Wavelets verknüpft.

Frequenzverhalten

Die analysierenden Funktionen der Fourier-Transformation – die Sinus- und Kosinusschwingungen - haben eine exakt definierte Frequenz. Wavelets dagegen enthalten eine Mischung verschiedener Frequenzen – man könnte von einer „Frequenzverschmierung“ sprechen. Leider überträgt sich dies auch auf die Wavelet-Koeffizienten.

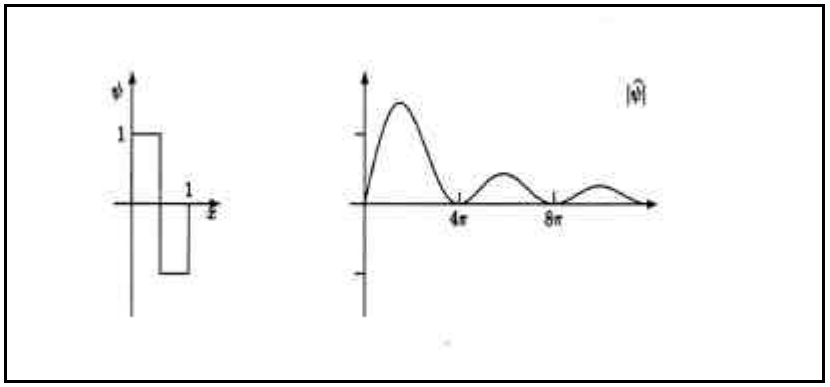


Abbildung 6.2: Das Haar-Wavelet und seine Fourier-Transformierte

Je enger der Frequenzbereich, desto frequenzselektiver ist das Wavelet, und desto präziser wird das Signal in seine Frequenzen zerlegt. Obenstehende Abbildung zeigt das Haar-Wavelet und seine Fourier-Transformierte. der Frequenzbereich ist sehr breit, woraus man schließen kann, dass das Haar-Wavelet keine besonders gute Frequenzauflösung hat. Limitierender Faktor ist auch hier wieder die in Kapitel 4 besprochene Unschärferelation. Wavelets, die sowohl die Zeit gut lokalisieren (wie das Haar-Wavelet mit seinem sehr kompakten Träger) als auch Frequenzen gut auflösen gibt es nicht. Folglich haben frequenzselektive Wavelets einen weniger kompakten Träger.

Kapitel 7

Anwendungen

Wavelets werden in vielen Gebieten wie den Geowissenschaften, der Signalverarbeitung oder auch der numerischen Mathematik angewandt. Neben den in den vorigen Kapiteln beschriebenen guten Eigenschaften der Zeit-Frequenz-Analyse sollen hier noch die Anwendungsmöglichkeiten der Wavelets in der Bildkomprimierung und er Rauschunterdrückung kurz geschildert werden

7.1 Komprimierung

Bilddatenkompression spielt in der modernen Multimedia-Welt eine wichtige Rolle. Geringere Datenmengen sorgen für höhere Übertragungsgeschwindigkeit und geringeren Speicherplatzbedarf. Beides wirkt sich zugunsten des Preises aus.
Als Komprimierungsverfahren zeichnen sich Wavelets durch eine Reihe von Vorteilen aus. Wie in Kapitel 4 gezeigt wurde, registrieren Wavelets nur Änderungen im Signal. Geringe Änderungen zeichnen sich durch kleine Koeffizienten aus. Dadurch eignen sich Wavelets für viele zu komprimierende Bilder. Die Wahrscheinlichkeit, dass sich benachbarte Punkte ähneln ist meistens sehr groß. Strukturarme Bilder haben demnach viele vernachlässigbare Wavelet-Koeffizienten. Damit verringert sich auch die Anzahl der beim Kodieren zu berücksichtigen Koeffizienten ganz wesentlich. Als Vorteil wirkt sich auch die „Kürze“ der Wavelets aus, wodurch sich die Approximationsfehler nur lokal auswirken.
Abbildung 7.1 zeigt das bekannte Beispiel der Kompression von digitalisierten Fingerabdrücken der FBI-Datenbank. Das linke Bild zeigt das Originalbild, das rechte die Rekonstruktion aufgrund einer im Verhältnis 26:1 komprimierten Version.

Abbildung 7.1: Komprimierung digitalisierter Fingerabdrücke

7.2 Rauschunterdrückung durch Thresholding

Verrauschte Signale können mit Hilfe von Wavelets entrauscht werden. Ein bisher oft angewendetes Verfahren ist die Bandpassfilterung, bei der ein bestimmter Frequenzbereich aus dem Signal herausgefiltert wird. Die lässt sich gut anwenden bei Signalen, von denen man weiß, dass das zu eliminierende Rauschen in einem gewissen Frequenzbereich liegt. Ohne diese Vorkenntnis versagt das Verfahren, da man nicht weiß, welcher Frequenzbereich

herauszufiltern ist. Ebenso wenig ist das Verfahren anwendbar, wenn sich die Frequenzen des eigentlichen Signals und des Rauschens überlagern. Würde man in diesem Fall versuchen das Rauchen durch eine Bandpassfilterung zu entfernen, ginge auch die im Signal enthaltene Information verloren. Für viele Anwendungen, wie beispielsweise medizinische Diagnoseverfahren ist ein solcher Informationsverlust nicht tragbar.
Rauschunterdrückung lässt sich mittels Wavelets durch das sogenannte „Thresholding“ bewerkstelligen. Dabei werden alle Wavelet-Koeffizienten des Signals, welche betragsmäßig kleiner als ein gewisser Schwellwert (Threshold) sind, durch Null ersetzt. Dies setzt allerdings voraus, dass das Signal-zu-Rausch-Verhältnis nicht zu klein wird, da sonst das Signal herausgefiltert wird und nicht das Rauschen. Im Gegensatz zur Bandpassfilterung werden beim Thresholding nicht ganze Frequenzbänder eliminiert, sondern nur Signalanteile mit einer verhältnismäßig kleinen Amplitude. Das Thresholding kann auch auf die Fourier-Koeffizienten angewendet werden, aber mit den Wavelets erhält man wegen ihrer lokalen Natur meist bessere Resultate.
Einen Vergleich zwischen Bandpassfilterung und Thresholding zeigt die Abbildung 7.2. Dem Signal (Bild a) wurde ein Rauschen überlagert (Bild b). Das Rauschen ist im Frequenzbereich nicht eindeutig vom Signal getrennt. Bild c ist das Resultat einer Bandpassfilterung, das Bild entsteht durch Thresholding der Fourier-Koeffizienten. Das letzte Bild entsteht schließlich durch Thresholding der Wavelet-Koeffizienten.

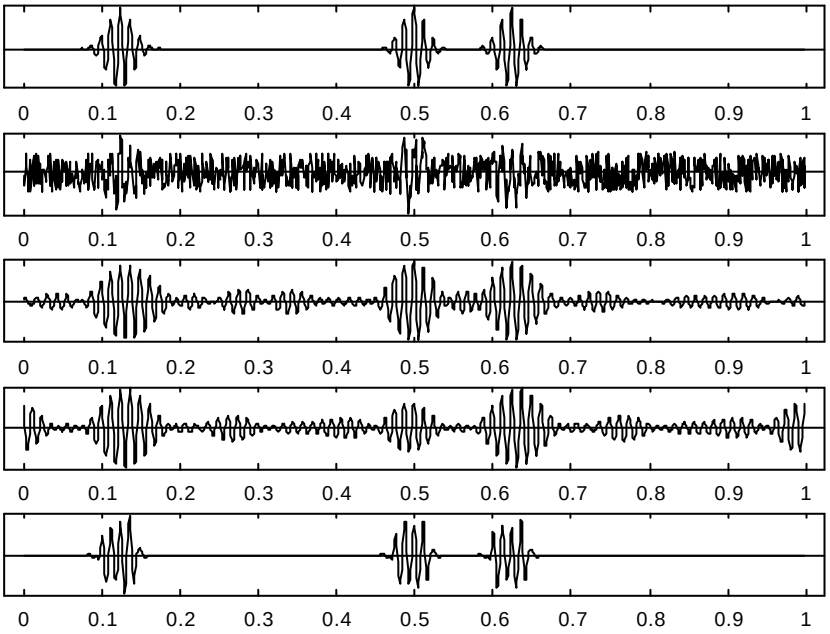


Abbildung 7.2: Bandpassfilterung und Thresholding

Kapitel 8

Die Programmiersprache Java

Kaum eine andere objektorientierte Programmiersprache hat in den vergangenen Jahren so viel Aufmerksamkeit erregt wie Java. Dies liegt sicherlich daran, dass Java im Gegensatz zu anderen objektorientierte Programmiersprachen plattformunabhängig ist. Dies bedeutet, dass Java-Programme auf jedem Computer – unabhängig von Prozessor und Betriebssystem – ausgeführt werden können. Daraus resultiert ein wesentlicher Vorteil von Java. Mehrfachentwicklungen von Programmen können entfallen, wenn die Programme von jedem System ausgeführt werden können. Den großen Bekanntheitsgrad erreichte Java wegen seinen internetfähigen Programmen, den Applets. Java-Applets können von nahezu jedem Webbrowser ausgeführt werden und geben dem Anwender die Möglichkeit mit den Internetseiten interaktiv zu agieren. Java hat sich seit seiner offiziellen Einführung im Januar 1996 sehr schnell zu einer der wichtigsten Sprachen des Internets entwickelt.
Das Kapitel 8.1 werden kurz die wesentlichen Merkmale objektorientierte Programmiersprachen erläutert. Objektorientierte Programmiersprachen stellen die neueste Entwicklung auf dem Gebiet der Programmiersprachen dar. Im anschließenden Kapitel 8.2 werden die Möglichkeiten der Erstellung von grafischen Benutzeroberflächen mittels Java beschrieben.

Plattformunabhängigkeit

Jedes Programm einer Programmiersprache muss zunächst in Maschinensprache übersetzt werden, bevor es von einem Computer ausgeführt werden kann. Die Übersetzung erfolgt durch spezielle Programme. Es existieren hierfür zwei verschiedene Konzepte. Ein Interpreter übersetzt das Programm während seiner Ausführung Stück für Stück in die Maschinensprache. Dies führt allerdings zu einer geringen Ausführungsgeschwindigkeit der Programme. Eine bessere Ausführungsgeschwindigkeit wird erreicht, wenn die Programme vor ihrer Ausführung mit einem Compiler in Maschinensprache übersetzt werden. Java bildet eine Mischform zwischen Interpreter- und Compiler-Sprache. Der Java-Compiler erzeugt anstelle eines Programms in Maschinensprache einen Zwischencode den sogenannten Byte-Code. Der Java-Interpreter übersetzt den Byte-Code zur Laufzeit in Maschinencode und führt diesen schließlich aus. Im Gegensatz zum Maschinencode ist der Java-Byte-Code plattformunabhängig und kann auf jedem Computersystem ausgeführt werden, das über eine Java-Virtual-Machine verfügt. Die Plattformunabhängigkeit ist Voraussetzung für die Verwendung von Java im Internet.

8.1 Objektorientierte Programmierung

Bei der objektorientierten Programmierung handelt es sich um ein Konzept der Softwareentwicklung, das sich grundlegend von den bisherigen Programmiersprachen unterscheidet. Anders als beim prozeduralen Ansatz (z.B. in PASCAL), der Probleme in einzelne Teilfunktionen zerlegt (funktionale Zerlegung), stellte der objektorientierte Ansatz die Objekte des Problembereichs in den Vordergrund (objektorientierte Zerlegung). Objektorientierte Programme bestehen aus miteinander kommunizierenden Objekten, die gemeinsam die Funktionalität des Programms realisieren.

Java unterstützt als objektorientierte Sprache alle Konzepte der objektorientierten Programmierung, wie

- Klassenbildung,
- Vererbung,
- Polymorphie,
- Abstraktion,
- und Kapselung.

Diese sollen im Folgenden kurz erläutert werden.

Klassenbildung

Objekte zeichnen sich dadurch aus, dass sie sowohl gewissen Attribute besitzen , als auch ein bestimmtes Verhalten aufweisen. Attribute bezeichnen den Zustand eines Objekts. Folglich handelt es sich bei den Attributen um Daten. Das Verhalten eines Objekt beschreibt, wie ein Objekt reagiert. Somit handelt es sich beim Verhalten eines Objekts um dessen Funktionen (Funktionen werden in der Terminologie der objektorientierten Programmierung als Methoden bezeichnet.).
Objekte mit gleichen Eigenschaften und gleichem Verhalten bilden eine sogenannte Klasse. Die gemeinsamen Eigenschaften und Funktionen werden durch die Klassendefinition beschrieben. Sie ist der „Bauplan“ für entsprechende Objekte. Ein Objekt ist eine konkrete Ausprägung einer Klasse. Hierfür wird auch der Begriff Instanz einer Klasse verwendet.
Ein kleines Beispiel soll die Instanzbildung veranschaulichen. Die Klasse heißt Auto, und umfasst alle Autos mit ihren Eigenschaften wie Farbe, Leistung und Sitzplätze und Funktionen wie beispielsweise beschleunigen. Das Objekt „meinAuto“ ist eine Instanz der Klasse Auto mit den konkretisierten Eigenschaften und seinem Verhalten.

Abbildung 8.1: Instanzierung

Die Gesamtfunktionalität eines Programms wird in der objektorientierten Programmierung durch Kommunikation zwischen einzelnen Objekten realisiert. Objekte kommunizieren miteinander, indem sie sich gegenseitig Nachrichten senden. Bei diesen Nachrichten handelt es sich um Methodenaufrufe. Im Beispiel könnte das Objekt „meinAuto“ vom Objekt „fahrerA“ die Nachricht „beschleunigen“ erhalten. Das Objekt „fahrerA“ löst also beim Objekt „meinAuto“ durch die Nachricht „beschleunigen“ eine Funktion aus.
Neben den benutzerdefinierten Klassen sind im Lieferumfang von Java eine Vielzahl von sogenannten Packages enthalten, die zusammen die Java-Klassenbibliothek bilden. Sie enthält Klassen aus zahlreichen Anwendungsgebieten, die in den jeweiligen Programmen verwendet werden können

- Mathematische Funktionen,
- Input- /Output-Funktionen
- GUI-Elemente
- Hilfsklassen
- usw.

Vererbung, Polymorphie und Abstraktion

Eines der wichtigsten Grundkonzepte der objektorientierten Programmierung ist die Vererbung. Nach dem Prinzip der Vererbung können einzelne Klassen (Oberklassen) als Grundlage von neuen Klassen - sogenannten Unterklassen – verwendet werden. Dabei erben die Unterklassen die Eigenschaften sowie das Verhallten ihrer jeweiligen Oberklassen. Die Unterklassen können das geerbte Verhalten und die geerbten Eigenschaften erweitern oder spezialisieren. Die Oberklasse und die daraus abgeleiteten Unterklassen bilden eine Klassenhierarchie, die am Beispiel der Oberklasse Fahrzeug veranschaulicht wird.

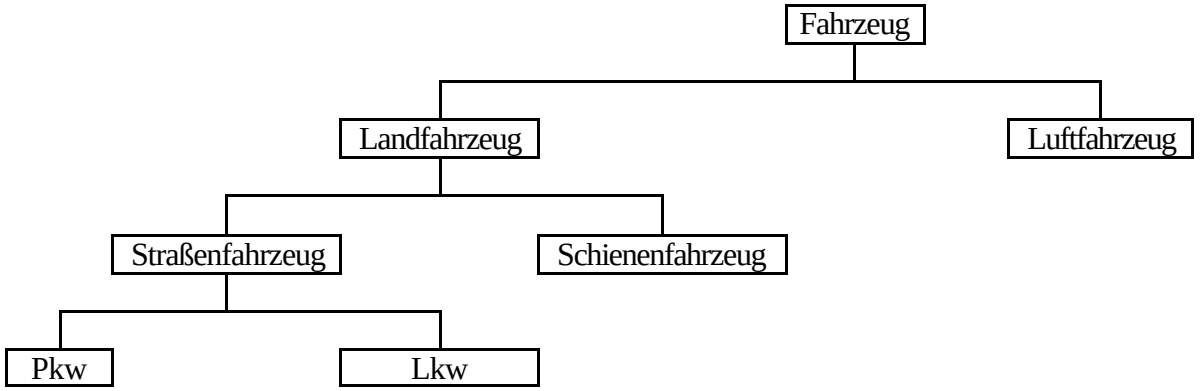


Abbildung 8.2: Klassenhierarchie

Das Konzept der Klassenbildung erlaubt in Verbindung mit dem Konzept der Vererbung die Bildung von Abstraktionen. Dabei werden die gemeinsamen Eigenschaften und das gemeinsame Verhalten der Unterklassen in der Oberklasse zusammengefasst. Im Beispiel wird das Verhalten Beschleunigen und Abbremsen in der Oberklasse Fahrzeug zusammengefasst. Wie das entsprechende Verhalten dann tatsächlich realisiert wird, wird erst in den Unterklassen festgelegt. Die Verhaltensweisen „Beschleunigen“ und „Abbremsen“ funktionieren beispielsweise bei Autos und Flugzeugen unterschiedlich. Den Sachverhalt, dass Instanzen verschiedener Klassen (Auto, Flugzeug) auf die gleiche Nachricht (Beschleunigen) unterschiedlich reagieren, bezeichnet man als Polymorphie. Klassen, die so allgemein sind, dass keine Instanzen von ihnen gebildet werden können, werden als abstrakte Klassen bezeichnet. Sie stellen lediglich ein allgemeines Konzept dar. Aus den abstrakten Klassen abgeleitete Klassen erweitern und spezialisieren das Verhalten und die Eigenschaften der abstrakten Klassen. Sie werden auch als konkrete Klassen bezeichnet.

Kapselung

Eines der wesentlichen Konzepte der objektorientierten Programmierung ist die Kapselung. Durch die Kapselung wird verhindert, dass die Attribute eines Objekts direkt manipuliert werden. Um die Attribute zu schützen, werden durch sogenannte Modifier Zugriffsebenen festgelegt. Die Zugriffsebene gibt an, ob andere Klassen auf die jeweiligen Variablen zugreifen können, oder ob diese für andere Klassen unsichtbar sind.

Klassenbildung, Vererbung und Abstraktion machen die Aufgabenstellung bei der Programmierung überschaubar, da komplexe Probleme in Teilprobleme zerlegt werden. Durch Vererbung können bestehende Programmteile wiederverwendet werden. Dies erhöht die Effizienz der Programmentwicklung, aber auch die Qualität der Programme, wenn man auf bewährte Bausteine zurückgreifen kann. Das Prinzip der Kapselung erhöht die Unabhängigkeit der einzelnen Programmbestandteile, wodurch eine geringere Fehleranfälligkeit erreicht werden kann.

8.2 Grafische Benutzeroberflächen

Jedes moderne Betriebssystem bietet die Möglichkeit Programme mittels grafikorientierter Ein- und Ausgabe zu bedienen. Grafische Benutzeroberflächen – auch GUIs[3] genannt – sollen den Umgang mit komplexen Programmen vereinfachen. Java stellt zwei umfangreiche Bibliotheken für GUI-Komponenten zur Verfügung. In den Versionen 1.0 und 1.1 war das Abstract Windowing Toolkit (AWT) die Standardbibliothek. Mit Java 1.2 wurde eine neue Bibliothek namens Swing eingeführt.
Ein Großteil der Grafikfunktionen ist direkt im Betriebssystem implementiert und über entsprechende Schnittstellen aufrufbar. Auch wenn sich die konkreten Schnittstellen je nach Betriebssystem stark unterscheiden, so sind doch die grundlegenden Konzepte meistens sehr ähnlich. So gibt es beispielsweise in jedem Betriebssystem Fenster mit Menüs, Buttons, Textfelder usw. mit denen die Interaktion zwischen Anwender und Programm möglich ist. Das AWT bietet eine auf Java basierende Schnittstelle zu den grafischen Möglichkeiten eines Betriebssystems. Das erklärte Ziel ist es, die Entwicklung plattformunabhängiger grafischer Benutzteroberflächen mit Hilfe von AWT zu ermöglichen. Um die Plattformunabhängigkeit zu garantieren stellt das AWT nur GUI-Komponenten zur Verfügung, die in allen gebräuchlichen Betriebssystemen implementiert sind. (GUI-Komponenten, die im Betriebssystem implementiert sind, werden auch als heavyweight-Komponenten bezeichnet, und dementsprechend werden GUI-Komponenten, die nicht im Betriebssystem implementiert sind auch als lightweight-Komponenten bezeichnet.) Die GUI-Komponenten des AWT bilden sozusagen das kleinste gemeinsame Nenner der heavyweight-Komponenten aller gebräuchlichen Betriebssysteme. Daraus resultieren einige Einschränkungen. Raffinierte Steuerungselemente z.B., die in einem Betriebssystem unterstützt werden, in einem anderen dagegen nicht, sind auch im AWT nicht zu finden. Auf dem AWT basierende Programme sind zwar plattformunabhängig, können aber häufig nicht mit plattformspezifischen Programmen mithalten. Eine Lösung dieses Problems bieten die Swing-Klassen. Mit Swing können anspruchsvolle grafische Benutzeroberflächen auf der Basis von lightweight-Komponenten erstellt werden. Das bedeutet, dass jeder grafische Inhalt, den Swing-Komponenten darstellen, vom Java-Code gezeichnet wird. Wie die Erscheinungsweise und das Verhalten einer Komponente aussieht, bestimmt somit deren Java-Implementierung. Die Swing-Bibliothek umfasst mehr als 500 Klassen. Diese stellen ein umfangreiches Werkzeug zur Programmierung von grafischen Benutzeroberflächen dar. Seine Funktionen entsprechen dem gegenwärtigen Stand der Technik.
Swing hat das AWT bei der Erstellung grafischer Benutzeroberflächen weitgehend ersetzt. Die Ausnahme bilden die Applets. Diese verwenden oft AWT-Komponenten, um mit bestehenden Webbrowsern, die Swing (noch) nicht unterstützen, kompatibel zu sein.

[3] Graphical User Interface

Kapitel 9

Die Java-Applications

Im Rahmen der Diplomarbeit wurden zwei Java-Programme (Application) geschrieben, die zur Signalanalyse mittels Wavelets dienen. Die erste Java-Application verwendet die kontinuierliche Wavelet-Transformation, die zweite die diskrete Wavelet-Transformation. Die folgenden zwei Abschnitte sollen einen Überblick darüber geben, wie die Programme funktionieren und welche Eigenschaften der Wavelet-Transformation mit den Programmen veranschaulicht werden können. Dabei wird auf die Wavelet-Theorie aus den Kapiteln 4 bis 7 Bezug genommen.
Die grafischen Benutzeroberflächen beider Java-Applications entstanden mit Hilfe der GUI-Komponenten der Swing-Klassen. Am Ende der Abschnitte wird in kurzer Form auf die internetfähigen Versionen der beiden Programme eingegangen. Im Gegensatz zu den Applications wurden die Applets mit den GUI-Elementen des AWT programmiert.

9.1 Die kontinuierliche Wavelet-Transformation

Das Programm CWT.java ermöglicht die Signalanalyse mit Hilfe der kontinuierlichen Wavelet-Transformation. Der erste Schritt ist das Laden des Signals. Dabei ist darauf zu achten, dass Signale nur aus ASCII-Dateien geladen werden können. Des weiteren dürfen in der Datei nur die Ordinatenwerte des Signals abgespeichert sein. Das Programm geht davon aus, dass der Abstand der Abszissenwerte konstant ist. Die Anzahl der Signalpunkte ist auf 512 begrenzt. Das geladene Signal wird im oberen Feld dargestellt (vgl. Abbildung 9.1).

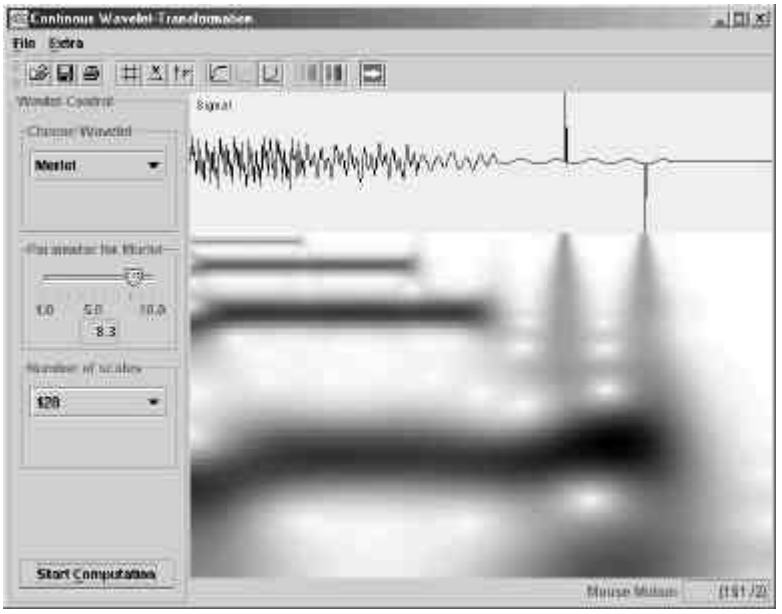


Abbildung 9.1: Die grafische Benutzeroberfläche des CWT-Programms

Zur Analyse des Signals stehen drei verschiedene Wavelets zur Verfügung:

- das Haar-Wavelet,
- das Mexikanerhut-Wavelet,
- und das Morlet-Wavelet.

Die mit Hilfe des gewählten Wavelets berechnete Transformation wir unter dem Signal grafisch dargestellt. Dabei wird die Größe der Wavelet-Koeffizienten durch eine farbliche Kodierung visualisiert. Rote Gebiete bedeuten kleine Wavelet-Koeffizienten, während blaue Gebiete große Wavelet-Koeffizienten bedeuten.
Nach Kapitel 4.2 müsste man theoretisch unendlich viele Koeffizienten berechnen. Tatsächlich werden die Koeffizienten nur an diskreten Positionen b und für diskrete Skalen a berechnet. Die maximale Anzahl von 512 Signalpunkten bildet hierbei auch die Anzahl der Positionen, an denen die Wavelet-Koeffizienten berechnet werden. Die Anzahl der Skalen kann der Anwender selber bestimmen, wobei die obere Grenze bei 128 Skalen liegt. Wird die maximale Anzahl von Skalen gewählt, so werden 512*128=65 536 Wavelet-Koeffizienten durch Skalarproduktbildung berechnet (vgl. Gleichung (4.2)). Um die daraus resultierenden lagen Rechenzeiten zu vermeiden, kann eine kleinere Anzahl von Skalen ausgewählt werden, was sich allerdings bei der Auflösung der Spektrum-Grafik bemerkbar macht.

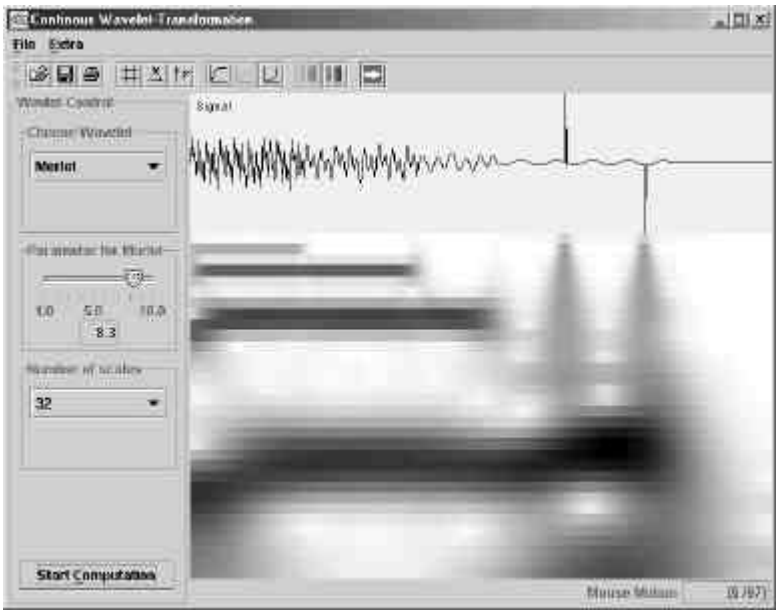


Abbildung 9.2: Wavelet-Spektrum mit nur 32 verschiedenen Skalen

Obwohl das Programm die kontinuierliche Wavelet-Transformation nur an diskreten Stellen berechnet, ist die Transformation hochredundant. Die Wavelets überlappen sich. Dadurch ist die meiste in einem herausgegriffenen Koeffizienten enthaltene Information auch in seinen Nachbarkoeffizienten enthalten. Dies kann man in der Abbildung 9.1 daran erkennen, dass sich die Verfärbungen, die der Größe der Wavelet-Koeffizienten entsprechen, sich langsam, kontinuierlich verändern. Oder anders ausgedrückt, die Verfärbung eines Punktes ist der seiner Nachbarpunkte sehr ähnlich.
Mit Hilfe des Programms lassen sich die Eigenschaften der drei zur Verfügung stehenden Wavelets sehr gut visualisieren. Hier sollen die Unterschiede zwischen dem Morlet-Wavelet und dem Haar-Wavelet gezeigt werden. Abbildung 9.1 zeigt die Wavelet-Transformation unter Verwendung des Morlet-Wavelets (Parameter 7,1). Für das gleiche Signal wurde die Wavelet-Transformation mit dem Haar-Wavelet als analysierende Funktion berechnet (Abbildung 9.3). Das Morlet-Wavelet vermag die im Signal enthaltenen Frequenzen sehr gut aufzulösen. Es ist deutlich zu sehen, dass das Signal aus vier verschiedenen Frequenzen besteht, die jeweils unterschiedlich lang anhalten. Wie aus Abbildung 9.3 ersichtlich wird, ist das Haar-Wavelet nicht sehr frequenzselektiv. Wie schon in Kapitel 6 beschrieben wurde, ist der Frequenzbereich des Haar-Wavelets sehr breit. Mit ihm können die im Signal enthaltenen

Frequenzen nicht unterschieden werden. Dafür hat das Haar-Wavelet einen sehr kompakten – also sehr kurzen – Träger, wodurch die zeitlich Auflösung sehr gut ist. Zu erkennen ist dies sehr deutlich an den beiden Impulsen im letzten Drittel des Signals. Die Impulse werden mit dem Haar-Wavelet zeitlich exakt aufgelöst, während das Morlet-Wavelet mit seinem weniger kompakten Träger nur eine verschwommene Aussage über die Zeitpunkte der Impulse zulässt.

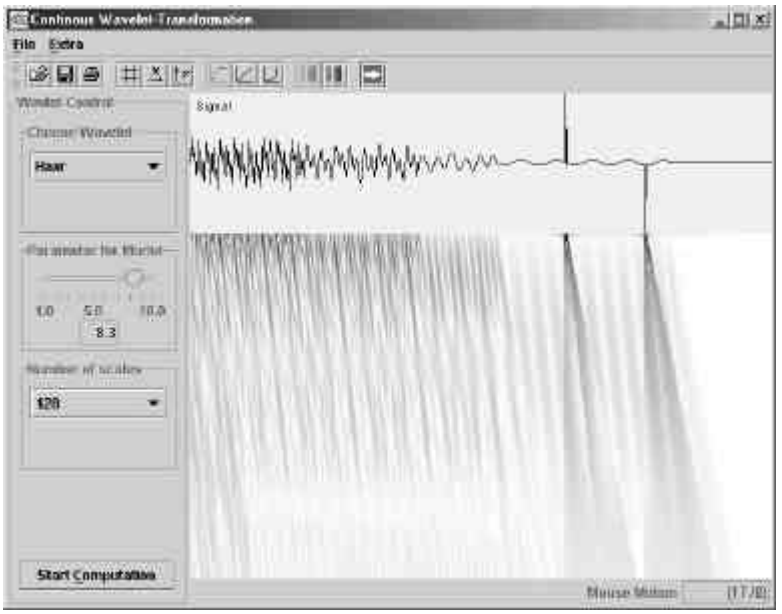


Abbildung 9.3: Kontinuierlich Wavelet-Transformation mit dem Haar-Wavelet

In Kapitel 4.1 wurde die Zeit-Frequenz-Auflösung der Wavelet-Transformation diskutiert. Dabei wurde festgestellt, dass mit breiterwerdenden Wavelets zwar die Frequenzauflösung steigt, dafür aber die zeitlich Auflösung abnimmt. Das Signal in Abbildung 9.4 besteht lediglich aus zwei Impulsen. Die Transformation wurde mit Hilfe des Morlet-Wavelets durchgeführt. Wie man erkennen kann, wird die zeitlich Auflösung zum unteren Rand der Spektrum-Grafik immer schlechter. Dies liegt daran, dass der Skalenparameter a größer wird, je weiter man nach unten geht. Große Skalenparameter a bedeuten nach Gleichung (4.4) breite Wavelet, die – aufgrund ihrer Breite – die Zeit immer weniger gut auflösen.
Eine weitere wesentlich Eigenschaft von Wavelets wird in Abbildung 9.4 verdeutlicht. Das Signal besteht außer den zwei Impulsen aus kontinuierlichen Signalabschnitten. Da Wavelets wegen Gleichung (4.5) nur Veränderungen im Signal wahrnehmen, sind die Wavelet-Koeffizienten für die konstanten Signalabschnitte gleich Null. Dies erscheint zwar trivial, ist aber eine der wichtigsten Eigenschaften der Wavelets, die beispielsweise bei der Bildkomprimierung genutzt wird (vgl. Kapitel 7.1).

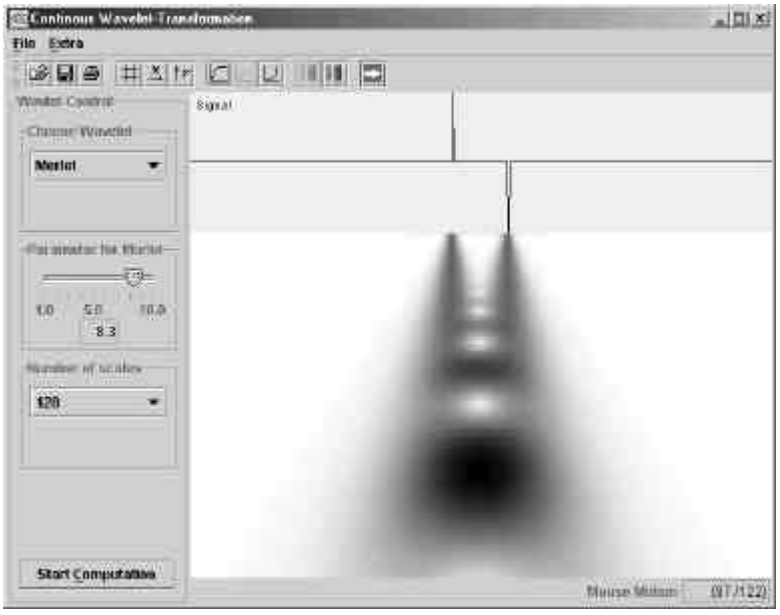


Abbildung 9.4: Zeitauflösung bei breiterwerdenden Wavelets

Bei der Verwendung des Morlet-Wavelets kann mit einem Parameter die Zeit-Frequenz-Auflösung variiert werden. Ein großer Morlet-Parameter bietet eine gute Frequenzauflösung, aber eine schlechte zeitliche Auflösung, während ein kleiner Morlet-Parameter eine gute Zeitauflösung aber eine schlechte Frequenzauflösung liefert. Das Morlet-Wavelet entsteht aus einer Kosinusschwingung und einer komplexen Sinusschwingung, die mit der Gauß'schen Glockenkurve multipliziert werden

$$y(t) = e^{iw_0 t} \cdot e^{-t^2/2}.$$

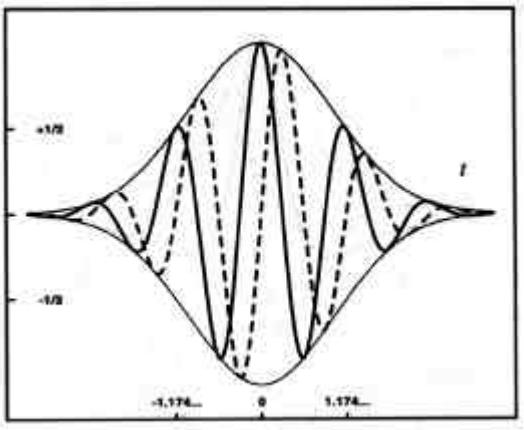


Abbildung 9.5: Das Morlet-Wavelet

Der Parameter ω_0 bestimmt die Frequenz der Kosinus- bzw. der Sinusschwingung. Ist ω_0 klein, so ist auch der Träger des Wavelets klein und die zeitliche Auflösung ist gut. Dies lässt sich in Abbildung 9.6 erkennen. Für einen Morlet-Parameter von 1,7 werden die zwei Impulse des Signals sehr gut aufgelöst. Die zeitliche Auflösung ist für den kleinsten Skalenparameter a am besten. Um den Bereich des kleinsten Skalenparameters a besser sehen zu können ist in Abbildung 9.7 das gleiche Wavelet-Spektrum mit nur 16 Skalen dargestellt.

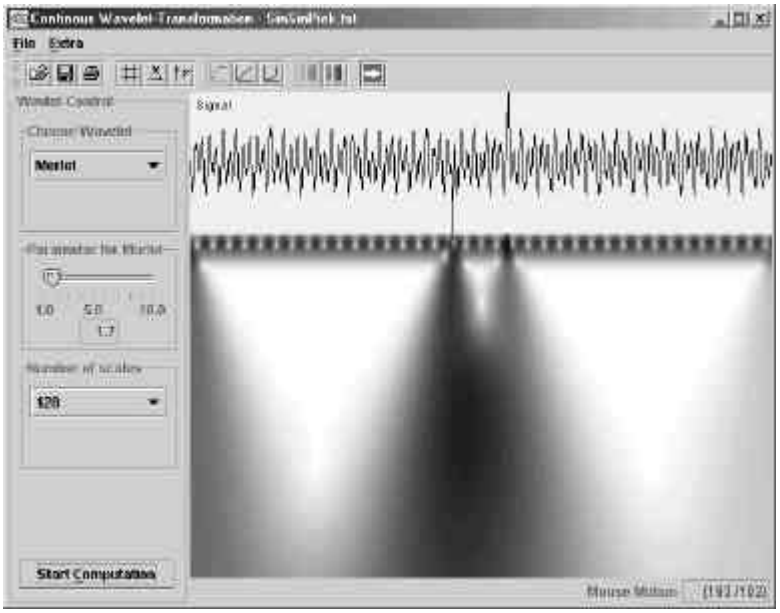


Abbildung 9.6: Morlet-Parameter: 1,7

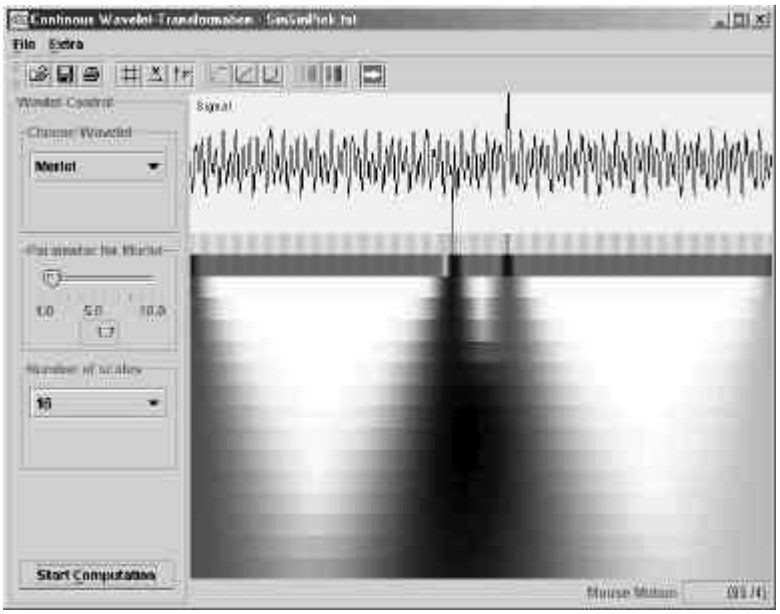


Abbildung 9.7: Morlet-Parameter: 1,7, 16 Skalen

Gute zeitliche Auflösung und gute Frequenzauflösung lassen sich nicht gleichzeitig erreichen. Dies wurde in den Kapiteln 4.1 und 6 diskutiert. Um die Frequenzen des Signals besser auflösen zu können, muss ein größerer Parameter ω_0 gewählt werden (Abbildungen 9.8 und 9.9). Wie man erkennen kann, werden die zwei Frequenzen des Signals deutlich besser aufgelöst. Dagegen ist die zeitliche Auflösung der Impulse wesentlich schlechter, was bei der kleinsten Skala a in Abbildung 9.9 zu erkennen ist.

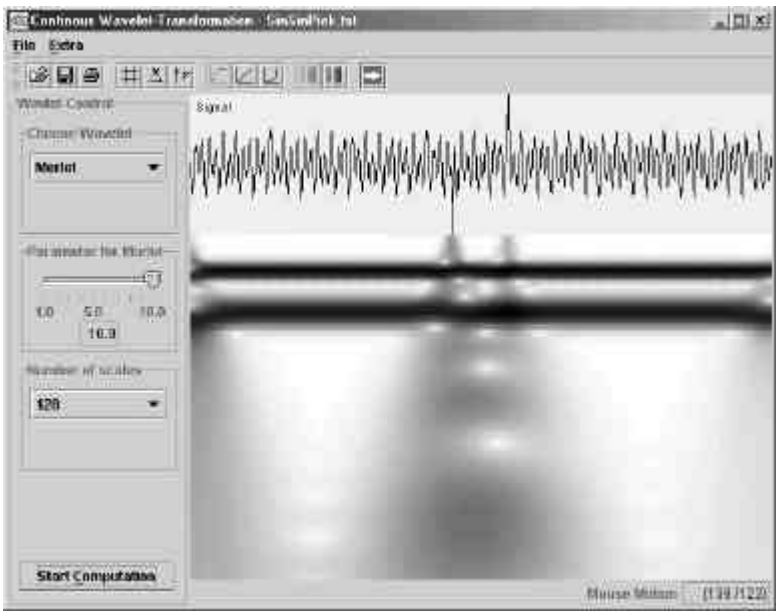


Abbildung 9.8: Morlet-Parameter: 10,0

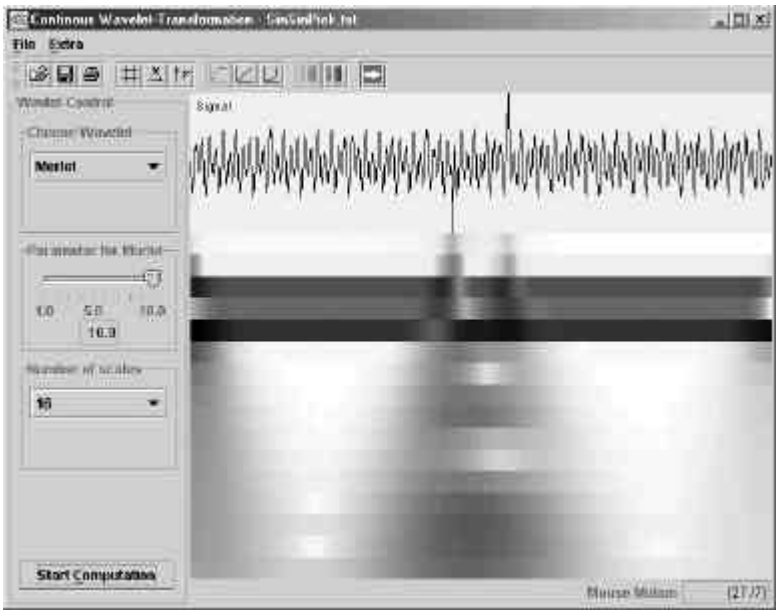


Abbildung 9.9: Morlet-Parameter: 10,0, 16 Skalen

Das Programm bietet nützliche Werkzeuge, die die Signalanalyse und den Umgang mit dem Programm erleichtern sollen. Zur besseren Orientierung können die X- und die Y-Achse sowie ein Koordinatengitter eingeblendet werden. Außerdem wird in der rechten unteren Ecke die Position des Mauszeigers innerhalb der Spektrum-Grafik angezeigt.
Die farbliche Kodierung der Wavelet-Koeffizienten kann durch drei verschiedene Stufen gespreizt werden (linear, exponentiell und logarithmisch). Dadurch kann die farbliche Kodierung dem aktuellen Wavelet-Spektrum optimal angepasst werden. So können beispielsweise farblich kaum erkennbare Wavelet-Koeffizienten durch die logarithmische Spreizung verstärkt werden, damit sie vom Hintergrund deutlicher zu unterscheiden sind. Des weiteren kann man zwischen Farb- und einer Graustufendarstellung wählen. Dabei bietet der Graustufenmodus lediglich beim ausdrucken auf Schwarz/Weiß-Druckern einen wesentlichen Vorteil. Beim Ausdrucken der Signal- und der Spektrumgrafik ist darauf zu achten, dass immer das gedruckt wird, was auf dem Bildschirm zu sehen ist. Ein farbiger Ausdruck ist im

Anhang zu finden. Schließlich können die Wavelet-Koeffizienten in einer ASCII-Datei abgespeichert werden.

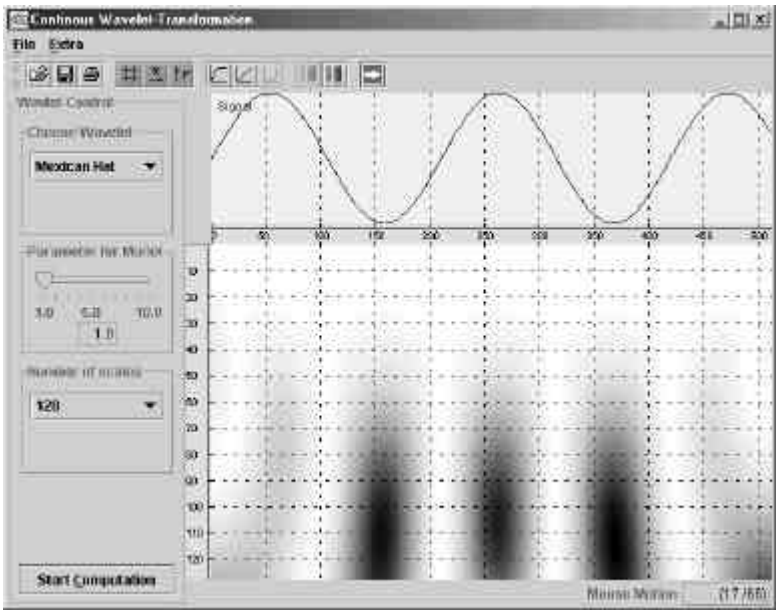


Abbildung 9.10: Koordinatenachsen und Koordinatengitter

9.2 Die diskrete Wavelet-Transformation

Das Programm der diskreten Wavelet-Transformation ist nicht nur ein weiteres Werkzeug zur Signalanalyse, sondern bietet auch die Möglichkeit Signale mittels „Thresholding“ zu manipulieren. Dadurch kann das Programm – wie in Kapitel 7.2 gezeigt – zur Rauschunterdrückung eingesetzt werden.
Zur Signalanalyse und Signalmanipulation bietet das Programm vier verschiedene Wavelets, die jeweils eine Orthonormalbasis nach Gleichung (5.1.6) bilden. Neben dem Haar-Wavelet stehen noch drei Daubechies-Wavelets zur Verfügung, welche in den folgenden Abbildungen dargestellt sind.

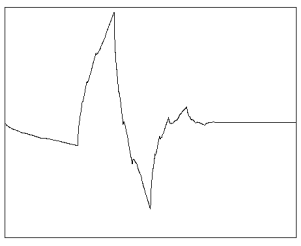


Abbildung 9.11: Daubechies 4

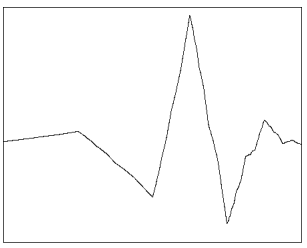


Abbildung 9.12: Daubechies 6

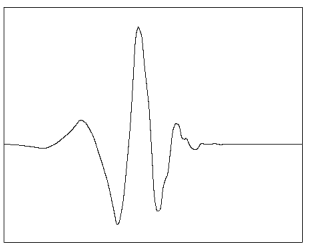


Abbildung 9.13: Daubechies 8

Abbildung 9.14 zeigt ein verrauschtes Sinus-Signal und sein diskretes Wavelet-Spektrum. Das Spektrum ist in farblich abgestufte Bereiche unterteilt, die zu den jeweiligen Skalen gehören. Auf der rechten Seite sind die Wavelet-Koeffizienten d^1 der Skala 1 abgebildet, dann folgen (im links anschließenden Bereich) die Koeffizienten d^2 der Skala 2, usw. Die Koeffizienten d^m lassen sich mit Hilfe der schnellen Wavelet-Transformation nach Gleichung (5.3.4) berechnen. Dafür wird nach dem Schema in Abbildung 5.4 zum Start eine Folge c^0 benötigt. Das Programm verwendet die Ordinatenwerte des geladenen Signals als Folge c^0 und berechnet daraus die Wavelet-Transformation.

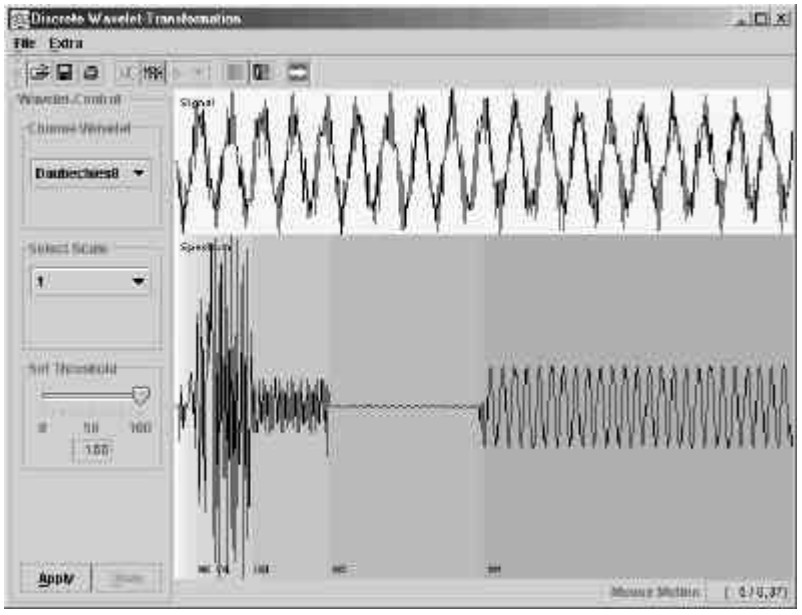


Abbildung 9.14: Diskrete Wavelet-Transformation

Möchte man nun das Signal entrauschen, so kann man einen Schwellwert (Threshold) einstellen und die Wavelet-Koeffizienten der gewählten Skala bis zum gewünschten Schwellwert eliminieren. In Abbildung 9.15 wurde für die Skala 1 der Schwellwert 100 gewählt. dadurch werden alle Wavelet-Koeffizienten dieser Skala auf Null gesetzt. Das Programm rekonstruiert aus den verbleibenden Koeffizienten das manipuliert Signal und stellt es im oberen Feld dar. Wie man erkennen kann, wurde das Rauschen bereits durch Entfernung der Wavelet-Koeffizienten der Skala 1 aus dem Signal entfernt.

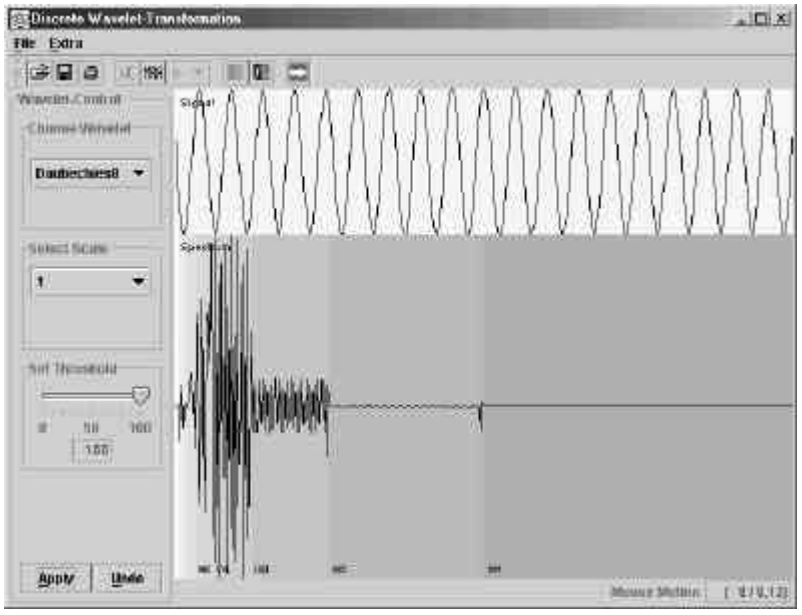


Abbildung 9.15: Entrauschtes Signal

Das gleiche Ergebnis hätte man aber auch durch eine gewöhnliche Hochpassfilterung erreichen können. Von großem Vorteil sind ist die Wavelet-Transformation dann, wenn die Signalinformation und das Rauschen im gleichen Frequenzbereich liegen. Das wird durch die Abbildungen 9.16 und 9.17 verdeutlicht. Das Rauschen und die zum Signal gehörenden

Impulse liegen im gleichen Frequenzbereich. Der Schwellwert muss so gewählt werden, dass nur das Rauschen eliminiert wird und nicht die zum Signal gehörenden Impulse. Abbildung 9.17 zeigt im oberen Feld das rekonstruierte, entrauchte Signal. Wie man erkennen kann, wurden durch den Schwellwert von 80 nicht alle Wavelet-Koeffizienten der Skala 1 auf Null gesetzt, wodurch die beiden Impulse auch nach der Rekonstruktion erhalten bleiben.

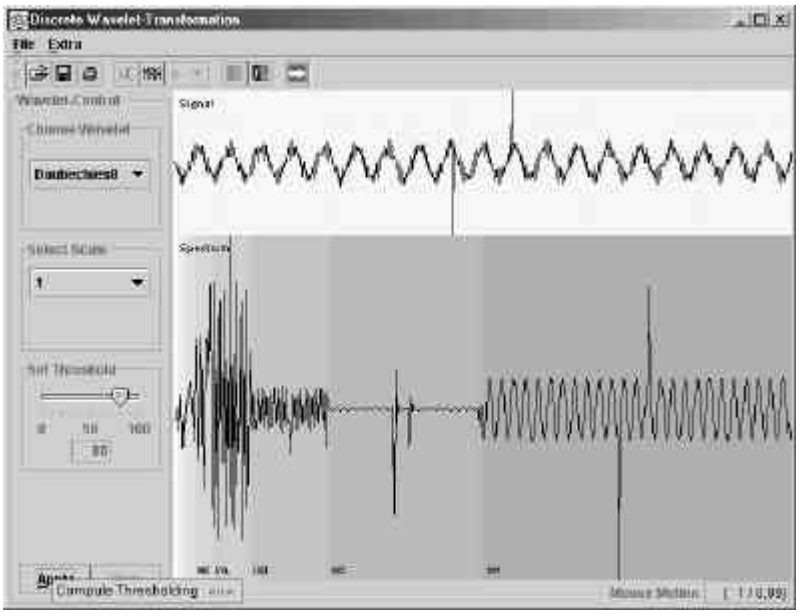


Abbildung 9.16: Verrauschtes Signal

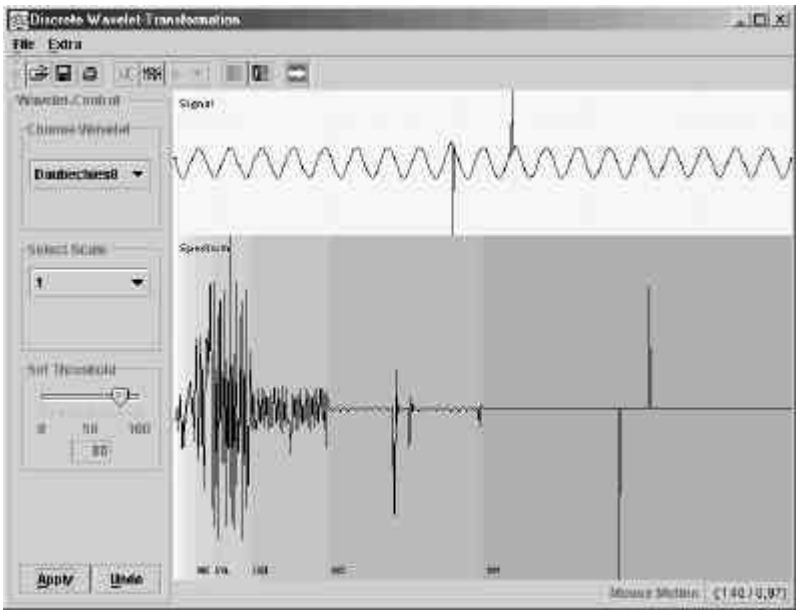


Abbildung 9.17: Entrauschtes Signal

Das Programm ermöglicht die Veranschaulichung einer Multi-Skalen-Analyse, wie sie Kapitel 5.2 beschrieben wurde (Gleichung (5.2.12)). Dabei können Detailräume bis zu einer Zerlegungstiefe von 5 dargestellt werden. Abbildung 9.18 zeigt die Zerlegung eines Signals in 4 Detailräume. Im Raum V4 ist die Approximation des Signals dargestellt, die entsteht, wenn das Signal ohne die Wavelet-Koeffizienten der ersten vier Skalen rekonstruiert wird. Analog zur Schreibweise in Gleichung (5.2.9) ergibt sich für die dargestellte Zerlegung:

$$V_0 = W_1 \oplus W_2 \oplus W_3 \oplus W_4 \oplus V_4 .$$

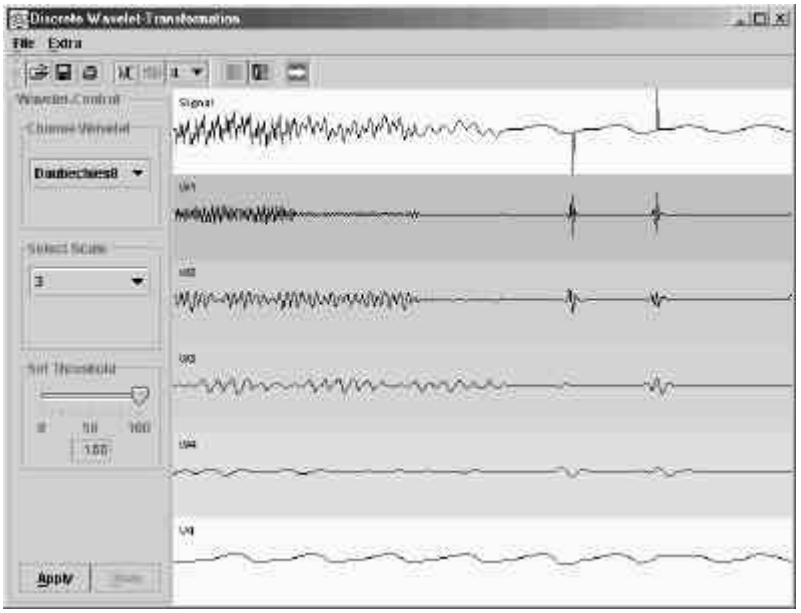


Abbildung 9.18: Multi-Skalen-Analyse

Mit Hilfe des Programms können auch die Approximationseigenschaften der jeweiligen Wavelets demonstriert werden. Dies wird in den Abbildungen 9.19 bis 9.22 anhand der vier verschiedenen Wavelets gezeigt. Dabei ist zu beachten, dass das rekonstruierte Signal im Approximationsraum V3 lediglich aus den Skalen die größer als 3 sind rekonstruiert wurde. Demnach wurden bei der Rekonstruktion anstatt aller 512 Wavelet-Koeffizienten nur 64 Wavelet-Koeffizienten verwendet. Dies entspricht einer Komprimierung im Verhältnis von 8:1. Die Approximationseigenschaften des Daubechies 8-Wavelets sind für das gewählte Signal sehr gut. Es ist gut zu erkennen, dass die jeweiligen Approximationen deutlich den Stempel der verwendeten Wavelet-Funktionen tragen.

Abbildung 9.19: Approximation mit dem Haar-Wavelet

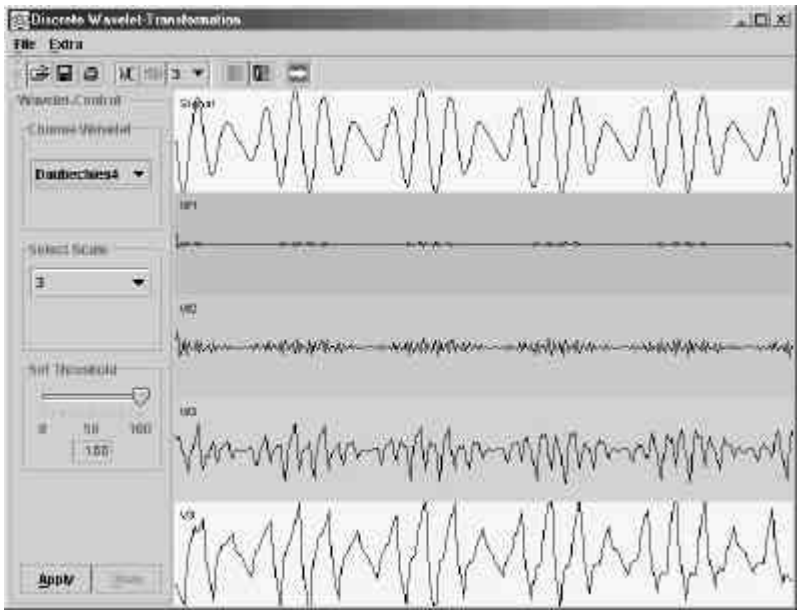


Abbildung 9.20: Approximation mit dem Daubechies 4-Wavelet

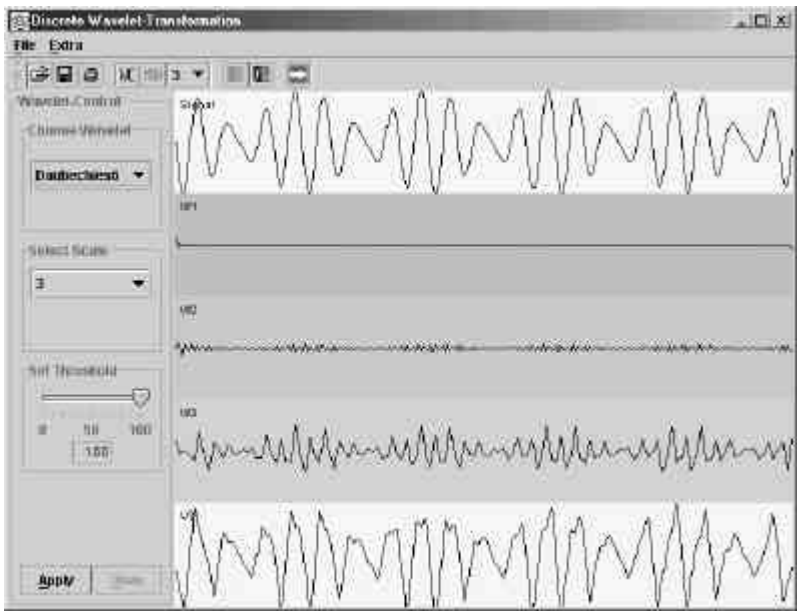


Abbildung 9.21: Approximation mit dem Daubechies 6-Wavelet

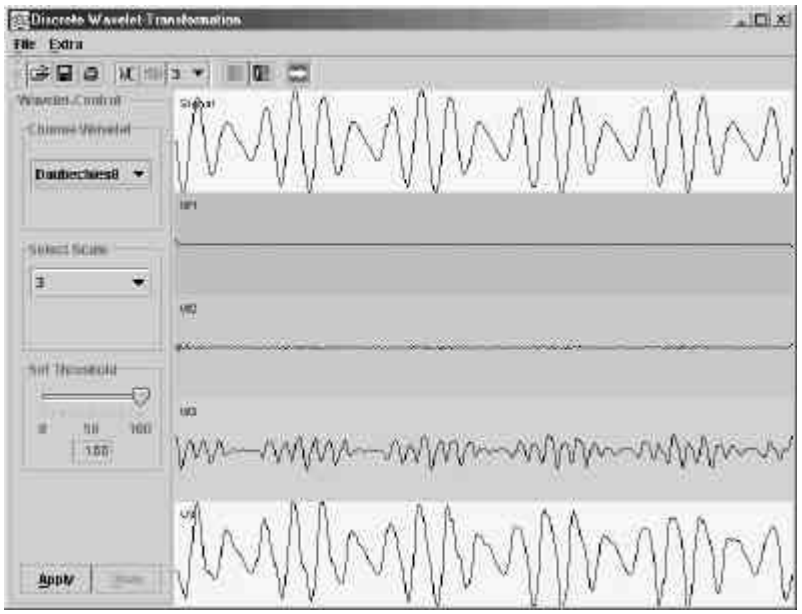


Abbildung 9.22: Approximation mit dem Daubechies 8-Wavelet

9.3 Die Java-Applets

Um die Möglichkeiten der beiden Programme auch im Internet nutzen zu können, wurden die zwei Applications zusätzlich als Applets programmiert. die Möglichkeiten der Applets sind nicht ganz so umfangreich weid die der Java-Applications. Um die Applets auf so vielen Browsern wie möglich laufen lassen zu können, wurden die grafischen Benutzeroberflächen mit den GUI-Komponenten des AWT erstellt. Da das AWT wesentlich weniger Gestaltungsmöglichkeiten bietet als die Swing-Klassen, ist die grafischen Benutzeroberfläche nicht ganz so ansprechend. Hier kommen die Möglichkeiten der Swing-Klassen erst richtig zur Geltung. Die Ausführung von Applets erlaubt es nicht, Daten in das Applet zu laden. Daher werden in den Applets einige Signale generiert, an denen man die Wirkungsweise der Wavelet-Transformation testen kann.

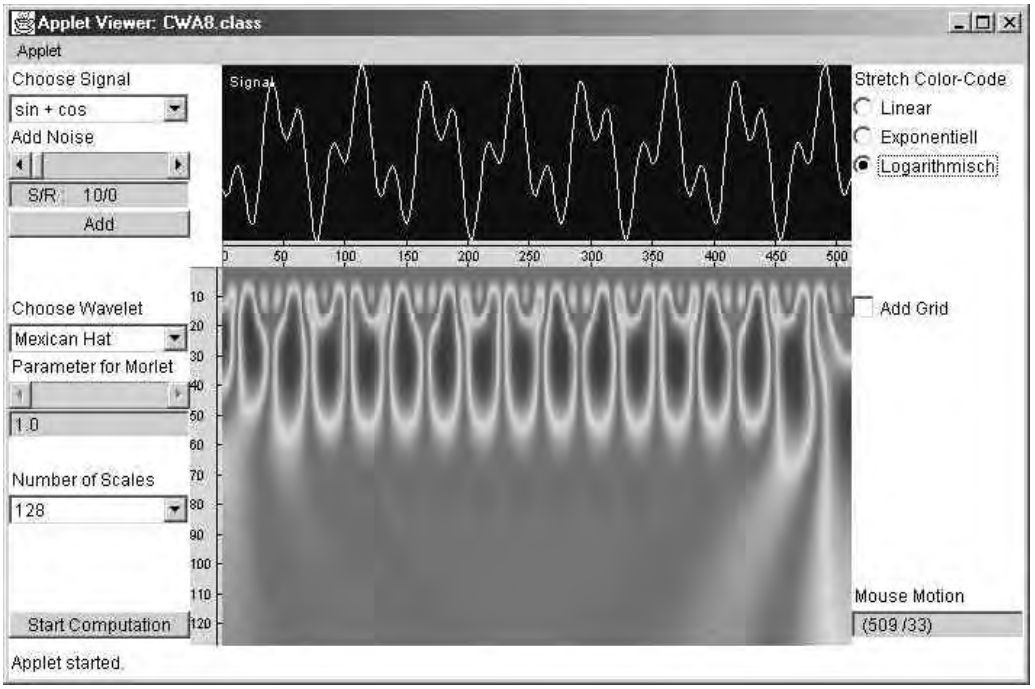


Abbildung 9.23: Das Applet der kontinuierlichen Wavelet-Transformation

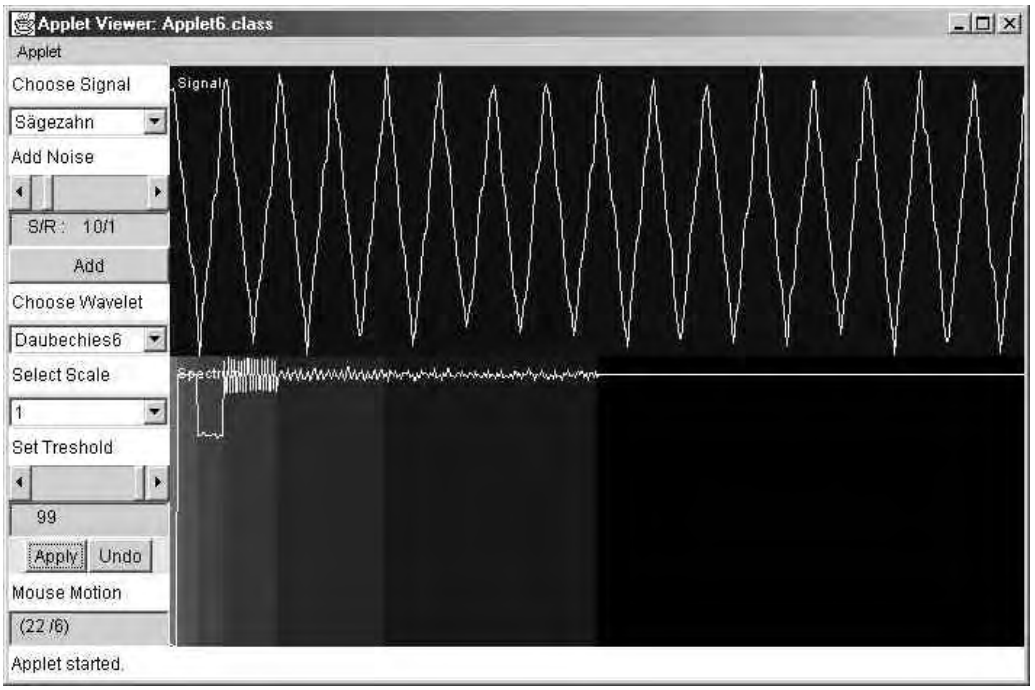


Abbildung 9.24: Das Applet der diskreten Wavelet-Transformation

9.4 Fazit

Die Programme ermöglichen die Analyse von eindimensionalen Signalen. Mit dem Programm der diskreten Wavelet-Transformation ist durch Thresholding auch die Manipulation des Signals möglich. In den Programmen ist nur einen kleiner Teil der bekannten Wavelets

implementiert. Sie können durch weitere Wavelets erweitert werden, wie zum Beispiel durch die Daubechies-Wavelets höherer Ordnung.
Im Rahmen dieser Diplomarbeit wurden nur die eindimensionale Wavelet-Transformation behandelt. Wenn man beispielsweise Bilder mittels Wavelet-Transformation analysieren oder komprimieren will, werden zweidimensionale Wavelets benötigt. Diese stellen die logische Erweiterung der in dieser Arbeit vorgestellten eindimensionalen Wavelets dar.

Anhang A

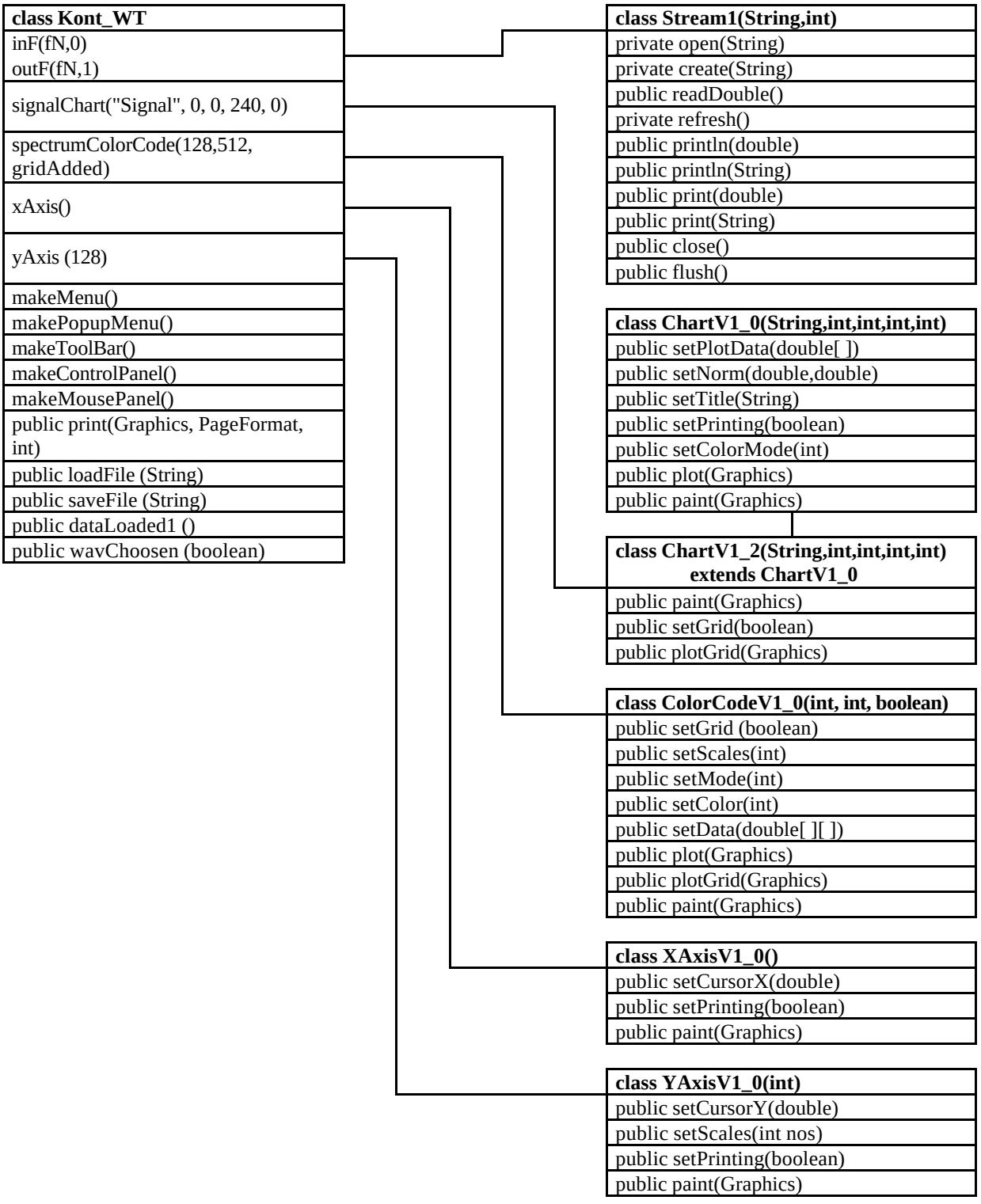
Die Java-Application zur kontinuierlichen Wavelet-Transformation

Die grafische Benutzeroberfläche wurde mit Hilfe des GridBagLayout erstellt. Die folgende Abbildung soll die Anordnung der Elemente veranschaulichen.

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td>0</td><td colspan="3">toolBar</td></tr>
<tr><td>1</td><td rowspan="4">controlPanel</td><td rowspan="2"></td><td>signalChart</td></tr>
<tr><td>2</td><td>xAxis</td></tr>
<tr><td>3</td><td rowspan="2">yAxis</td><td>spectrumColorCode</td></tr>
<tr><td>4</td><td></td><td>mousePanel</td></tr>
</table>

Abbildung A.1: Das GridBagLayout

Folgende Grafik soll die verschiedenen Klassen in kurzer Form beschreiben. Die Hauptklasse Kont_WT erzeugt weitere Objekt aus anderen Klassen. Hier sollen nur die Klassen, die die Grafiken erzeugen sowie die Klasse mit der die Daten ein- bzw. ausgelesen werden können, dargestellt werden.


class Kont_WT
inF(fN,0)
outF(fN,1)
signalChart("Signal", 0, 0, 240, 0)
spectrumColorCode(128,512, gridAdded)
xAxis()
yAxis (128)
makeMenu()
makePopupMenu()
makeToolBar()
makeControlPanel()
makeMousePanel()
public print(Graphics, PageFormat, int)
public loadFile (String)
public saveFile (String)
public dataLoaded1 ()
public wavChoosen (boolean)
class Stream1(String,int)
private open(String)
private create(String)
public readDouble()
private refresh()
public println(double)
public println(String)
public print(double)
public print(String)
public close()
public flush()
class ChartV1_0(String,int,int,int,int)
public setPlotData(double[ ])
public setNorm(double,double)
public setTitle(String)
public setPrinting(boolean)
public setColorMode(int)
public plot(Graphics)
public paint(Graphics)
class ChartV1_2(String,int,int,int,int) extends ChartV1_0
public paint(Graphics)
public setGrid(boolean)
public plotGrid(Graphics)
class ColorCodeV1_0(int, int, boolean)
public setGrid (boolean)
public setScales(int)
public setMode(int)
public setColor(int)
public setData(double[ ][ ])
public plot(Graphics)
public plotGrid(Graphics)
public paint(Graphics)
class XAxisV1_0()
public setCursorX(double)
public setPrinting(boolean)
public paint(Graphics)
class YAxisV1_0(int)
public setCursorY(double)
public setScales(int nos)
public setPrinting(boolean)
public paint(Graphics)

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.io.File;
import java.awt.*;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.*;
import java.awt.color.*;
import javax.swing.*;
import javax.swing.AbstractButton;
import javax.swing.JComponent;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.ImageIcon;
import javax.swing.filechooser.*;
import javax.swing.text.*;
import javax.swing.event.*;
import java.awt.print.*;
import java.awt.print.PrinterJob;
import java.awt.geom.*;


public class Kont_WT    extends     JFrame
                        implements  ActionListener, ItemListener,
                                    ChangeListener, Printable,
                                    MouseMotionListener {

  Stream1   inF, outF;
  String    inFileName, outFileName;

  boolean gridAdded=false;
  boolean xAxisAdded=false;
  boolean yAxisAdded=false;
  String wav;                 // Wavelettyp
  int nos=128;                // Anzahl der Skalen
  double omega0=1.0;          // Parameter fuer Morlet
  int colorCodePicked=1;      // lin, log, exp
  ChartV1_2 signalChart = new ChartV1_2("Signal", 0, 0, 240, 0);
  ColorCodeV1_0 spectrumColorCode = new ColorCodeV1_0(128,512, gridAdded);
  XAxisV1_0 xAxis = new XAxisV1_0();
  YAxisV1_0 yAxis = new YAxisV1_0(128);
  double[] signal = new double[512];
  double[][] cwtspec = new double[128][512];

  // GUI-Variablen**************
  JMenuBar  menu;
  JMenu           file, extra, colorCode, stretchColorCode;
  JMenuItem load, save, print, exit, sin_cos, menuItem;
  JPopupMenu      colorCodePopup;
  JRadioButtonMenuItem  ccLinear, ccExp, ccLog, ccLinear1, ccExp1, ccLog1,
                  SW, RGB, SW1, RGB1;
  JCheckBoxMenuItem     addXAxis, addYAxis, addGrid;
  JToolBar  toolBar;
  JButton   tbLoad, tbSave, tbPrint, tbExit, tbLinear, tbLog, tbExp,
            tbSW, tbRGB;
  JToggleButton   tbGrid, tbXAchse, tbYAchse;
  JPanel    toolBarPanel, controlPanel, mousePanel;
  JPanel    waveletChoicePanel, morletPanel, numberOfScalePanel,
buttonPanel;
```

```
JComboBox waveletChoice,numberOfScale;
JSlider   morletSlider;
JTextField      morletTextField,mouseTextField;
JButton   run;
JLabel    mouseLabel;
JFileChooser    inFile, outFile;

public Kont_WT() {
  inFile = new JFileChooser();
  outFile = new JFileChooser();
  makeMenu();
  makePopupMenu();
  makeToolBar();
  makeControlPanel();
  makeMousePanel();

  // Einrichten des GridBagLayouts
  Container contentPane = getContentPane();
  GridBagLayout gridbag = new GridBagLayout();
  GridBagConstraints c = new GridBagConstraints();
  contentPane.setLayout(gridbag);
  c.fill = GridBagConstraints.BOTH;

  // ToolBar**********
  c.gridx = 0;
  c.gridy = 0;
  c.gridwidth = 3;
  c.gridheight = 1;
  c.weightx = 0.0;
  c.weighty = 0.0;
  gridbag.setConstraints(toolBar, c);
  contentPane.add(toolBar);
  // ControlPanel**********
  c.gridx = 0;
  c.gridy = 1;
  c.gridwidth = 1;
  c.gridheight = 4;
  gridbag.setConstraints(controlPanel, c);
  contentPane.add(controlPanel);
  // Signal**********
  c.gridx = 2;
  c.gridy = 1;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 1.0;
  c.weighty = 1.0;
  c.ipadx = 500;
  c.ipady = 80;
  gridbag.setConstraints(signalChart, c);
  contentPane.add(signalChart);
  // Spectrum**********
  c.gridx = 2;
  c.gridy = 3;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 1.0;
  c.weighty = 1.0;
  c.ipadx = 500;
  c.ipady = 260;
  spectrumColorCode.addMouseMotionListener(this);
  gridbag.setConstraints(spectrumColorCode, c);
  contentPane.add(spectrumColorCode);
  // X-Achse**********
```

```
  c.gridx = 2;
  c.gridy = 2;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 0.0;
  c.weighty = 0.0;
  c.ipadx = 0;
  c.ipady = 8;
  gridbag.setConstraints(xAxis, c);
  contentPane.add(xAxis);
  xAxis.setVisible(xAxisAdded);
  // Y-Achse**********
  c.gridx = 1;
  c.gridy = 3;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 0.0;
  c.weighty = 0.0;
  c.ipadx = 12;
  c.ipady = 0;
  gridbag.setConstraints(yAxis, c);
  contentPane.add(yAxis);
  yAxis.setVisible(false);
  // MousePanel**********
  c.gridx = 2;
  c.gridy = 4;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 0.0;
  c.weighty = 0.0;
  c.ipadx = 0;
  c.ipady = 0;
  gridbag.setConstraints(mousePanel, c);
  contentPane.add(mousePanel);
  // Fenster schließen durch [x] ******************
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      dispose();
      System.exit(0);
    }
  });
  setTitle("Continous Wavelet-Transformation");
  pack();
  setVisible(true);
}  // Ende Konstruktor Kont_WT

// Behandler**********************************
public void actionPerformed (ActionEvent e) {
  // Datei laden und Signal darstellen
  if (e.getSource() == load || e.getSource() == tbLoad) {
    inFile.showOpenDialog(Kont_WT.this);
    File file = inFile.getSelectedFile();
    inFileName = file.getAbsolutePath();
    try {
      loadFile(inFileName);
    }
    catch (IOException e1) {
      System.out.println(e1);
    }
    dataLoaded1();
    setTitle("Continous Wavelet-Transformation - "+ file.getName());
  }
  // Wavelet-Koeffizienten als ASCII-Datei abspeichern
```

```
else if (e.getSource() == save || e.getSource() == tbSave) {
  int returnVal = outFile.showSaveDialog(Kont_WT.this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
    File file = outFile.getSelectedFile();
    outFileName = file.getAbsolutePath();
    try {
    saveFile(outFileName);
  }
  catch (IOException e1) {
    System.out.println(e1);
  }
  }
}
// Grafiken ausdrucken
else if (e.getSource() == print || e.getSource() == tbPrint) {
  PrinterJob printJob = PrinterJob.getPrinterJob();
  printJob.setPrintable(this);
  if (printJob.printDialog()) {
    try {
      printJob.print();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
// Programm beenden
else if (e.getSource() == exit || e.getSource() == tbExit) {
  dispose();
  System.exit(0);
}
// Demo-Signal erstellen und darstellen
else if (e.getSource() == sin_cos) {
  double sin[] = new double[512];
  for (int l=0; l<100; l++){
    sin[l]= 10*(Math.sin((double)l/0.5));
  }
  for (int l=0; l<200; l++){
    sin[l]= sin[l]+8*(Math.sin((double)l/1));
  }
  for (int l=0; l<270; l++){
    sin[l]= sin[l]+5*(Math.sin((double)l/2));
  }
  for (int l=0; l<420; l++){
    sin[l]= sin[l]+2*(Math.sin((double)l/5));
    if(l==330) sin[l]=50;
    if(l==400) sin[l]=-50;
  }
  for (int l=0; l<512; l++){
    sin[l]= sin[l]+0;
  }
  signal = sin;
  inFileName = "DemoData";
  dataLoaded1();
}
// Color-Code: linear
else if (e.getSource() == ccLinear || e.getSource() == ccLinear1
                                   || e.getSource() == tbLinear) {
  ccLinear.setSelected(true);
  ccLinear1.setSelected(true);
  colorCodePicked = 1;
  tbLinear.setEnabled(false);
  tbLog.setEnabled(true);
```

```
      tbExp.setEnabled(true);
      spectrumColorCode.setMode(colorCodePicked);
      spectrumColorCode.setData(cwtspec);
      spectrumColorCode.paint(spectrumColorCode.getGraphics());
    }
    // Color-Code: logarithmic
    else if (e.getSource() == ccLog || e.getSource() == ccLog1 ||
e.getSource() == tbLog) {
      ccLog.setSelected(true);
      ccLog1.setSelected(true);
      colorCodePicked = 3;
      tbLinear.setEnabled(true);
      tbLog.setEnabled(false);
      tbExp.setEnabled(true);
      spectrumColorCode.setMode(colorCodePicked);
      spectrumColorCode.setData(cwtspec);
      spectrumColorCode.paint(spectrumColorCode.getGraphics());
    }
    // Color-Code: exponential
    else if (e.getSource() == ccExp || e.getSource() == ccExp1 ||
e.getSource() == tbExp) {
      ccExp.setSelected(true);
      ccExp1.setSelected(true);
      colorCodePicked = 2;
      tbLinear.setEnabled(true);
      tbLog.setEnabled(true);
      tbExp.setEnabled(false);
      spectrumColorCode.setMode(colorCodePicked);
      spectrumColorCode.setData(cwtspec);
      spectrumColorCode.paint(spectrumColorCode.getGraphics());
    }
    // Grafiken in Schwarz/Weiss
    else if (e.getSource() == tbSW || e.getSource() == SW || e.getSource()
== SW1) {
      SW.setSelected(true);
      SW1.setSelected(true);
      tbSW.setEnabled(false);
      tbRGB.setEnabled(true);
      spectrumColorCode.setColor(0);
      spectrumColorCode.setData(cwtspec);
      spectrumColorCode.paint(spectrumColorCode.getGraphics());
      signalChart.setColorMode(1);
      signalChart.paint(signalChart.getGraphics());
    }
    // Grafiken farbig
    else if (e.getSource() == tbRGB || e.getSource() == RGB ||
e.getSource() == RGB1) {
      RGB.setSelected(true);
      RGB1.setSelected(true);
      tbSW.setEnabled(true);
      tbRGB.setEnabled(false);
      spectrumColorCode.setColor(1);
      spectrumColorCode.setData(cwtspec);
      spectrumColorCode.paint(spectrumColorCode.getGraphics());
      signalChart.setColorMode(0);
      signalChart.paint(signalChart.getGraphics());
    }
    // Berechnung auslösen und Spektrum darstellen
    else if (e.getSource() == run) {
      if(wav.equals("Morlet")) wav="morlet";
      if(wav.equals("Mexican Hat")) wav="mexhat";
      if(wav.equals("Haar")) wav="haar";
      CWT instanceOfCWT=new CWT (512,nos);
```

```
    instanceOfCWT.run(signal,wav,omega0,cwtspec);
    spectrumColorCode.setMode(colorCodePicked);
    spectrumColorCode.setScales(nos);
    spectrumColorCode.setData(cwtspec);
    spectrumColorCode.paint(spectrumColorCode.getGraphics());
    if (yAxisAdded) {
      yAxis.setScales(nos);
      yAxis.paint(yAxis.getGraphics());
    }
    Toolkit.getDefaultToolkit().beep();
    save.setEnabled(true);
    tbSave.setEnabled(true);
    print.setEnabled(true);
    tbPrint.setEnabled(true);
  }
}

public void itemStateChanged (ItemEvent e) {
  int index=10;
  Object picked = e.getSource();
  if (picked == waveletChoice) {
    JComboBox cb = (JComboBox)e.getSource();
    wav = (String)cb.getSelectedItem();
    if (wav == "Morlet") morletSlider.setEnabled(true);
    else if (wav != "Morlet") morletSlider.setEnabled(false);
    if (wav == "none") wavChoosen(false);
    else if (wav != "none") wavChoosen(true);
  }
  else if (picked == numberOfScale) {
    JComboBox cb = (JComboBox)e.getSource();
    String noos = (String)cb.getSelectedItem();
    nos = Integer.parseInt(noos);
  }
  else if (picked == addXAxis) {
    xAxisAdded = true;
    xAxis.setVisible(true);
    index=0;
  }
  else if (picked == addYAxis) {
    yAxis.setScales(nos);
    yAxisAdded = true;
    yAxis.setVisible(true);
    index=1;
  }
  else if (picked == addGrid) {
    gridAdded=true;
    signalChart.setGrid(gridAdded);
    signalChart.paint(signalChart.getGraphics());
    spectrumColorCode.setGrid(gridAdded);
    spectrumColorCode.paint(spectrumColorCode.getGraphics());
    index=2;
  }
  else if (picked == tbXAchse) {
    addXAxis.setSelected(true);
    index=3;
  }
  else if (picked == tbYAchse) {
    addYAxis.setSelected(true);
    index=4;
  }
  else if (picked == tbGrid) {
    addGrid.setSelected(true);
    index=5;
```

```
    }
    if (e.getStateChange() == ItemEvent.DESELECTED) {
      if (index == 0) {
      xAxisAdded = false;
      xAxis.setVisible(false);
      }
      else if (index == 1) {
      yAxisAdded = false;
      yAxis.setVisible(false);
      }
      else if (index == 2) {
      gridAdded=false;
      signalChart.setGrid(gridAdded);
      signalChart.paint(signalChart.getGraphics());
        spectrumColorCode.setGrid(gridAdded);
        spectrumColorCode.paint(spectrumColorCode.getGraphics());
      }
      else if (index == 3) addXAxis.setSelected(false);
      else if (index == 4) addYAxis.setSelected(false);
      else if (index == 5) addGrid.setSelected(false);
    }
  }
  public void stateChanged(ChangeEvent e) {
    JSlider source = (JSlider)e.getSource();
      if (!source.getValueIsAdjusting()) {
      omega0 = (double)source.getValue()/10;
        morletTextField.setText(String.valueOf(omega0));
      }
  }
  public void mouseMoved(MouseEvent e) {
    int x = spectrumColorCode.getSize().width;
    int y = spectrumColorCode.getSize().height;
    double xs = (double)e.getX() / (double) x;
    double ys = (double)e.getY() / (double) y;
    mouseTextField.setText(" (" + (int)(512*xs) + " /" + (int)(nos*ys) +
")");
  }
  public void mouseDragged(MouseEvent e) {
    if (xAxisAdded) {
      xAxis.setCursorX((double)e.getX());
      xAxis.paint(xAxis.getGraphics());
    }
    if (yAxisAdded) {
      yAxis.setCursorY((double)e.getY());
      yAxis.paint(yAxis.getGraphics());
    }
    int x = spectrumColorCode.getSize().width;
    int y = spectrumColorCode.getSize().height;
    double xs = (double)e.getX() / (double) x;
    double ys = (double)e.getY() / (double) y;
    mouseTextField.setText(" (" + (int)(512*xs) + " /" + (int)(nos*ys) +
")");
  }

  // Methoden************************
  void makeMenu() {
    menu = new JMenuBar();
      file = new JMenu("File");
      file.setMnemonic(KeyEvent.VK_F);
        load = new JMenuItem("Load...", new ImageIcon("open.gif"));
      load.setMnemonic(KeyEvent.VK_L);
      load.setAccelerator(KeyStroke.getKeyStroke(
            KeyEvent.VK_L, ActionEvent.CTRL_MASK));
```

```
    load.addActionListener(this);
  save = new JMenuItem("Save as...", new ImageIcon("save.gif"));
  save.setMnemonic(KeyEvent.VK_S);
  save.setAccelerator(KeyStroke.getKeyStroke(
        KeyEvent.VK_S, ActionEvent.ALT_MASK));
  save.setEnabled(false);
    save.addActionListener(this);
  exit = new JMenuItem("Exit", new ImageIcon("exit.gif"));
  exit.addActionListener(this);
  print = new JMenuItem("Print..", new ImageIcon("print1.gif"));
  print.addActionListener(this);
  print.setEnabled(false);
  file.add(load);
  file.add(save);
  file.addSeparator();
  file.add(print);
  file.addSeparator();
  file.add(exit);
menu.add(file);
  extra = new JMenu("Extra");
  extra.setMnemonic(KeyEvent.VK_E);
  addXAxis = new JCheckBoxMenuItem("Add X-Axis");
  addXAxis.addItemListener(this);
  addYAxis = new JCheckBoxMenuItem("Add Y-Axis");
  addYAxis.addItemListener(this);
  addGrid = new JCheckBoxMenuItem("Add Grid");
  addGrid.addItemListener(this);
  stretchColorCode = new JMenu("Stretch Color-Code");
    ButtonGroup group = new ButtonGroup();
      ccLog = new JRadioButtonMenuItem("Logarithmic");
      ccLog.addActionListener(this);
    group.add(ccLog);
      ccLinear = new JRadioButtonMenuItem("Linear");
      ccLinear.setSelected(true);
      ccLinear.addActionListener(this);
    group.add(ccLinear);
      ccExp = new JRadioButtonMenuItem("Exponential");
      ccExp.addActionListener(this);
    group.add(ccExp);
  stretchColorCode.add(ccLog);
  stretchColorCode.add(ccLinear);
  stretchColorCode.add(ccExp);
  colorCode = new JMenu("ColorCode");
    ButtonGroup group0 = new ButtonGroup();
      SW = new JRadioButtonMenuItem("BW");
      SW.addActionListener(this);
    group0.add(SW);
      RGB = new JRadioButtonMenuItem("RGB");
      RGB.setSelected(true);
      RGB.addActionListener(this);
    group0.add(RGB);
  colorCode.add(SW);
  colorCode.add(RGB);
  sin_cos = new JMenuItem("Demo-Data");
  sin_cos.addActionListener(this);
  extra.add(addXAxis);
  extra.add(addYAxis);
  extra.add(addGrid);
  extra.addSeparator();
  extra.add(stretchColorCode);
  extra.add(colorCode);
  extra.addSeparator();
  extra.add(sin_cos);
```

```
  menu.add(extra);
  setJMenuBar(menu);
}
void makePopupMenu() {
  colorCodePopup = new JPopupMenu();
    ButtonGroup group1 = new ButtonGroup();
    ccLog1 = new JRadioButtonMenuItem("Logarithmic");
    ccLog1.addActionListener(this);
    group1.add(ccLog1);
    ccLinear1 = new JRadioButtonMenuItem("Linear");
    ccLinear1.setSelected(true);
    ccLinear1.addActionListener(this);
    group1.add(ccLinear1);
    ccExp1 = new JRadioButtonMenuItem("Exponential");
    ccExp1.addActionListener(this);
    group1.add(ccExp1);
    ButtonGroup group2 = new ButtonGroup();
    SW1 = new JRadioButtonMenuItem("Black/White");
    SW1.addActionListener(this);
    group2.add(SW1);
    RGB1 = new JRadioButtonMenuItem("RGB");
    RGB1.setSelected(true);
    RGB1.addActionListener(this);
    group2.add(RGB1);
  colorCodePopup.add(ccLog1);
  colorCodePopup.add(ccLinear1);
  colorCodePopup.add(ccExp1);
  colorCodePopup.addSeparator();
  colorCodePopup.add(SW1);
  colorCodePopup.add(RGB1);
  MouseListener popupListener = new PopupListener();
  spectrumColorCode.addMouseListener(popupListener);
}
void makeToolBar() {
  toolBar = new JToolBar();
    tbLoad = new JButton(new ImageIcon("open.gif"));
    tbLoad.addActionListener(this);
    tbLoad.setToolTipText("Open File");
  toolBar.add(tbLoad);
    tbSave = new JButton(new ImageIcon("save.gif"));
    tbSave.addActionListener(this);
    tbSave.setToolTipText("Save File");
    tbSave.setEnabled(false);
  toolBar.add(tbSave);
    tbPrint = new JButton(new ImageIcon("print1.gif"));
    tbPrint.addActionListener(this);
    tbPrint.setToolTipText("Print File");
    tbPrint.setEnabled(false);
  toolBar.add(tbPrint);
  toolBar.addSeparator();
    tbGrid = new JToggleButton(new ImageIcon("grid.gif"));
    tbGrid.setMaximumSize(tbPrint.getMaximumSize());
    tbGrid.addItemListener(this);
    tbGrid.setToolTipText("Add Grid");
  toolBar.add(tbGrid);
    tbXAchse = new JToggleButton(new ImageIcon("xAchse.gif"));
    tbXAchse.setMaximumSize(tbPrint.getMaximumSize());
    tbXAchse.addItemListener(this);
    tbXAchse.setToolTipText("Add X-Axis");
  toolBar.add(tbXAchse);
    tbYAchse = new JToggleButton(new ImageIcon("yAchse.gif"));
    tbYAchse.setMaximumSize(tbPrint.getMaximumSize());
    tbYAchse.addItemListener(this);
```

```
      tbYAchse.setToolTipText("Add Y-Axis");
    toolBar.add(tbYAchse);
    toolBar.addSeparator();
      tbLog = new JButton(new ImageIcon("log.gif"));
      tbLog.addActionListener(this);
      tbLog.setToolTipText("Stretch Color-Code: Logarithmic");
    toolBar.add(tbLog);
      tbLinear = new JButton(new ImageIcon("linear.gif"));
      tbLinear.setEnabled(false);
      tbLinear.addActionListener(this);
      tbLinear.setToolTipText("Stretch Color-Code: Linear");
    toolBar.add(tbLinear);
      tbExp = new JButton(new ImageIcon("Exp.gif"));
      tbExp.addActionListener(this);
      tbExp.setToolTipText("Stretch Color-Code: Exponential");
    toolBar.add(tbExp);
    toolBar.addSeparator();
      tbSW = new JButton(new ImageIcon("SW.gif"));
      tbSW.addActionListener(this);
      tbSW.setToolTipText("Color Code: SW");
    toolBar.add(tbSW);
      tbRGB = new JButton(new ImageIcon("RGB.gif"));
      tbRGB.setEnabled(false);
      tbRGB.addActionListener(this);
      tbRGB.setToolTipText("Color Code: RGB");
    toolBar.add(tbRGB);
    toolBar.addSeparator();
      tbExit = new JButton(new ImageIcon("exit.gif"));
      tbExit.addActionListener(this);
      tbExit.setToolTipText("Exit");
    toolBar.add(tbExit);
  }
  void makeControlPanel() {
    controlPanel = new JPanel();
    controlPanel.setLayout(new GridLayout(4,1,5,5));
    controlPanel.setBorder(BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("Wavlet-Control"),
                    BorderFactory.createEmptyBorder(5,5,5,5)));
    controlPanel.setPreferredSize(new Dimension(160,450));
      waveletChoicePanel = new JPanel();
      waveletChoicePanel.setLayout(new BoxLayout(waveletChoicePanel,
BoxLayout.Y_AXIS));
      waveletChoicePanel.setBorder(BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("Choose Wavelet"),
                    BorderFactory.createEmptyBorder(5,5,5,5)));
      waveletChoicePanel.setAlignmentY(CENTER_ALIGNMENT);
      waveletChoice = new JComboBox();
      waveletChoice.setPreferredSize(new Dimension(150,30));
        waveletChoice.addItem("none");
        waveletChoice.addItem("Haar");
        waveletChoice.addItem("Mexican Hat");
        waveletChoice.addItem("Morlet");
      waveletChoice.setEnabled(false);
      waveletChoice.addItemListener(this);
      waveletChoicePanel.add(waveletChoice);
    controlPanel.add(waveletChoicePanel);
      morletPanel = new JPanel();
      morletPanel.setLayout(new BoxLayout(morletPanel, BoxLayout.Y_AXIS));
      morletPanel.setBorder(BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("Parameter for
Morlet"),
                    BorderFactory.createEmptyBorder(5,5,5,5)));
      morletPanel.setAlignmentY(CENTER_ALIGNMENT);
```

```
        morletSlider = new JSlider(JSlider.HORIZONTAL,10,100,10);
      morletSlider.setPreferredSize(new Dimension (70,45));
      morletSlider.setMajorTickSpacing(10);
      morletSlider.setPaintTicks(true);
        Hashtable labelTable = new Hashtable();
        labelTable.put(new Integer(1), new JLabel("1.0"));
        labelTable.put(new Integer(50), new JLabel("5.0"));
        labelTable.put(new Integer(100), new JLabel("10.0"));
      morletSlider.setLabelTable(labelTable);
      morletSlider.setPaintLabels(true);
      morletSlider.setEnabled(false);
      morletSlider.addChangeListener(this);
      morletTextField = new JTextField ("1.0",1);
      morletTextField.setHorizontalAlignment(JTextField.RIGHT);
      morletTextField.setMaximumSize(new Dimension(35,25));
      morletTextField.setEnabled(true);
      morletTextField.setEditable(false);
      morletPanel.add(morletSlider);
      morletPanel.add(morletTextField);
    controlPanel.add(morletPanel);
      numberOfScalePanel = new JPanel();
      numberOfScalePanel.setLayout(new BoxLayout(numberOfScalePanel,
BoxLayout.Y_AXIS));
      numberOfScalePanel.setBorder(BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("Number of scales"),
                    BorderFactory.createEmptyBorder(5,5,5,5)));
      numberOfScalePanel.setAlignmentY(CENTER_ALIGNMENT);
      numberOfScale = new JComboBox();
      numberOfScale.setPreferredSize(new Dimension(150,30));
      numberOfScale = new JComboBox();
      numberOfScale.setPreferredSize(new Dimension(150,30));
        for (int i=7; i>=0; i--) {
          int j = (int) Math.pow(2, i);
          numberOfScale.addItem(String.valueOf(j));
        }
      numberOfScale.setEnabled(false);
      numberOfScale.addItemListener(this);
      numberOfScalePanel.add(numberOfScale);
    controlPanel.add(numberOfScalePanel);
      buttonPanel = new JPanel();
      buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
      run = new JButton("Start Computation");
      run.setAlignmentY(JComponent.BOTTOM_ALIGNMENT);
      run.setMnemonic(KeyEvent.VK_C);
      run.setEnabled(false);
      run.addActionListener(this);
      run.setToolTipText("Ausgangssignal zeichnen");
      buttonPanel.add(run);
    controlPanel.add(buttonPanel);
  } // Ende MakeControlPanel
  void makeMousePanel() {
    mousePanel = new JPanel();
    mousePanel.setLayout (new FlowLayout(FlowLayout.RIGHT,1,1));
      mouseLabel = new JLabel ("Mouse Motion ");
      mouseTextField = new JTextField (" ",7);
      mouseTextField.setHorizontalAlignment(JTextField.RIGHT);
      mouseTextField.setEditable(false);
    mousePanel.add(mouseLabel);
    mousePanel.add(mouseTextField);
  }
  // Grafiken ausdrucken
  public int print(Graphics g, PageFormat pf, int pi) throws
PrinterException {
```

```
    if (pi >= 1) {
      return Printable.NO_SUCH_PAGE;
    }
    Graphics2D g2 = (Graphics2D) g;
    g2.translate(100,80);
      signalChart.setPrinting(true);
      signalChart.paint(g2);
      signalChart.setPrinting(false);
    g2.translate(0,70);
    if (xAxisAdded){
      xAxis.setPrinting(true);
      xAxis.paint(g2);
      xAxis.setPrinting(false);
    g2.translate(0,20);
    }
      spectrumColorCode.setPrinting(true);
      spectrumColorCode.paint(g2);
      spectrumColorCode.setPrinting(false);
    if (yAxisAdded){
    g2.translate(-20,0);
      yAxis.setPrinting(true);
      yAxis.paint(g2);
      yAxis.setPrinting(false);
    }
    g2.translate(0,280);
      g2.setColor(Color.black);
      g2.setFont(new Font ("Helvetica", Font.PLAIN, 10));
      g2.drawString("Wavelettype: "+wav,0,0);
      if (wav=="morlet") g2.drawString("Parameter for Morlet:
"+omega0,150,0);
      g2.drawString("Number of scales: "+nos,0,20);
      g2.drawString("Inputfile: "+inFileName,0,40);
    return Printable.PAGE_EXISTS;
  }
  public void loadFile (String fN) throws  IOException {
    inF = new Stream1 (fN, 0);
    // Daten Einlesen
    try {
      for (int i=0; i<512; i++) {
        signal[i] = inF.readDouble();
      }
    }
    catch (EOFException e) {}
  }
  public void saveFile (String fN) throws  IOException {
    outF = new Stream1 (fN, 1);
    // Daten Einlesen
    try {
      for (int j=0; j<nos; j++) {
        for (int i=0; i<512; i++) {
          outF.print(cwtspec[j][i]);
          outF.print("\t");
        }
      outF.println("");
      }
      outF.close();
    }
    catch (EOFException e) {}
  }
  // Methode die nach dem Laden des Signals ausgeführt werden
  public void dataLoaded1 () {
    signalChart.setPlotData(signal);
    signalChart.paint(signalChart.getGraphics());
```

```
    waveletChoice.setEnabled(true);
  }
  // Methode die nach der Wahl des Wavelets ausgeführt werden
  public void wavChoosen (boolean b) {
      run.setEnabled(b);
      numberOfScale.setEnabled(b);
  }

  public static void main (String args[]) throws IOException {
    new Kont_WT();
  }


/************************************************
*                                      *
* Klasse PopupListener                       *
*                                      *
************************************************/
    class PopupListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            maybeShowPopup(e);
        }
        public void mouseReleased(MouseEvent e) {
            maybeShowPopup(e);
        }
        private void maybeShowPopup(MouseEvent e) {
            if (e.isPopupTrigger()) {
                colorCodePopup.show(e.getComponent(),
                           e.getX(), e.getY());
            }
        }
    }
} // Ende der Klasse A1

/**********************************************************************
*
*    Class Stream1
*     Konstruktorvariablen
*     filename:   Name des Dateinamens
*     how:        0: lesen
*                 1: schreiben
*
**********************************************************************/
class Stream1 {

  private BufferedReader in;
  private PrintWriter out;
  private StringTokenizer ST;
  private String S;

  public static final int     READ = 0,
                        WRITE = 1;

  public Stream1 (String filename, int how) throws FileNotFoundException,
IOException {
      switch(how) {
        case READ: in = open(filename); break;
        case WRITE: out = create(filename); break;
      }
  }
  private BufferedReader open(String filename) throws FileNotFoundException
{
      return new BufferedReader(new FileReader(filename));
```

```
  }
  private PrintWriter create(String filename) throws IOException {
      return new PrintWriter(new FileWriter(filename));
  }
  // Methode zum lesen aus dem File
  public double readDouble () throws IOException {
    if (ST==null) refresh();
    while (true) {
      try {
        String item = ST.nextToken();
        return Double.valueOf(item.trim()).doubleValue();
      }
      catch (NoSuchElementException e1) {
        refresh ();
      }
      catch (NumberFormatException e2) {
        System.out.println("Error in number, try again.");
      }
    }
  }
  private void refresh () throws IOException {
    S = in.readLine ();
    if (S==null) throw new EOFException();
    ST = new StringTokenizer (S);
  }
  // Methode zuum schreiben in das File out
  public void println(double s) {
      out.println(s);
      out.flush();
  }
  public void println(String s) {
      out.println(s);
      out.flush();
  }
  public void print(double s) {
      out.print(s);
      out.flush();
  }
  public void print(String s) {
      out.print(s);
      out.flush();
  }
  //  Methode zum schließen des Files out
  public void close() throws IOException {
      if (out != null)
            out.close();
      if (in != null)
            in.close();
  }
  public void flush() {
      out.flush();
  }
} // Stream

/*************************************************************************
*
*
*     Class XAxisV1_0
*     keine Konstruktorvariablen
*     xc: Cursor-Position
*
*************************************************************************/
```

```
class XAxisV1_0 extends JPanel {
  double xc;
  boolean isPrinting=false;
  public XAxisV1_0() {
  } //ends constructor of XAxis
  public void setCursorX(double xc1) {
    xc=xc1;
  }
  public void setPrinting(boolean p) {
    isPrinting=p;
  }
  public void paint(Graphics g2) {
    int x=getSize().width;
    int y=getSize().height;
    if (isPrinting){
      x=400;
      y=20;
    }
    g2.setColor(new Color (220,220,220));
    g2.fillRect(0,0,x,y);
    g2.setColor(Color.black);
    g2.drawLine(0, (int)y/6, x, (int)y/6);
    for (int i=0; i<512; i += 50) g2.drawLine(x*i/512, 0, x*i/512,
(int)y/6);
    g2.setFont(new Font("Helvetica",Font.PLAIN,9));
    g2.drawString("0", 0, (int)y*8/10);
    for (int i=50; i<512; i += 50) g2.drawString(String.valueOf(i),
                                    ((int)x*i/512)-6, (int)y*8/10);
    for (int i=0; i<512; i += 50) g2.drawLine(x*i/512, y, x*i/512,
(int)y*7/8);
    g2.setColor(Color.red);
    g2.drawLine((int)xc,0,(int)xc,y);
  }
} // Ende Klasse XAxisV1_0

/*************************************************************************
*
*
*     Class YAxisV1_0
*     numbOfScale: Beschriftung und Skalierung ist abhängig von der
*                  Anzahl der verwendeten Skalen
*     yc: Cursor-Position
*
*************************************************************************/

class YAxisV1_0 extends JPanel {

  int numbOfScale;
  double yc;
  boolean isPrinting=false;

  public YAxisV1_0(int nos) {
    numbOfScale=nos;
  }  // Ende Konstruktor YAxis5

  public void setCursorY(double yc1) {
    yc=yc1;
  }
  public void setScales(int nos) {
    numbOfScale=nos;
  }
  public void setPrinting(boolean p) {
    isPrinting=p;
```

```
  }
  public void paint(Graphics g) {
    int x=getSize().width;
    int y=getSize().height;
    if (isPrinting){
      x=20;
      y=250;
    }
    g.setColor(new Color (220,220,220));
    g.fillRect(0,0,x,y);
    g.setColor(Color.black);
    g.setFont(new Font("Helvetica",Font.PLAIN,9));
    g.drawLine((int)x*6/7, 0, (int)x*6/7, y);
    if (numbOfScale>32) {
      for (int i=10; i<numbOfScale; i+=10) {
      g.drawLine((int)x*6/7, y*i/numbOfScale, x, y*i/numbOfScale);
      g.drawString(String.valueOf(i), 0, y*i/numbOfScale+4);
      }
    }
    else if (numbOfScale<=32 & numbOfScale>16) {
      for (int i=5; i<numbOfScale; i+=5) {
      g.drawLine((int)x*6/7, y*i/numbOfScale, x, y*i/numbOfScale);
      g.drawString(String.valueOf(i), 0, y*i/numbOfScale+4);
      }
    }
    else if (numbOfScale<=16) {
      for (int i=1; i<numbOfScale; i++) {
      g.drawLine((int)x*6/7, y*i/numbOfScale, x, y*i/numbOfScale);
      g.drawString(String.valueOf(i), 0, y*i/numbOfScale+4);
      }
    }
    g.setColor(Color.red);
    g.drawLine(0,(int)yc,x,(int)yc);
  }
} // Ende Klasse YAxisV1_0

/*********************************************************************
*
*    Class ChartV1_0
*     chartname gibt den Namen der Grafik an
*     die Farben red, green, blue duerfen int-Werte von 0-255 annehmen
*     mode:       0: farbig
*           1: Graustufen
*
*********************************************************************/
class ChartV1_0 extends JPanel {

  String    chartname;
  int             red, green, blue, mode;
  double    norm, minimum;
  double[]  polygon = new double[512];
  boolean   isPrinting = false;

  ChartV1_0 (String title, int r, int g, int b, int m) {
    chartname=title;
    red=r;
    green=g;
    blue=b;
    mode=m;
    for(int i=0; i<512; i++) polygon[i]=0.0;
  }  // Ende Konstruktor ChartV1_0

  public void setPlotData(double[] data) {
```

```
  for(int i=0; i<512; i++) polygon[i]=data[i];
}
public void setNorm(double n, double m){
  norm=n;
  minimum=m;
}
public void setTitle(String t) {
  chartname=t;
}
public void setColorMode(int c) {
  mode=c;
}
public void setPrinting(boolean p) {
  isPrinting=p;
}
public void plot(Graphics g) {
  g.setFont(new Font("Helvetica",Font.PLAIN,10));
  if (mode==0) g.setColor(Color.white);
  if (mode==1) g.setColor(Color.black);
  g.drawString(chartname,5,15);
  if (mode==0)g.setColor(Color.yellow);
  if (mode==1)g.setColor(Color.black);
  int xsize=getSize().width;
  int ysize=getSize().height;
  if (isPrinting) {
    xsize=400;
    ysize=70;
  }
  int i;
  double max,min,dy,dx;
  max=-1.0e+14;min=-max;
  for(i=0;i<512;i++) {
     if(polygon[i]>max) {max=polygon[i];}
     if(polygon[i]<min) {min=polygon[i];}
  }
  dx=512.0/xsize;
  dy=ysize/(max-min);
  if (norm!=0.0) {
    dy=ysize/norm;
    min=minimum;
  }
  if((max-min)>1.0e-12) {
    for(i=0;i<xsize-1;i++) {
      g.drawLine(i,(int)((polygon[(int)(i*dx)]-min)*dy),
           i+1,(int)((polygon[(int)((i+1)*dx)]-min)*dy));
    }
  }
  else {
    g.drawLine(0,ysize/2,xsize-1,ysize/2);
  }
}
public void paint(Graphics g) {
  int x=getSize().width;
  int y=getSize().height;
  if (isPrinting) {
    x=400;
    y=70;
  }
  if (mode==0){
    g.setColor(new Color(red,green,blue));
    g.fillRect(0,0,x,y);
  }
  else if (mode==1){
```

```
      int maxInt=0;
      if (red >= maxInt)   maxInt=red;
      if (green >= maxInt) maxInt=green;
      if (blue >= maxInt)  maxInt=blue;
      g.setColor(new Color(maxInt,maxInt,maxInt));
      g.fillRect(0,0,x,y);
    }
    plot(g);
  }

} // Ende Zeichenen*************************************

/*********************************************************************
*
*    Class ChartV1_2
*     erweitert die Klasse ChartV1_0. Durch die Methoden
*           setGrid(boolean)
*           plotGrid(Graphics)
*     hat man  die Moeglichkeit ein Gitter in die
*     Grafik einzufuegen.
*
*********************************************************************/
class ChartV1_2 extends ChartV1_0 {
  boolean gridAdded;

  ChartV1_2 (String title, int r, int g, int b, int m) {
    super(title,r,g,b,m);
  } // Ende Konstruktor ChartV1_2

  public void setGrid(boolean g) {
    gridAdded = g;
  }
  public void plotGrid(Graphics g) {
    int xsize=getSize().width;
    int ysize=getSize().height;
    if (isPrinting) {
      xsize=400;
      ysize=70;
    }
    g.setColor(Color.black);
    for (int j=50; j<512; j+=50) {
      for (int k=0; k<ysize; k+=10) g.drawLine(xsize*j/512, k, xsize*j/512,
k+2);
    }
  }
  public void paint(Graphics g) {
    int x=getSize().width;
    int y=getSize().height;
    if (isPrinting) {
      x=400;
      y=70;
    }
    if (mode==0){
      g.setColor(new Color(red,green,blue));
      g.fillRect(0,0,x,y);
    }
    else if (mode==1){
      int maxInt=0;
      if (red >= maxInt)   maxInt=red;
      if (green >= maxInt) maxInt=green;
      if (blue >= maxInt)  maxInt=blue;
      g.setColor(new Color(maxInt,maxInt,maxInt));
      g.fillRect(0,0,x,y);
```

```
    }
    plot(g);
    if (gridAdded) plotGrid(g);
  }
} // Ende ChartV1_2

/*************************************************************************
*
*    Class ColorCodeV1_0
*     Nrow: Anzahl der Zeilen
*     Ncol: Anzahl der Spalten
*     coding  1: linear
*           2: exponential
*           3: logarithmic
*     farbe   0: Graustufen
*           1: farbig
*     gridAdded: Gitter einzeichenen (true/false)
*     isPrinting: Druckausgabe (true/false)
*
*************************************************************************/
class ColorCodeV1_0 extends JPanel {

  int Nrow,Ncol;
  double[][] data;
  int coding;
  int farbe=1;
  boolean gridAdded, isPrinting;

  public ColorCodeV1_0(int rows, int cols, boolean gA){
    Nrow=rows;
    Ncol=cols;
    gridAdded = gA;
    data = new double[Nrow][Ncol];
  }  // Ende Konstruktor ColorCodeV1_0

  public void setGrid (boolean g) {gridAdded = g;}
  public void setPrinting(boolean p) {
    isPrinting = p;
  }
  public void setScales(int nos) {
    Nrow=nos;
    data = new double[Nrow][Ncol];
  }
  public void setMode(int Mode){
    coding=Mode;
  }
  public void setColor(int colo){
    farbe=colo;
  }
  public void setData(double[][] daten){
    int i,j;
    for(i=0;i<Nrow;i++)
      for(j=0;j<Ncol;j++) data[i][j]=daten[i][j];
  }
  public void plot(Graphics g) {
    int xsize=getSize().width;
    int ysize=getSize().height;
    if (isPrinting){
      xsize=400;
      ysize=250;
    }
    int i,j,color,red,green,blue;
    double max,min,dy,dx;
```

```
    max=-1.0e+14;min=-max;
    for(i=0;i<Nrow;i++)  // determining maximum and minimum
      for(j=0;j<Ncol;j++){
        if(data[i][j]>max) max=data[i][j];
        if(data[i][j]<min) min=data[i][j];
      }
    for(i=0;i<Nrow;i++){
      for(j=0;j<Ncol;j++){
        switch(coding){
          case 1:
            color=(int)(255*(data[i][j]-min)/(max-min)); //color number
            break;
          case 2:
          color=(int)(255*(Math.exp(3.0*(data[i][j]-min)/(max-min))-1.0)
                         /(Math.exp(3.0)-1));
            break;
          case 3:
          color=(int)(255*(Math.log(3.0*(data[i][j]-min)/(max-min)+1.0)
                         /Math.log(4.0)));
            break;
          default: color=0;
        }

        red=255-color;
        green=(int)(255-2.0*Math.abs(128-color));
        if(green<0) green=0;
        blue=color;

        if(farbe==1)g.setColor(new Color(red,green,blue)); // color set
        if(farbe==0)g.setColor(new Color(red,red,red)); // color set
        g.fillRect((int)((j*xsize*1.0)/(Ncol*1.0)),
                   (int)((i*ysize*1.0)/(Nrow*1.0)),
                   (int)((xsize*1.0)/(Ncol*1.0))+1,
                   (int)((ysize*1.0)/(Nrow*1.0))+1);
      }
    }
  }// ends plot
  public void plotGrid(Graphics g) {
    int xsize=getSize().width;
    int ysize=getSize().height;
    if (isPrinting){
      xsize=400;
      ysize=250;
    }
    g.setColor(Color.black);
    for (int l=50; l<512; l+=50) {
       for (int m=0; m<ysize; m+=10) g.drawLine(xsize*l/512, m,
xsize*l/512, m+2);
    }
    if (Nrow>32) {
      for (int n=10; n<Nrow; n+=10) {
      for (int m=0; m<xsize; m+=10) g.drawLine(m, ysize*n/Nrow, m+2,
ysize*n/Nrow);
      }
    }
    else if (Nrow<=32 & Nrow>16) {
      for (int n=5; n<Nrow; n+=5) {
      for (int m=0; m<xsize; m+=10) g.drawLine(m, ysize*n/Nrow, m+2,
ysize*n/Nrow);
      }
    }
    else if (Nrow<=16) {
      for (int n=1; n<Nrow; n++) {
```

```
      for (int m=0; m<xsize; m+=10) g.drawLine(m, ysize*n/Nrow, m+2,
ysize*n/Nrow);
      }
    }
  }
  public void paint(Graphics g) {
    plot(g);
    if (gridAdded) plotGrid(g);
  }
}// Ende ColorCodeV1_0

/*****************************************************************/
/*    continuous wavelet transformation                          */
/*****************************************************************/
class CWT{

final int NMAX=512;


int npkt,nscale;

wavbox wav =new wavbox();

public CWT(int Npkt,int Nscale){
npkt=Npkt;
nscale=Nscale;
}

public void run(double[] signal,String wavelet,double omega0,double[][]
cwtspec){
 int i,j;
 Complex[] extsig= new Complex[2*npkt];
 double nu,dom;
 Complex[] wavspec=new Complex[2*npkt];
 Complex[] a=new Complex[2*npkt];
 double amin,amax,da;


 if(nscale<0) {System.out.println("only a positive number of scales is
allowed\n");
               System.exit(1);
              }


if(wavelet.equals("mexhat")) {wav.setOmega(Math.sqrt(2.0));
                                                    omega0=Math.sqrt(2.0);
                                                   }
if(wavelet.equals("haar")) {wav.setOmega(2.0*Math.PI);
                                                  omega0=2.0*Math.PI;
                                                 }
if(wavelet.equals("morlet")) wav.setOmega(omega0);


if(npkt>NMAX) {System.out.println("Signal too long\n");
               System.exit(1);
               }
if(nscale>256) {System.out.println("Not more than 256 scales allowed\n");
                System.exit(1);
                }



for(i=0;i<2*npkt;i++) {extsig[i]=new Complex(0.0);}
```

```
for(i=0;i<npkt/2;i++) extsig[i]=new Complex(signal[npkt/2-i+1],0.0);
for(i=0;i<npkt;i++) extsig[(npkt/2+i)]= new Complex(signal[i],0.0);;
for(i=0;i<npkt/2;i++) extsig[(3*npkt/2+i)]=new Complex(signal[npkt-i-
1],0.0);



/*    peparation for CWT                              */

nu=Math.PI;             /* Nyquist frequency         */
dom=2.0*nu/npkt;        /* frequency resolution      */
amin=omega0/nu;              /* minimal scale        */
amax=npkt/8.0;               /* maximal scale        */
da=(amax-amin)/nscale;


      /* data spectrum       */

FFT fft = new FFT();
extsig=fft.calcFFT(extsig,true);

for(i=0;i<nscale;i++)        /*    for all scales         */
   {
   for(j=0;j<2*npkt;j++)
    {if(wavelet.equals("morlet"))
            {
             wavspec[j] =new Complex(wav.morlet(-
nu+j*dom/2.0,amin+i*da),0.0);
            }
     if(wavelet.equals("mexhat"))
            {
             wavspec[j]=new Complex(wav.mexhat(-
nu+j*dom/2.0,amin+i*da),0.0);
            }
     if(wavelet.equals("haar"))
            {
             wavspec[j]=wav.haar(-nu+j*dom/2.0,amin+i*da);

            }

     if(!(wavelet.equals("morlet") || wavelet.equals("mexhat") ||
         wavelet.equals("haar")))
        {System.out.println(wavelet+"  wavelet not
implemented\n");System.exit(1);}
    }




  fftshift(wavspec,2*npkt);



   for(j=0;j<2*npkt;j++)     /* spectrum data * spectrum wavelet*/
      {
         //a[j]=extsig[j];
         //a[j].mul(wavspec[j]);
         a[j]=new Complex(extsig[j].re*wavspec[j].re-
extsig[j].im*wavspec[j].im,

extsig[j].im*wavspec[j].re+extsig[j].re*wavspec[j].im);
```

```
       }

 // if(i==15)
 //  for(j=0;j<2*npkt;j++) System.out.println(a[j].re+"\t"+a[j].im);

 a=fft.calcFFT(a,false);           /* W_signal(a)               */



  for(j=0;j<npkt;j++)        //output of modulus
          cwtspec[i][j]=Math.sqrt(a[(npkt/2+j)].re*a[(npkt/2+j)].re+
                    a[(npkt/2+j)].im*a[(npkt/2+j)].im);



 }// end of loop over all scales



}// end of method run


private void fftshift(Complex[] a,int n)
{Complex z=new Complex(0.0);
 for(int i=0;i<n/2;i++)
    {z=a[n/2+i];a[n/2+i]=a[i];a[i]=z;
    }
} //end of method fftshift

}//end of class cwt1

//*****************************************************************
//
//*****************************************************************
class Complex
{
    double re, im;

    // Constructor 1: re&euml;el en imaginair deel opgeven:
    public Complex(double r, double i)
    {
        re = r;
        im = i;
    }

    // Constructor 2: hoek theta opgeven via exp(it) = cos t + i sin t:
    public Complex(double theta)
    {
         re = Math.cos(theta);
            im = Math.sin(theta);
    }

    // Tel een complex getal op bij dit object:
    public void add(Complex c1)
    {
        re += c1.re;
        im += c1.im;
    }

    public void sub(Complex c1) {
```

```
    re-=c1.re;
    im-=c1.im;
    }
    // Vermenigvuldig dit object met een complex getal:
    public void mul(Complex c1)
    {
        double rt = re;

        re = re *c1.re -im *c1.im;
        im = rt *c1.im +im *c1.re;
    }

    public void conj()
    {
        im = -im;
    }

}

/************************************************
*                                       *
* Klasse FFT                            *
*                                       *
************************************************/
class FFT
{
   public static Complex[] calcFFT(Complex x[], boolean forward)
   {
      int n = x.length;
      Complex omega[] = new Complex[n>>1];
      double theta = -2*Math.PI/(double)n;
      if (!forward) theta = -theta;
      for (int i=0; i<(n>>1);i++)
      {
         omega[i] = new Complex((double)i*theta);
      }
      Complex y[] = calcFFT(x,omega);
      if (!forward)
         for(int i=0; i<n; i++)
         {
            y[i].re/=n;
            y[i].im/=n;
         }
      return y;
   }

   public static Complex[] calcFFT(Complex x[],Complex omega[])
   {
         int n = x.length, k=0, shift, shifted=0;
         if (n==1)
         {
            Complex z1[] = new Complex[1];
            z1[0] = new Complex(x[0].re,x[0].im);
            return z1;
         }

         shift= (omega.length/(n>>1));

         Complex z[] = new Complex[n>>1];
         Complex w[] = new Complex[n>>1];

         for (k=0; k<n>>1; k++)
         {
```

```
            z[k] = new Complex(x[k<<1].re,x[k<<1].im);
            w[k] = new Complex(x[(k<<1)+1].re,x[(k<<1)+1].im);
         }

         z = calcFFT(z,omega);
         w = calcFFT(w,omega);

         Complex y[] = new Complex[n];
         for (k=0; k<n>>1;k++)
         {
            y[k] = new Complex(z[k].re,z[k].im);
            w[k].mul(omega[shifted]);
            y[k].add(w[k]);
            y[k+(n>>1)] = new Complex(z[k].re,z[k].im);
            y[k+(n>>1)].sub(w[k]);
            shifted += shift;
         }
         return y;
   }


}

/************************************************
*                                      *
* Klasse wavebox                       *
*                                      *
************************************************/
class wavbox{

double omega0;
final double EPS=1.0e-14;

public wavbox(){
omega0=1.0;
}

public void setOmega(double om0){
omega0=om0;
}

public Complex  haar(double omega,double a)
{double modulus;
  Complex z = new Complex(0.0);
 if(Math.abs(a*omega)<EPS)
      modulus=Math.sin(a*omega/4.0);
 else
   modulus=Math.sin(a*omega/4.0)/(a*omega/4.0)*Math.sin(a*omega/4.0);
 z.re=Math.sin(a*omega/2.0)*modulus/Math.sqrt(2.0*Math.PI);
 z.im=Math.cos(a*omega/2.0)*modulus/Math.sqrt(2.0*Math.PI);
return z;
} // ends method haar


double mexhat(double omega,double a)
{double z;
 z=a*a*omega*omega;
return(Math.sqrt(a)*z*Math.exp(-0.5*z));
}// ends method mexhat

double morlet(double omega,double a)
{double z;
 z=(omega0/a-omega);z=z*z*a*a/2.0;
```

```
 return(Math.exp(-z)*a/Math.sqrt(Math.sqrt(Math.PI)));
} // ends method morlet

}// ends class wavbox
```

Anhang B

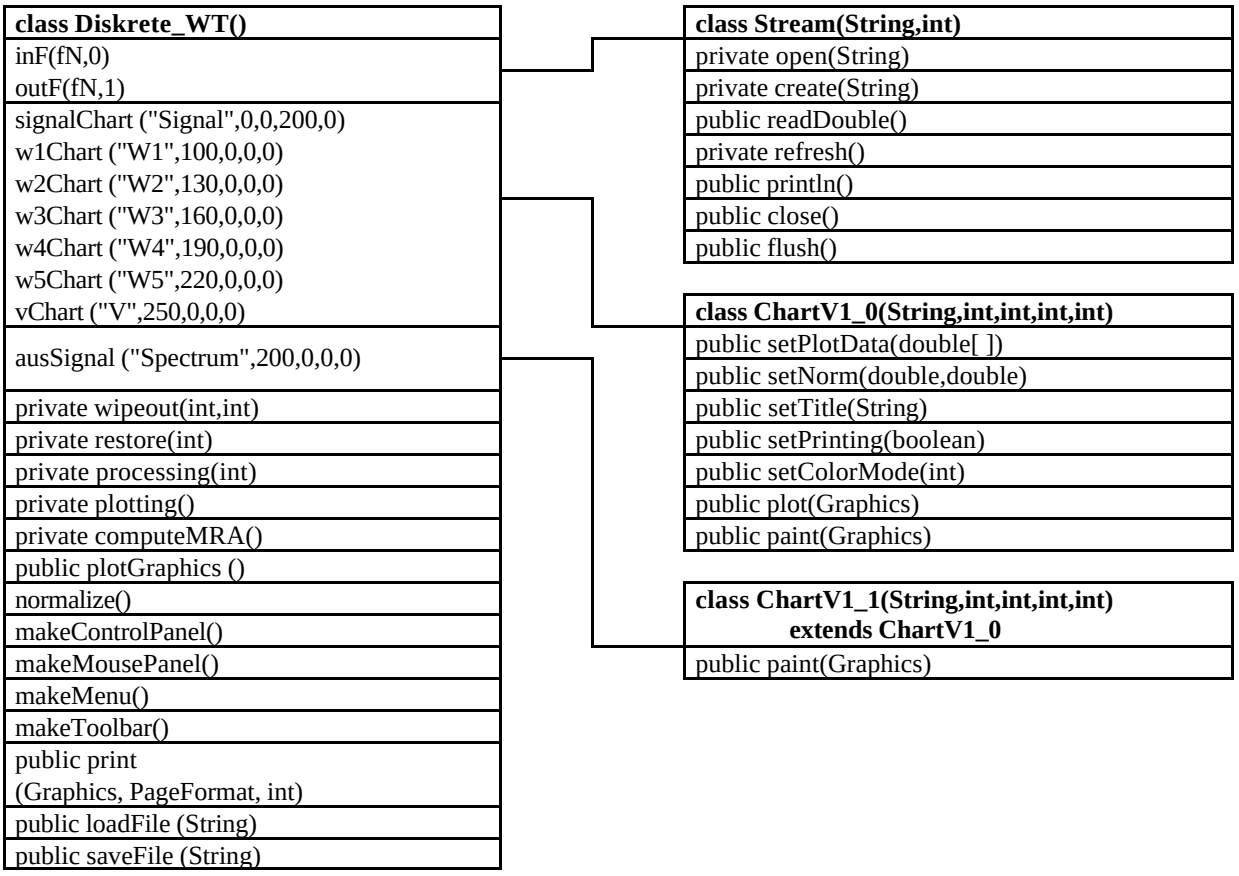
Die Java-Application zur diskreten Wavelet-Transformation

Die grafische Benutzeroberfläche wurde mit Hilfe des GridBagLayout erstellt. Die folgende Abbildung soll die Anordnung der Elemente veranschaulichen.

<table>
<tr><th></th><th>0</th><th>1</th></tr>
<tr><td>0</td><td colspan="2">toolBar</td></tr>
<tr><td>1</td><td rowspan="9">controlPanel</td><td>signalChart</td></tr>
<tr><td>2</td><td>ausSignal</td></tr>
<tr><td>3</td><td>w1Chart</td></tr>
<tr><td>4</td><td>w2Chart</td></tr>
<tr><td>5</td><td>w3Chart</td></tr>
<tr><td>6</td><td>w4Chart</td></tr>
<tr><td>7</td><td>w5Chart</td></tr>
<tr><td>8</td><td>vChart</td></tr>
<tr><td>9</td><td>mousePanel</td></tr>
</table>

Abbildung B.1: Das GridBagLayout

Folgende Grafik soll die verschiedenen Klassen in kurzer Form beschreiben. Die Hauptklasse Diskrete_WT erzeugt weitere Objekt aus anderen Klassen. Hier sollen nur die Klassen, die die Grafiken erzeugen sowie die Klasse mit der die Daten ein- bzw. ausgelesen werden können, dargestellt werden.



```
import java.io.*;
import java.util.*;
import java.text.*;
import java.io.File;
import java.awt.*;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.*;
import java.awt.color.*;
import javax.swing.*;
import javax.swing.AbstractButton;
import javax.swing.JComponent;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.ImageIcon;
import javax.swing.filechooser.*;
import javax.swing.text.*;
import javax.swing.event.*;
import java.awt.print.*;
import java.awt.print.PrinterJob;
import java.awt.geom.*;

public class Diskrete_WT     extends     JFrame
                             implements  ActionListener, ItemListener,
                                         ChangeListener, Printable,
                                         MouseMotionListener{

  // GUI-Variablen************
  JMenuBar  menu;
  JMenu     file, extra;
  JMenuItem       load,save,exit,print,demoData;
  JMenu           representation,colorCode;
  JRadioButtonMenuItem  coeff, mra, bw, rgb;
  JToolBar  toolBar;
  JButton   tbexit,tbopen,tbsave,tbprint,tbmsa,tbwc,tbbw,tbrgb;
  JPanel    controlPanel,mousePanel;
  JPanel    waveletChoicePanel,scalePanel,treshPanel,buttonPanel;
  JSlider   treshold;
  JTextField      textFeld, mouseTextField;
  JButton   undo, apply;
  Jlabel    mouseLabel;
  JComboBox scale,waveletChoice,zerlegungstiefe;

  boolean   mraChoosen = false;
  String    inFileName,outFileName;
  Stream    inF,outF;
  JFileChooser    inFile = new JFileChooser();
  JFileChooser  outFile = new JFileChooser();

  double[] sigback = new double[512];
  double[] signal = new double[512];
  double[] spectrum = new double[512];
  double[] specback = new double[512];
  double[] v = new double[512];
  double[] w5 = new double[512];
  double[] w4 = new double[512];
  double[] w3 = new double[512];
  double[] w2 = new double[512];
```

```
double[] w1 = new double[512];
int length_filt,level=1,zt=5;
String wavelet;
int fps;
ChartV1_0 signalChart = new ChartV1_0("Signal",0,0,200,0);
CHARTV1_1 ausSignal = new CHARTV1_1("Spectrum",200,0,0,0);
ChartV1_0 vChart = new ChartV1_0("V",250,0,0,0);
ChartV1_0 w5Chart = new ChartV1_0("W5",220,0,0,0);
ChartV1_0 w4Chart = new ChartV1_0("W4",190,0,0,0);
ChartV1_0 w3Chart = new ChartV1_0("W3",160,0,0,0);
ChartV1_0 w2Chart = new ChartV1_0("W2",130,0,0,0);
ChartV1_0 w1Chart = new ChartV1_0("W1",100,0,0,0);

daubechies wavbox = new daubechies();

public Diskrete_WT() {
  makeMenu();         // Initialisierung
  makeToolbar();
  makeControlPanel();
  makeMousePanel();

  //  Einrichten des GridBagLayouts***********************
  Container contentPane = getContentPane();
  GridBagLayout gridbag = new GridBagLayout();
  GridBagConstraints c = new GridBagConstraints();
  contentPane.setLayout(gridbag);
  c.fill = GridBagConstraints.BOTH;
  c.weightx = 0.5;
  // ToolBar*********************************************
  c.gridx = 0;
  c.gridy = 0;
  c.gridwidth = 2;
  gridbag.setConstraints(toolBar, c);
  contentPane.add(toolBar);
  // ControlPanel****************************************
  c.gridx = 0;
  c.gridy = 1;
  c.gridwidth = 1;
  c.gridheight = 9;
  c.weightx = 0.0;
  gridbag.setConstraints(controlPanel, c);
  contentPane.add(controlPanel);
  // Eingangsingnal**************************************
  c.ipady = 30;
  c.ipadx = 540;
  c.weighty = 1.0;
  c.weightx = 0.5;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.gridx = 1;
  c.gridy = 1;
  gridbag.setConstraints(signalChart, c);
  contentPane.add(signalChart);
  signalChart.setVisible(true);
  // Spektrum********************************************
  c.ipady = 30;
  c.ipadx = 540;
  c.weighty = 3.0;
  c.weightx = 0.5;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.gridx = 1;
  c.gridy = 2;
```

```
gridbag.setConstraints(ausSignal, c);
contentPane.add(ausSignal);
ausSignal.setVisible(true);
ausSignal.addMouseMotionListener(this);
// W1*****************************************************
c.ipady = 30;
c.ipadx = 540;
c.weighty = 1.0;
c.weightx = 0.5;
c.gridwidth = 1;
c.gridheight = 1;
c.gridx = 1;
c.gridy = 3;
gridbag.setConstraints(w1Chart, c);
contentPane.add(w1Chart);
w1Chart.setVisible(false);
// W2*****************************************************
c.ipady = 30;
c.ipadx = 540;
c.weighty = 1.0;
c.weightx = 0.5;
c.gridwidth = 1;
c.gridheight = 1;
c.gridx = 1;
c.gridy = 4;
gridbag.setConstraints(w2Chart, c);
contentPane.add(w2Chart);
w2Chart.setVisible(false);
// W3*****************************************************
c.ipady = 30;
c.ipadx = 540;
c.weighty = 1.0;
c.weightx = 0.5;
c.gridwidth = 1;
c.gridheight = 1;
c.gridx = 1;
c.gridy = 5;
gridbag.setConstraints(w3Chart, c);
contentPane.add(w3Chart);
w3Chart.setVisible(false);
// W4*****************************************************
c.ipady = 30;
c.ipadx = 540;
c.weighty = 1.0;
c.weightx = 0.5;
c.gridwidth = 1;
c.gridheight = 1;
c.gridx = 1;
c.gridy = 6;
gridbag.setConstraints(w4Chart, c);
contentPane.add(w4Chart);
w4Chart.setVisible(false);
// W5*****************************************************
c.ipady = 30;
c.ipadx = 540;
c.weighty = 1.0;
c.weightx = 0.5;
c.gridwidth = 1;
c.gridheight = 1;
c.gridx = 1;
c.gridy = 7;
gridbag.setConstraints(w5Chart, c);
contentPane.add(w5Chart);
```

```
    w5Chart.setVisible(false);
    // V*************************************************
    c.ipady = 30;
    c.ipadx = 540;
    c.weighty = 1.0;
    c.weightx = 0.5;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.gridx = 1;
    c.gridy = 8;
    gridbag.setConstraints(vChart, c);
    contentPane.add(vChart);
    vChart.setVisible(false);
    // MousePanel*****************************************
    c.ipady = 0;
    c.ipadx = 0;
    c.weighty = 0;
    c.weightx = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.gridx = 1;
    c.gridy = 9;
    gridbag.setConstraints(mousePanel, c);
    contentPane.add(mousePanel);

    // Fenster schließen durch [x]
    addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e) {
         dispose();
         System.exit(0);
       }
        });
  }         // Ende Konstruktor************************

  //
Behandler*****************************************************************
  public void actionPerformed (ActionEvent e) {
    // Programm beenden
    if (e.getSource() == exit || e.getSource() == tbexit) {
      dispose();
      System.exit(0);
    }
    // Datei laden und Signal darstellen
    else if  (e.getSource() == load || e.getSource() == tbopen) {
      inFile.showOpenDialog(Diskrete_WT.this);
      File file = inFile.getSelectedFile();
      inFileName = file.getAbsolutePath();
      try {
            loadFile(inFileName);
      }
      catch (IOException e1) {
      System.out.println(e1);
      }
      signalChart.setPlotData(sigback);
      plotGraphics();
      waveletChoice.setEnabled(true);
      normalize();
    }
    // Demo-Signal erstellen und darstellen
    else if  (e.getSource() == demoData) {
      double sin[] = new double[512];
      for (int l=0; l<100; l++){
        sin[l]= 10*(Math.sin((double)l/0.5));
```

```
  }
  for (int l=0; l<200; l++){
    sin[l]= sin[l]+8*(Math.sin((double)l/1));
  }
  for (int l=0; l<270; l++){
    sin[l]= sin[l]+5*(Math.sin((double)l/2));
  }
  for (int l=0; l<420; l++){
    sin[l]= sin[l]+2*(Math.sin((double)l/5));
    if(l==330) sin[l]=50;
    if(l==400) sin[l]=-50;
  }
  for (int l=0; l<512; l++){
    sin[l]= sin[l]+4*(Math.sin((double)l/10));
  }
  sigback = sin;
    signalChart.setPlotData(sigback);
    waveletChoice.setEnabled(true);
  plotGraphics();
  normalize();
  inFileName="Demo-Data";
}
// Spektrum-Daten als ASCII-Datei abspeichern
else if  (e.getSource() == save || e.getSource() == tbsave) {
  int returnVal = outFile.showSaveDialog(Diskrete_WT.this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
    File file = outFile.getSelectedFile();
    //this is where a real application would save the file.
    outFileName = file.getAbsolutePath();
  try {
          saveFile(outFileName);
  }
  catch (IOException e1) {
    System.out.println(e1);
  }
  }
  else {}
}
// Signal und Spektrum neu berechnen und darstellen
else if  (e.getSource() == apply) {
  wipeout(level, fps);
  plotting();
  undo.setEnabled(true);
}
// Letzte "apply"-Aktion widerrufen
else if  (e.getSource() == undo) {
  restore(level);
  plotting();
  undo.setEnabled(false);
}
// Berechnen und darstellen der MSA
else if  (e.getSource() == tbmsa || e.getSource() == mra) {
  tbwc.setEnabled(true);
  tbmsa.setEnabled(false);
  mra.setSelected(true);
  zerlegungstiefe.setEnabled(true);
  mraChoosen=true;
  computeMRA();
  plotGraphics();
}
// Wavelet-Spektrum darstellen
else if  (e.getSource() == tbwc || e.getSource() == coeff) {
  tbmsa.setEnabled(true);
```

```
      tbwc.setEnabled(false);
      coeff.setSelected(true);
      zerlegungstiefe.setEnabled(false);
      mraChoosen=false;
      plotGraphics();
    }
    // Grafiken in Schwarz/Weiss darstellen
    else if  (e.getSource() == tbbw || e.getSource() == bw) {
      w1Chart.setColorMode(1);
      w2Chart.setColorMode(1);
      w3Chart.setColorMode(1);
      w4Chart.setColorMode(1);
      w5Chart.setColorMode(1);
      vChart.setColorMode(1);
      signalChart.setColorMode(1);
      ausSignal.setColorMode(1);
      tbrgb.setEnabled(true);
      tbbw.setEnabled(false);
      bw.setSelected(true);
      plotGraphics();
    }
    // Grafiken farbig darstellen
    else if  (e.getSource() == tbrgb || e.getSource() == rgb) {
      w1Chart.setColorMode(0);
      w2Chart.setColorMode(0);
      w3Chart.setColorMode(0);
      w4Chart.setColorMode(0);
      w5Chart.setColorMode(0);
      vChart.setColorMode(0);
      signalChart.setColorMode(0);
      ausSignal.setColorMode(0);
      tbrgb.setEnabled(false);
      tbbw.setEnabled(true);
      rgb.setSelected(true);
      plotGraphics();
    }
    // Grafiken ausdrucken
    else if (e.getSource() == tbprint || e.getSource() == print) {
      PrinterJob printJob = PrinterJob.getPrinterJob();
      printJob.setPrintable(this);
      if (printJob.printDialog()) {
        try {
          printJob.print();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
  }

  public void itemStateChanged (ItemEvent e) {
    Object picked = e.getSource();
    // Wahl der Skala
    if (picked == scale) {
      JComboBox cb = (JComboBox)e.getSource();
      String z = (String)cb.getSelectedItem();
      level = Integer.parseInt(z);
    }
    // Wahl der Zerlegungstiefe
    if (picked == zerlegungstiefe) {
      JComboBox cb = (JComboBox)e.getSource();
      zt = Integer.parseInt((String)cb.getSelectedItem());
      computeMRA();
```

```
    }
    // Wahl des Wavelets
    else if (picked == waveletChoice){
      JComboBox cb = (JComboBox)e.getSource();
      wavelet = (String)cb.getSelectedItem();
      if (wavelet == "Haar") {processing(1);}
      else if (wavelet == "Daubechies4") {processing(2);}
      else if (wavelet == "Daubechies6") {processing(3);}
      else if (wavelet == "Daubechies8") {processing(4);}
      if (wavelet != "none") {
      save.setEnabled(true);
      tbsave.setEnabled(true);
      print.setEnabled(true);
      tbprint.setEnabled(true);
      }
      if (wavelet != "none") {
            scale.setEnabled(true);
      apply.setEnabled(true);
      treshold.setEnabled(true);
      }
      else if (wavelet == "none") {
            scale.setEnabled(false);
      apply.setEnabled(false);
      treshold.setEnabled(false);
      }
    }
  }

  public void stateChanged(ChangeEvent e) {
    JSlider source = (JSlider)e.getSource();
    // Threshold einstellen
    if (!source.getValueIsAdjusting()) {
      fps = (int)source.getValue();
      textFeld.setText(String.valueOf(fps));
    }
  }

  private static DecimalFormat N = new DecimalFormat();
  private static final String spaces = "                    ";
  String format(double number, int align, int frac) {
    N.setGroupingUsed(false);
      N.setMaximumFractionDigits(frac);
      N.setMinimumFractionDigits(frac);
      String num = N.format(number);
      if (num.length() < align)
         num = spaces.substring(0,align-num.length()) + num;
    return num;
  }

  public void mouseMoved(MouseEvent e) {
    int x = ausSignal.getSize().width;
    int y = ausSignal.getSize().height;
    double xs = (double)e.getX() / (double) x;
    double ys = (double)e.getY() / (double) y;
    mouseTextField.setEnabled(true);
    mouseTextField.setText(" (" + format(512*xs,4,0) + " / " +
format(ys,4,2) + ")");
    mouseTextField.setEnabled(true);
  }
  public void mouseDragged(MouseEvent e) {
  }

  // Methoden****************************************
```

```
  private void wipeout(int level,int tresh){
    int i,start;
    double norm;
    start=1;
    for(i=0;i<9-level;i++) start*=2;
    norm=0.0;
    for(i=start;i<2*start;i++) norm+=spectrum[i]*spectrum[i];
    norm=Math.sqrt(norm/start);
    for(i=start;i<2*start;i++){
      specback[i]=spectrum[i];
      if(Math.abs(spectrum[i])<norm*tresh*0.05) {spectrum[i]=0.0;}
    }
  }
  private void restore(int level) {
    int i,start;
    start=1;
    for(i=0;i<9-level;i++) start*=2;
    for(i=start;i<2*start;i++) spectrum[i]=specback[i];
  }
  private void processing(int order) {
    int i;
    length_filt=wavbox.daub_init(order);
    for(i=0;i<512;i++) spectrum[i]=sigback[i];
    i=wavbox.decomp(spectrum,512,length_filt);
    ausSignal.setPlotData(spectrum);
    signalChart.setPlotData(sigback);
    if (mraChoosen) computeMRA();
    plotGraphics();
  }
  private void plotting() {
    int i;
    ausSignal.setPlotData(spectrum);
    for(i=0;i<512;i++) signal[i]=spectrum[i];
    i=wavbox.reconst(signal,512,length_filt);
    signalChart.setPlotData(signal);
    plotGraphics();
    if (mraChoosen) computeMRA();
  }
  private void computeMRA() {
    for (int i=0; i<512; i++)
{w1[i]=0;w2[i]=0;w3[i]=0;w4[i]=0;w5[i]=0;v[i]=0;}
    for (int i=256; i<512; i++) w1[i] = spectrum[i];
    wavbox.reconst(w1,512,length_filt);
    w1Chart.setPlotData(w1);
    if(zt>=2){
    for (int i=128; i<256; i++) w2[i] = spectrum[i];
    wavbox.reconst(w2,512,length_filt);
    w2Chart.setPlotData(w2);
    }
    if(zt>=3){
    for (int i=64; i<128; i++) w3[i] = spectrum[i];
    wavbox.reconst(w3,512,length_filt);
    w3Chart.setPlotData(w3);
    }
    if(zt>=4){
    for (int i=32; i<64; i++) w4[i] = spectrum[i];
    wavbox.reconst(w4,512,length_filt);
    w4Chart.setPlotData(w4);
    }
    if(zt>=5){
    for (int i=16; i<32; i++) w5[i] = spectrum[i];
    wavbox.reconst(w5,512,length_filt);
    w5Chart.setPlotData(w5);
```

```
    }
    for (int i=0; i<(int)Math.pow(2,4+5-zt); i++) v[i] = spectrum[i];
    wavbox.reconst(v,512,length_filt);
    vChart.setPlotData(v);
    vChart.setTitle("V"+zt);
    plotGraphics();
  }
  public void plotGraphics () {
    w1Chart.setVisible(mraChoosen);
    w2Chart.setVisible(mraChoosen && zt>=2);
    w3Chart.setVisible(mraChoosen && zt>=3);
    w4Chart.setVisible(mraChoosen && zt>=4);
    w5Chart.setVisible(mraChoosen && zt>=5);
    vChart.setVisible(mraChoosen);
    ausSignal.setVisible(!mraChoosen);
    mousePanel.setVisible(!mraChoosen);
    if(mraChoosen) w1Chart.paint(w1Chart.getGraphics());
    if(mraChoosen && zt>=2) w2Chart.paint(w2Chart.getGraphics());
    if(mraChoosen && zt>=3) w3Chart.paint(w3Chart.getGraphics());
    if(mraChoosen && zt>=4) w4Chart.paint(w4Chart.getGraphics());
    if(mraChoosen && zt>=5) w5Chart.paint(w5Chart.getGraphics());
    if(mraChoosen) vChart.paint(vChart.getGraphics());
    signalChart.paint(signalChart.getGraphics());
    if(!mraChoosen) ausSignal.paint(ausSignal.getGraphics());
  }
  void normalize() {    // setzt die max- und min-Werte zur besseren
Darstellung
    double max,min,minMax;
    max=-1.0e+14;min=-max;
    for(int i=0;i<512;i++) {
       if(sigback[i]>max) {max=sigback[i];}
       if(sigback[i]<min) {min=sigback[i];}
    }
    minMax=max-min;
    w1Chart.setNorm(minMax,min);
    w2Chart.setNorm(minMax,min);
    w3Chart.setNorm(minMax,min);
    w4Chart.setNorm(minMax,min);
    w5Chart.setNorm(minMax,min);
    vChart.setNorm(minMax,min);
  }
  void makeControlPanel() {
    controlPanel = new JPanel();
    controlPanel.setLayout(new GridLayout(4,1,5,5));
    controlPanel.setBorder(BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("Wavelet-Control"),
                    BorderFactory.createEmptyBorder(5,5,5,5)));
    controlPanel.setMaximumSize(new Dimension(90,200));
      waveletChoicePanel = new JPanel();
      waveletChoicePanel.setLayout(new BoxLayout(waveletChoicePanel,
BoxLayout.Y_AXIS));
      waveletChoicePanel.setBorder(BorderFactory.createCompoundBorder(
                      BorderFactory.createTitledBorder("Choose Wavelet"),
                    BorderFactory.createEmptyBorder(15,5,30,5)));
      waveletChoicePanel.setPreferredSize(new Dimension(90,80));
        waveletChoice = new JComboBox();
        waveletChoice.setAlignmentX(JComponent.CENTER_ALIGNMENT);
        waveletChoice.addItem("none");
        waveletChoice.addItem("Haar");
        waveletChoice.addItem("Daubechies4");
        waveletChoice.addItem("Daubechies6");
        waveletChoice.addItem("Daubechies8");
        waveletChoice.setEnabled(false);
```

```
      waveletChoice.addItemListener(this);
    waveletChoicePanel.add(waveletChoice);
  controlPanel.add(waveletChoicePanel);
    scalePanel = new JPanel();
    scalePanel.setLayout(new BoxLayout(scalePanel, BoxLayout.Y_AXIS));
    scalePanel.setBorder(BorderFactory.createCompoundBorder(
                  BorderFactory.createTitledBorder("Select Scale"),
                  BorderFactory.createEmptyBorder(5,5,5,5)));
    scalePanel.setAlignmentY(CENTER_ALIGNMENT);
    scale = new JComboBox();
    for (int i=1; i<10; i++) {
      scale.addItem(String.valueOf(i));
    }
    scale.setEnabled(false);
    scale.addItemListener(this);
    scalePanel.add(scale);
  controlPanel.add(scalePanel);
    treshPanel = new JPanel();
    treshPanel.setLayout(new BoxLayout(treshPanel, BoxLayout.Y_AXIS));
    treshPanel.setBorder(BorderFactory.createCompoundBorder(
                  BorderFactory.createTitledBorder("Set Threshold"),
                  BorderFactory.createEmptyBorder(5,5,5,5)));
    treshPanel.setAlignmentY(CENTER_ALIGNMENT);
      treshold = new JSlider(JSlider.HORIZONTAL,0,100,1);
    treshold.setPreferredSize(new Dimension (100,45));
    treshold.setMajorTickSpacing(50);
    treshold.setMinorTickSpacing(10);
    treshold.setPaintTicks(true);
    treshold.setPaintLabels(true);
    treshold.setEnabled(false);
    treshold.addChangeListener(this);
    textFeld = new JTextField ("0",1);
    textFeld.setHorizontalAlignment(JTextField.RIGHT);
    textFeld.setMaximumSize(new Dimension(35,25));
    textFeld.setEnabled(true);
    textFeld.setEditable(false);
    treshPanel.add(treshold);
    treshPanel.add(textFeld);
  controlPanel.add(treshPanel);
    buttonPanel = new JPanel();
    buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
    apply = new JButton("Apply");
    apply.setAlignmentY(JComponent.BOTTOM_ALIGNMENT);
    apply.setMnemonic(KeyEvent.VK_A);
    apply.setEnabled(false);
    apply.addActionListener(this);
    apply.setToolTipText("Compute Thresholding");
    undo = new JButton("Undo");
    undo.setAlignmentY(JComponent.BOTTOM_ALIGNMENT);
    undo.setMnemonic(KeyEvent.VK_U);
    undo.setEnabled(false);
    undo.addActionListener(this);
    undo.setToolTipText("Remove last Thresholding");
    buttonPanel.add(apply);
    buttonPanel.add(undo);
  controlPanel.add(buttonPanel);
}

void makeMousePanel() {
  mousePanel = new JPanel();
  mousePanel.setLayout (new FlowLayout(FlowLayout.RIGHT,1,1));
    mouseLabel = new JLabel ("Mouse Motion ");
    mouseTextField = new JTextField (" ",7);
```

```
    mouseTextField.setHorizontalAlignment(JTextField.RIGHT);
    mouseTextField.setEnabled(true);
    mouseTextField.setEditable(false);
  mousePanel.add(mouseLabel);
  mousePanel.add(mouseTextField);
}

void makeMenu() {
  ImageIcon openButtonIcon = new ImageIcon("open.gif");
  ImageIcon saveButtonIcon = new ImageIcon("save.gif");
  ImageIcon exitButtonIcon = new ImageIcon("exit.gif");
  ImageIcon printButtonIcon = new ImageIcon("print1.gif");
    menu = new JMenuBar();
      file = new JMenu("File");
    file.setMnemonic(KeyEvent.VK_F);
      load = new JMenuItem("Open...", openButtonIcon);
        load.addActionListener(this);
        load.setMnemonic(KeyEvent.VK_O);
      load.setAccelerator(KeyStroke.getKeyStroke(
          KeyEvent.VK_O, ActionEvent.CTRL_MASK));
      save = new JMenuItem("Save as...", saveButtonIcon);
      save.setEnabled(false);
        save.addActionListener(this);
        save.setMnemonic(KeyEvent.VK_S);
      save.setAccelerator(KeyStroke.getKeyStroke(
          KeyEvent.VK_S, ActionEvent.CTRL_MASK));
      exit = new JMenuItem("Exit", exitButtonIcon);

      exit.addActionListener(this);
      print = new JMenuItem("Print..", printButtonIcon);
      print.addActionListener(this);
      print.setEnabled(false);
        print.setMnemonic(KeyEvent.VK_P);
      print.setAccelerator(KeyStroke.getKeyStroke(
          KeyEvent.VK_P, ActionEvent.CTRL_MASK));
    file.add(load);
    file.add(save);
    file.addSeparator();
    file.add(print);
    file.addSeparator();
    file.add(exit);
    menu.add(file);
      extra = new JMenu("Extra");
    extra.setMnemonic(KeyEvent.VK_E);
      representation = new JMenu("Representation");
        ButtonGroup group = new ButtonGroup();
          coeff = new JRadioButtonMenuItem("Wavelet-Coefficients");
          coeff.setSelected(true);
          coeff.addActionListener(this);
        group.add(coeff);
          mra = new JRadioButtonMenuItem("Multi-Resolution-Analysis");
          mra.addActionListener(this);
        group.add(mra);
      representation.add(coeff);
      representation.add(mra);
      extra.add(representation);
      colorCode = new JMenu("Color-Code");
        ButtonGroup group1 = new ButtonGroup();
          bw = new JRadioButtonMenuItem("Black/White");
          bw.addActionListener(this);
        group1.add(bw);
          rgb = new JRadioButtonMenuItem("RGB");
          rgb.addActionListener(this);
```

```
            rgb.setSelected(true);
          group1.add(rgb);
        colorCode.add(bw);
        colorCode.add(rgb);
        extra.addSeparator();
        extra.add(colorCode);
        demoData = new JMenuItem("Demo-Data");
        demoData.addActionListener(this);
        extra.addSeparator();
      extra.add(demoData);
      menu.add(extra);
    setJMenuBar(menu);
  }
  // Druckt die dargestellten Grafiken aus
  public int print(Graphics g, PageFormat pf, int pi) throws
PrinterException {
    if (pi >= 1) {
      return Printable.NO_SUCH_PAGE;
    }
    Graphics2D g2 = (Graphics2D) g;
    if (!mraChoosen){
      g2.translate(100,80);
      signalChart.setPrinting(true);
      signalChart.paint(g2);
      signalChart.setPrinting(false);
      g2.translate(0,70);
      ausSignal.setPrinting(true);
      ausSignal.paint(g2);
      ausSignal.setPrinting(false);
    }
    if (mraChoosen){
      g2.translate(100,80);
      signalChart.setPrinting(true);
      signalChart.paint(g2);
      signalChart.setPrinting(false);
      g2.translate(0,70);
      w1Chart.setPrinting(true);
      w1Chart.paint(g2);
      w1Chart.setPrinting(false);
      if(zt>=2) {
      g2.translate(0,70);
      w2Chart.setPrinting(true);
      w2Chart.paint(g2);
      w2Chart.setPrinting(false);
      }
      if(zt>=3) {
      g2.translate(0,70);
      w3Chart.setPrinting(true);
      w3Chart.paint(g2);
      w3Chart.setPrinting(false);
      }
      if(zt>=4) {
      g2.translate(0,70);
      w4Chart.setPrinting(true);
      w4Chart.paint(g2);
      w4Chart.setPrinting(false);
      }
      if(zt>=5) {
      g2.translate(0,70);
      w5Chart.setPrinting(true);
      w5Chart.paint(g2);
      w5Chart.setPrinting(false);
      }
```

```
    g2.translate(0,70);
    vChart.setPrinting(true);
    vChart.paint(g2);
    vChart.setPrinting(false);
  }
    g2.translate(0,100);
    g2.setColor(Color.black);
    g2.setFont(new Font ("Helvetica", Font.PLAIN, 10));
    g2.drawString("Wavelettype: "+wavelet,0,0);
    g2.drawString("Inputfile: "+inFileName,0,20);
    return Printable.PAGE_EXISTS;
}

void makeToolbar() {
  toolBar = new JToolBar();
    JButton button = null;
    tbopen = new JButton(new ImageIcon("open.gif"));
    tbopen.addActionListener(this);
    tbopen.setToolTipText("Open File");
  toolBar.add(tbopen);
    tbsave = new JButton(new ImageIcon("save.gif"));
    tbsave.addActionListener(this);
    tbsave.setToolTipText("Save File");
    tbsave.setEnabled(false);
  toolBar.add(tbsave);
    tbprint = new JButton(new ImageIcon("print1.gif"));
    tbprint.addActionListener(this);
    tbprint.setToolTipText("Print");
    tbprint.setEnabled(false);
  toolBar.add(tbprint);
  toolBar.addSeparator();
    tbwc = new JButton(new ImageIcon("wc.gif"));
    tbwc.setEnabled(false);
    tbwc.addActionListener(this);
    tbwc.setToolTipText("Wavelet-Coefficients");
  toolBar.add(tbwc);
    tbmsa = new JButton(new ImageIcon("msa.gif"));
    tbmsa.addActionListener(this);
    tbmsa.setToolTipText("Multi-Resolution-Analysis");
  toolBar.add(tbmsa);
    zerlegungstiefe = new JComboBox();
    for (int i=5; i>=1; i--) {
      zerlegungstiefe.addItem(String.valueOf(i));
    }
    zerlegungstiefe.setMaximumSize(new Dimension(40,23));
    zerlegungstiefe.setEnabled(false);
    zerlegungstiefe.addItemListener(this);
    zerlegungstiefe.setToolTipText("Zerlegungstiefe");
  toolBar.add(zerlegungstiefe);
  toolBar.addSeparator();
    tbbw = new JButton(new ImageIcon("bw1.gif"));
    tbbw.addActionListener(this);
    tbbw.setToolTipText("Black/White");
  toolBar.add(tbbw);
    tbrgb = new JButton(new ImageIcon("rgb1.gif"));
    tbrgb.setEnabled(false);
    tbrgb.addActionListener(this);
    tbrgb.setToolTipText("Colored");
  toolBar.add(tbrgb);
  toolBar.addSeparator();
    tbexit = new JButton(new ImageIcon("exit.gif"));
    tbexit.addActionListener(this);
    tbexit.setToolTipText("Exit");
```

```
    toolBar.add(tbexit);
  }
  public void loadFile (String fN) throws  IOException {
      inF = new Stream (fN, 0);
      // Daten Einlesen
      try {
        for (int i=0; i<512; i++) {
          sigback[i] = inF.readDouble();
        }
      }
      catch (EOFException e) {}
  }
  public void saveFile (String fN) throws  IOException {
      outF = new Stream (fN, 1);
      // Daten Einlesen
      try {
        for (int i=0; i<512; i++) {
          outF.println(signal[i]);
        }
      outF.close();
      }
      catch (EOFException e) {}
  }

    public String toString (double x) {
      return " "+x;
    }

  public static void main(String args[]) throws  IOException {
    Diskrete_WT window = new Diskrete_WT();
    window.setTitle("Discrete Wavelet-Transformation");
    window.pack();
    window.setVisible(true);
  }
}     // Ende Klasse Diskrete_WT**************************************

/*********************************************************************
*
*    Class Stream
*     Konstruktorvariablen
*     filename:   Name des Dateinamens
*     how:        0: lesen
*                 1: schreiben
*
*********************************************************************/
class Stream {

  private BufferedReader in;
  private PrintWriter out;
  private StringTokenizer ST;
  private String S;

  public static final int     READ = 0,
                        WRITE = 1;

  public Stream (String filename, int how) throws FileNotFoundException,
IOException {
      switch(how) {
        case READ: in = open(filename); break;
        case WRITE: out = create(filename); break;
      }
  }
```

```
  private BufferedReader open(String filename) throws FileNotFoundException
{
      return new BufferedReader(new FileReader(filename));
  }
  private PrintWriter create(String filename) throws IOException {
      return new PrintWriter(new FileWriter(filename));
  }
  public double readDouble () throws IOException {
    if (ST==null) refresh();
    while (true) {
      try {
        String item = ST.nextToken();
        return Double.valueOf(item.trim()).doubleValue();
      }
      catch (NoSuchElementException e1) {
        refresh ();
      }
      catch (NumberFormatException e2) {
        System.out.println("Error in number, try again.");
      }
    }
  }
  private void refresh () throws IOException {
    S = in.readLine ();
    if (S==null) throw new EOFException();
    ST = new StringTokenizer (S);
  }
  // Methode zuum schreiben in das File out
  public void println(double s) {
      out.println(s);
      out.flush();
  }
  //  Methode zum schließen des Files out
  public void close() throws IOException {
      if (out != null)
            out.close();
      if (in != null)
            in.close();
  }
  public void flush() {
      out.flush();
  }
} // Ende der Klasse Stream*****************************************

/*********************************************************************
*
*    Class ChartV1_0
*     chartname gibt den Namen der Grafik an
*     die Farben red, green, blue duerfen int-Werte von 0-255 annehmen
*     mode:       0: farbig
*                 1: Graustufen
*
*********************************************************************/
class ChartV1_0 extends JPanel {

  String    chartname;
  int             red, green, blue, mode;
  double    norm, minimum;
  double[]  polygon = new double[512];
  boolean   isPrinting = false;

  ChartV1_0 (String title, int r, int g, int b, int m) {
    chartname=title;
```

```
  red=r;
  green=g;
  blue=b;
  mode=m;
  for(int i=0; i<512; i++) polygon[i]=0.0;
}

public void setPlotData(double[] data) {     // uebergibt die Daten
  for(int i=0; i<512; i++) polygon[i]=data[i];
}
public void setNorm(double n, double m){     // setzt die min-/max-Werte
  norm=n;
  minimum=m;
}
public void setTitle(String t) {       // setzt den Titel neu
  chartname=t;
}
public void setColorMode(int c) {      // farbig oder Graustufen
  mode=c;
}
public void setPrinting(boolean p) {   // Objekt drucken true/false
  isPrinting=p;
}
public void plot(Graphics g) {         // zeichent das Signal
  g.setFont(new Font("Helvetica",Font.PLAIN,10));
  if (mode==0) g.setColor(Color.white);
  if (mode==1) g.setColor(Color.black);
  g.drawString(chartname,5,15);
  if (mode==0)g.setColor(Color.yellow);
  if (mode==1)g.setColor(Color.black);
  int xsize=getSize().width;
  int ysize=getSize().height;
  if (isPrinting) {        // im Druckermodus muss die Groesse der Grafik
    xsize=400;             // vorgegeben werden
    ysize=70;
  }
  int i;
  double max,min,dy,dx;
  max=-1.0e+14;min=-max;
  for(i=0;i<512;i++) {
     if(polygon[i]>max) {max=polygon[i];}
     if(polygon[i]<min) {min=polygon[i];}
  }
  dx=512.0/xsize;
  dy=ysize/(max-min);
  if (norm!=0.0) {
    dy=ysize/norm;
    min=minimum;
  }
  if((max-min)>1.0e-12) {
    for(i=0;i<xsize-1;i++) {
      g.drawLine(i,(int)((polygon[(int)(i*dx)]-min)*dy),
           i+1,(int)((polygon[(int)((i+1)*dx)]-min)*dy));
    }
  }
  else {
    g.drawLine(0,ysize/2,xsize-1,ysize/2);
  }
}
public void paint(Graphics g) {        // zeichnet den Hintergrund
  int x=getSize().width;
  int y=getSize().height;
  if (isPrinting) {
```

```
      x=400;
      y=70;
    }
    if (mode==0){
      g.setColor(new Color(red,green,blue));
      g.fillRect(0,0,x,y);
    }
    else if (mode==1){
      int maxInt=0;
      if (red >= maxInt)   maxInt=red;
      if (green >= maxInt) maxInt=green;
      if (blue >= maxInt)  maxInt=blue;
      g.setColor(new Color(maxInt,maxInt,maxInt));
      g.fillRect(0,0,x,y);
    }
    plot(g);
  }

} // Ende ChartV1_1*****************************************************

/*********************************************************************
*
*    Class ChartV1_1
*     erweitert die Klasse ChartV1_0.
*     Der Hintergrund der Grafik ist farblich unterteilt,
*     um die Skalen der diskreten Wavelet-Transformation
*     darstellen zu können. Dafuer wird die Methode paint
*     ueberschrieben.
*
*********************************************************************/
class CHARTV1_1 extends ChartV1_0 {
  CHARTV1_1 (String title, int r, int g, int b, int m) {
    super(title,r,g,b,m);
  }
  public void paint(Graphics g) {  // zeichnet den Hintergrund farblich
abgestuft
    int x=getSize().width;
    int y=getSize().height;
    if (isPrinting) {
      x=400;
      y=70;
    }
    if (mode==0){
      int i;
      for (i=1;i<=7;i++){
        g.setColor(new Color(40+i*30,0,0));
        g.fillRect((int)(x-x*(Math.pow(2,i)-
1)/Math.pow(2,i)),0,(int)(x*1/Math.pow(2,i))+1,y);
      }
      g.setColor(new Color(250,0,0));
      g.fillRect(0,0,(int)(x-x*(Math.pow(2,i-1)-1)/(Math.pow(2,i-1)))+1,y);
    }
    else if (mode==1){
      int i;
      for (i=1;i<=7;i++){
        g.setColor(new Color(100+i*20,100+i*20,100+i*20));
        g.fillRect((int)(x-x*(Math.pow(2,i)-
1)/Math.pow(2,i)),0,(int)(x*1/Math.pow(2,i))+1,y);
      }
      g.setColor(new Color(250,250,250));
      g.fillRect(0,0,(int)(x-x*(Math.pow(2,i-1)-1)/(Math.pow(2,i-1)))+1,y);
    }
    plot(g);
```

```
  }
} // Ende ChartV1_1*******************************************************

class daubechies {

double[] h =new double[8];
double[] g = new double[8];

/*************************************************************************/
/*    Daubechies smoothing filter H                                      */
/*             diffentation filter G                                     */
/*************************************************************************/



private void H(double[] sig,double[] Hsig,int length_sig,int length_h)
{int k,l;
 for(k=0;k<length_sig/2;k++)
     {Hsig[k]=0.0;
      for(l=0;l<length_h;l++)
          Hsig[k]+=h[l]*sig[(l+2*k)%length_sig];
      }
 }

private void G(double[] sig,double[] Gsig,int length_sig,int length_h)
{int k,l;
 for(k=0;k<length_sig/2;k++)
     {Gsig[k]=0.0;
      for(l=0;l<length_h;l++)
          Gsig[k]+=g[l]*sig[(l+2*k)%length_sig];
      }
 }


/*************************************************************************/
/*    Daubechies adjoint filters H*, G*                                  */
/*************************************************************************/

private void Had(double[] sig,double[] Hsig,int length_Hsig,int length_h)
{int k,l;

for(k=0;k<2*length_Hsig;k+=2)
   {sig[k]=0.0;
    for(l=0;l<length_h;l+=2)
      sig[k]+=h[l]*Hsig[((k-l)/2+length_Hsig)%length_Hsig];
    }
for(k=1;k<2*length_Hsig;k+=2)
   {sig[k]=0.0;
    for(l=1;l<length_h;l+=2)
       sig[k]+=h[l]*Hsig[((k-l)/2+length_Hsig)%length_Hsig];
   }
}

private void Gad(double[] sig,double[] Gsig,int length_Gsig,int length_g)
{int k,l;

for(k=0;k<2*length_Gsig;k+=2)
   {sig[k]=0.0;
    for(l=0;l<length_g;l+=2)
      sig[k]+=g[l]*Gsig[((k-l)/2+length_Gsig)%length_Gsig];
    }
for(k=1;k<2*length_Gsig;k+=2)
   {sig[k]=0.0;
```

```
    for(l=1;l<length_g;l+=2)
       sig[k]+=g[l]*Gsig[((k-l)/2+length_Gsig)%length_Gsig];
   }
}

/*************************************************************************/
/*    Initialization of Daubechies filters h,g                           */
/*                                                                       */
/*    order       order of the wavelet to be initialized                 */
/*    length_filt length of the corresponding filters                    */
/*    h,g         corresponding filter coefficients                      */
/*                                                                       */
/*    return:     0     O.K.                                             */
/*          -1    fiters of the desired order not implemented            */
/*************************************************************************/

public int daub_init(int order)
{int i,sig,l;
 switch(order)
  {case 1 : l=2;
            h[0]=1.0/Math.sqrt(2.0); h[1]=h[0];
          break;
  case 2: l=4;
          h[3]=(1.0-Math.sqrt(3.0))/(4.0*Math.sqrt(2.0));
        h[2]=(3.0-Math.sqrt(3.0))/(4.0*Math.sqrt(2.0));
        h[1]=(3.0+Math.sqrt(3.0))/(4.0*Math.sqrt(2.0));
        h[0]=(1.0+Math.sqrt(3.0))/(4.0*Math.sqrt(2.0));
        break;
 case 3: l=6;
          h[0]=0.3326705529500825;
       h[1]=0.8068915093110924;
       h[2]=0.4598775021184914;
       h[3]=-0.1350110200102546;
       h[4]=-0.0854412738820267;
       h[5]=0.0352262918857095;
       break;
 case 4: l=8;
          h[0]=0.2303778133088964;
       h[1]=0.7148465705529154;
       h[2]=0.6308807679398587;
       h[3]=-0.0279837694168599;
       h[4]=-0.1870348117190931;
       h[5]=0.0308413818355607;
         h[6]=0.0328830116668852;
       h[7]=-0.0105974017850690;
       break;
 default: return(0);
  }
sig=-1;
for(i=0;i<l;i++)
 {sig*=-1;g[i]=sig*h[l-1-i];}
return(l);
}


/*************************************************************************/
/*    Wavelt decomposition                                               */
/*                                                                       */
/*    sig         signal to be decomposed                                */
/*    length_sig  signal length                                          */
/*    length_filt filter_length (set by daub_init)                       */
/*    h           smoothing filter (set by daub_init)                    */
/*    g           differencing filter (set by daub_init)                 */
```

```
/*                                                          */
/*    output:           sig overwritten by the wavelet spectrum
*/
/*                decomp : 0  OK                            */
/*                         1    not enough memory          */
/*************************************************************************/

public int decomp(double[] sig,int length_sig,int length_filt)
{

 int i,j;

 /*   allocation of buffer                                      */

 double[] Hsig= new double[length_sig/2];
 double[] Gsig= new double[length_sig/2];

/*    Wavelet decomposition                                  */

 while(length_sig>=length_filt)
  {H(sig,Hsig,length_sig,length_filt);
   G(sig,Gsig,length_sig,length_filt);
   for(i=0;i<length_sig/2;i++)
     {sig[i]=Hsig[i];sig[length_sig/2+i]=Gsig[i];}
   length_sig/=2;
  }

/*    release memory                                         */


return(0);
}


/*************************************************************************/
/*    Wavelet reconstruction                                  */
/*                                                       */
/*    spec        spectrum used for reconstruction          */
/*    length_spec spectrum length                              */
/*    length_filt filter_length (set by daub_init)             */
/*    h           smoothing filter (set by daub_init)          */
/*    g           differencing filter (set by daub_init)             */
/*                                                          */
/*    output:           spec overwritten by the signal
*/
/*                reconst: 0  OK                            */
/*                         1    not enough memory          */
/*************************************************************************/

public int reconst(double[] spec,int length_spec,int length_filt)
{int l,length;

 double[] Hsig = new double[length_spec/2];
 double[] Gsig = new double[length_spec/2];
 double[] buff = new double[length_spec];


 switch(length_filt)
   {case 2 : length=1;
             break;
    case 4 : length=2;
          break;
    case 6 : length=4;
```

```
            break;
     case 8 : length=4;
            break;
    default: length=1;
    }


 while(length<=length_spec/2)
 {
  for(l=0;l<length;l++) {Hsig[l]=spec[l];Gsig[l]=spec[length+l];}
  Had(buff,Hsig,length,length_filt);
  Gad(spec,Gsig,length,length_filt);
  length*=2;
  for(l=0;l<length;l++) spec[l]+=buff[l];
 }

return(0);
}
}
```

Anhang C

Das Java-Applet zur kontinuierlichen Wavelet-Transformation

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class CWTApplet  extends     Applet
                        implements  ActionListener, ItemListener,
                                    AdjustmentListener,
                                    MouseMotionListener {

  // GUI-Variablen************************
  private Panel   leftPanel, rightPanel;
  private Panel   waveletPanel, scalePanel, morletPanel, gridPanel,
                  dataPanel, noisePanel, mousePanel, colorCodePanel;
  private Choice  selectData, chooseWavelet, selectScale;
  private Scrollbar     morletSlider, noise;
  private TextField     morletTextField, mouseTextField, noiseTextField;
  private Label   inputLabel, waveletLabel, scaleLabel,
                  morletLabel, mouseLabel,
                  noiseLabel, colorCodeLabel, spacer;
  private Button  start, add;
  private CheckboxGroup colorCodeGroup;
  private Checkbox      addGrid;
  private Checkbox[]    colorCheck = new Checkbox[3];

  boolean gridAdded=false;
  boolean xAxisAdded=false;
  boolean yAxisAdded=true;
  String wav;
  int nos=128;
  int omegaInt;
  double omega0 = 1.0;
  cwt1 CWT;

  ChartNew1 signalChart   = new ChartNew1("blue", "Signal", gridAdded);
  ColorCodeNew2 spectrumChart = new ColorCodeNew2(128,512,"blabla",
                                                  gridAdded);
  XAxis xAxis = new XAxis();
  YAxis yAxis = new YAxis(128);

  int colorCodePicked=1;
  boolean noiseAdded = false;

  double[] signal = new double[512];
  double[] rauschen = new double[512];
  double[] sigPlusNoise = new double[512];
  double[][] cwtspec = new double[128][512];

  private String[] dataItems = {"none", "Sinus", "Sinus 2Frequ",
                       "sin + cos",  "t²*sin(wt)",
                        "sin(wt²)", "Sinus + Impuls", "Sägezahn",};
  static final int none1 =0, sinus=1, sin2f=2, sincos=3, t2sin=4,
                  sint2=5, sinimp=6, saege=7;
  private String[] waveletItems = {"none", "Haar", "Mexican Hat",
                                   "Morlet"};
  static final int none=0, haar=1, mh=2, morlet=3;

  public void init() {
    initialize();
```

```
} // Ende init

// Behandler*******************************

public void actionPerformed (ActionEvent e) {
  if (e.getSource() == add) {
    for (int i=0; i<502; i++) {
    rauschen[i] = ((double)noise.getValue()/10) * Math.random();
    sigPlusNoise[i] = rauschen[i] + signal[i];
    }
    drawSignal(sigPlusNoise);
    noiseAdded = true;
  }
  else if (e.getSource() == start) {
    if (noiseAdded) signal = sigPlusNoise;
    if(wav.equals("Morlet")) wav="morlet";
    if(wav.equals("Mexican Hat")) wav="mexhat";
    if(wav.equals("Haar")) wav="haar";
    if(wav!="none") {
      CWT=new cwt1(512,nos);
      CWT.run(signal,wav,omega0,cwtspec);
      spectrumChart.setMode(colorCodePicked);
      spectrumChart.setScales(nos);
      spectrumChart.setData(cwtspec);
      spectrumChart.paint(spectrumChart.getGraphics());
    Toolkit.getDefaultToolkit().beep();
    }
  }
}

int dataPicked, waveletPicked;
public void itemStateChanged (ItemEvent e) {
  int index=3;
  Object picked = e.getSource();
  if (picked == selectData) {
  dataPicked = ((Choice) picked).getSelectedIndex();
  noiseAdded = false;
  switch (dataPicked) {
    case none1 :      for (int i=0; i<512; i++) signal[i]=0;
                      dataLoaded(false);
                break;
    case sinus :      signal = DataSet2.sinus();
                      dataLoaded(true);
                break;
    case sin2f :      signal = DataSet2.sin2freq();
                      dataLoaded(true);
                break;
    case sincos :     signal = DataSet2.sincos();
                      dataLoaded(true);
                break;
    case t2sin :      signal = DataSet2.sinPo2();
                      dataLoaded(true);
                break;
    case sint2 :      signal = DataSet2.sinPow2();
                      dataLoaded(true);
                break;
    case sinimp :     signal = DataSet2.sinImpuls();
                      dataLoaded(true);
                break;
    case saege :      signal = DataSet2.saege();
                      dataLoaded(true);
                break;
  } // Ende Switch
```

```
    }
    else if (picked == chooseWavelet) {
      waveletPicked = ((Choice) picked).getSelectedIndex();
      if (noiseAdded) signal = sigPlusNoise;
      wav = ((Choice) picked).getSelectedItem();
      if (wav == "Morlet") morletSlider.setEnabled(true);
      else if (wav != "Morlet") morletSlider.setEnabled(false);
    }
    else if (picked == selectScale) {
      nos = Integer.parseInt(((Choice) picked).getSelectedItem());
    }

    else if (picked == addGrid) {
      gridAdded=true;
      signalChart.setGrid(gridAdded);
      signalChart.paint(signalChart.getGraphics());
      spectrumChart.setGrid(gridAdded);
      spectrumChart.paint(spectrumChart.getGraphics());
      index=2;
    }
    if (e.getStateChange() == ItemEvent.DESELECTED) {
      if (index == 2) {
      gridAdded=false;
      signalChart.setGrid(gridAdded);
      signalChart.paint(signalChart.getGraphics());
        spectrumChart.setGrid(gridAdded);
        spectrumChart.paint(spectrumChart.getGraphics());
      }
    }
    else for (int i=0; i<3; i++) {
      if (picked == colorCheck[i]) {
      colorCodePicked = i+1;

      spectrumChart.setMode(colorCodePicked);
      spectrumChart.setData(cwtspec);
      spectrumChart.paint(spectrumChart.getGraphics());}
    }
  }
  public void adjustmentValueChanged(AdjustmentEvent e) {
    Object picked = e.getSource();
    if (picked == noise) {
      noiseTextField.setText("   S/R :    10/" +
String.valueOf(noise.getValue()));
    }
    else if (picked == morletSlider) {
      omegaInt = (int)morletSlider.getValue();
      omega0 = (double)omegaInt/10;
      morletTextField.setText(String.valueOf(omega0));
    }
  }
  public void mouseMoved(MouseEvent e) {
    int x = spectrumChart.getSize().width;
    int y = spectrumChart.getSize().height;
    double xs = (double)e.getX() / (double) x;
    double ys = (double)e.getY() / (double) y;
    mouseTextField.setText(" (" + (int)(512*xs) + " /" + (int)(100*ys) +
")");
  }
  public void mouseDragged(MouseEvent e) {
  }
// Methoden************************************************
  public static void main(String args[]) {
    Frame f = new Frame("GridBag Layout Example");
```

```
    CWA8 ex = new CWA8();
    ex.init();
    f.add("Center", ex);
    f.pack();
    f.setSize(f.getPreferredSize());
    f.show();
  }
  public void initialize() {
    makeLeftPanel();
    makeRightPanel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    setLayout(gridbag);
    c.fill = GridBagConstraints.BOTH;
    // RightPanel**********
    c.gridx = 3;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 3;
    c.weightx = 0.0;
    c.weighty = 0.0;
    gridbag.setConstraints(rightPanel, c);
    add(leftPanel);
    // LeftPanel**********
    c.gridx = 0;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 3;
    gridbag.setConstraints(leftPanel, c);
    add(rightPanel);
    // Signal**********
    c.gridx = 2;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 1.0;
    c.weighty = 1.0;
    c.ipadx = 350;
    c.ipady = 80;
    gridbag.setConstraints(signalChart, c);
    add(signalChart);
    // Spectrum**********
    c.gridx = 2;
    c.gridy = 2;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 1.0;
    c.weighty = 1.0;
    c.ipadx = 350;
    c.ipady = 220;
    gridbag.setConstraints(spectrumChart, c);
    add(spectrumChart);
    spectrumChart.addMouseMotionListener(this);
    // X-Achse**********
    c.gridx = 2;
    c.gridy = 1;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0.0;
    c.weighty = 0.0;
    c.ipadx = 0;
    c.ipady = 8;
    gridbag.setConstraints(xAxis, c);
```

```
  add(xAxis);
  xAxis.setVisible(true);
  // Y-Achse**********
  c.gridx = 1;
  c.gridy = 2;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 0.0;
  c.weighty = 0.0;
  c.ipadx = 12;
  c.ipady = 0;
  gridbag.setConstraints(yAxis, c);
  add(yAxis);
  yAxis.setVisible(true);
  setSize(700, 400);
}
public void drawSignal (double[] sig) {
  signalChart.setPlotData(sig);
  signalChart.paint(signalChart.getGraphics());
}
public void dataLoaded (boolean b) {
  drawSignal(signal);
  noise.setEnabled(b);
  add.setEnabled(b);
  chooseWavelet.setEnabled(b);
  selectScale.setEnabled(b);
  morletSlider.setEnabled(b);
  start.setEnabled(b);
  for (int i=0; i<3; i++) colorCheck[i].setEnabled(b);
  noise.setValue(0);
  noiseTextField.setText("   S/R :    10/0");
}
void makeLeftPanel() {
  leftPanel = new Panel();
  leftPanel.setLayout (new GridLayout(3,1,5,15));
    dataPanel = new Panel();
    dataPanel.setLayout(new GridLayout(6,1,1,1));
      inputLabel = new Label ("Choose Signal ");
      selectData = new Choice ();
    for (int i=0; i<dataItems.length; i++) {
        selectData.addItem(dataItems[i]);
    }
      selectData.addItemListener(this);
    dataPanel.add(inputLabel);
    dataPanel.add(selectData);
      noiseLabel = new Label ("Add Noise");
    noise = new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,11);
    noise.addAdjustmentListener(this);
    noise.setEnabled(false);
    noiseTextField = new TextField(" ",7);
    noiseTextField.setEditable(false);
    add = new Button("Add");
    add.setEnabled(false);
    add.addActionListener(this);
    dataPanel.add(noiseLabel);
    dataPanel.add(noise);
    dataPanel.add(noiseTextField);
    dataPanel.add(add);
    waveletPanel = new Panel();
    waveletPanel.setLayout(new GridLayout(6,1,1,1));
    waveletLabel = new Label ("Choose Wavelet");
    chooseWavelet = new Choice();
    for (int i=0; i<waveletItems.length; i++) {
```

```
          chooseWavelet.addItem(waveletItems[i]);
      }
      chooseWavelet.setEnabled(false);
      chooseWavelet.addItemListener(this);
      waveletPanel.add(new Label(""));
      waveletPanel.add(waveletLabel);
      waveletPanel.add(chooseWavelet);
      morletLabel = new Label("Parameter for Morlet");
      morletSlider = new Scrollbar(Scrollbar.HORIZONTAL,10,1,10,101);
      morletSlider.addAdjustmentListener(this);
      morletSlider.setEnabled(false);
      morletTextField = new TextField(String.valueOf(omega0));
      morletTextField.setEditable(false);
      waveletPanel.add(morletLabel);
      waveletPanel.add(morletSlider);
      waveletPanel.add(morletTextField);

      scalePanel = new Panel();
      scalePanel.setLayout(new GridLayout(6,1,1,1));
        scaleLabel = new Label("Number of Scales");
      selectScale = new Choice();
      for (int i=7; i>=0; i--) {
        int j = (int) Math.pow(2,i);
        selectScale.addItem(String.valueOf(j));
      }
      selectScale.setEnabled(false);
      selectScale.addItemListener(this);
      scalePanel.add(scaleLabel);
      scalePanel.add(selectScale);
      scalePanel.add(new Label(""));
      scalePanel.add(new Label(""));
      scalePanel.add(new Label(""));
        start = new Button("Start Computation");
        start.setEnabled(false);
        start.addActionListener(this);
      scalePanel.add(start);
    leftPanel.add(dataPanel);
    leftPanel.add(waveletPanel);
    leftPanel.add(scalePanel);
  }
  void makeRightPanel() {
    rightPanel = new Panel();
    rightPanel.setLayout(new GridLayout(3,1,5,15));
      colorCodePanel = new Panel();
      colorCodePanel.setLayout (new GridLayout(6,1,1,1));
        colorCodeLabel = new Label ("Stretch Color-Code");
        colorCodeGroup = new CheckboxGroup();
        colorCheck[0] = new Checkbox("Linear", true, colorCodeGroup);
        colorCheck[1] = new Checkbox("Exponentiell", false,
colorCodeGroup);
        colorCheck[2] = new Checkbox("Logarithmisch", false,
colorCodeGroup);
      colorCodePanel.add(colorCodeLabel);
      for (int i=0; i<3; i++) {
      colorCheck[i].addItemListener(this);
      colorCodePanel.add(colorCheck[i]);
        colorCheck[i].setEnabled(false);
      }
      gridPanel = new Panel();
      gridPanel.setLayout (new GridLayout(6,1,1,1));
      addGrid = new Checkbox ("Add Grid");
      addGrid.addItemListener(this);
      gridPanel.add(new Label(""));
```

```
      gridPanel.add(addGrid);
      mousePanel = new Panel();
      mousePanel.setLayout (new GridLayout(6,1,1,1));
        mouseLabel = new Label ("Mouse Motion ");
        mouseTextField = new TextField (" ",7);
        mouseTextField.setEnabled(true);
        mouseTextField.setEditable(false);
      mousePanel.add(new Label(""));
      mousePanel.add(new Label(""));
      mousePanel.add(new Label(""));
      mousePanel.add(new Label(""));
      mousePanel.add(mouseLabel);
      mousePanel.add(mouseTextField);

    rightPanel.add(colorCodePanel);
    rightPanel.add(gridPanel);
    rightPanel.add(mousePanel);
  } // Ende MakeControlPanel
} // Ende Klasse Applet


/************************************************
*                                       *
* Klasse Chart1                               *
*                                       *
************************************************/
class ChartNew1 extends Panel {

String paint,chartname;
boolean gridBoolean;
int i,n;
double[] polygon = new double[512];

public ChartNew1(String color,String title, boolean gA) {
int i;
paint =color;
chartname=title;
gridBoolean = gA;
for(i=0;i<512;i++) polygon[i]=0.0;

} //ends constructor of Chart

public void setPlotData(double[] data) {
int i;
for(i=0;i<512;i++) polygon[i]=data[i];

}

public void setGrid(boolean g) {
  gridBoolean = g;
}

public void plot(Graphics g) {
int xsize=getSize().width;
int ysize=getSize().height;

int i;
double max,min,dy,dx;
max=-1.0e+14;min=-max;
for(i=0;i<512;i++)
   {if(polygon[i]>max) {max=polygon[i];}
    if(polygon[i]<min) {min=polygon[i];}
   }
```

```
dx=512.0/xsize;
dy=ysize/(max-min);

if((max-min)>1.0e-12)
   {for(i=0;i<xsize-1;i++)
    g.drawLine(i,(int)((polygon[(int)(i*dx)]-min)*dy),
             i+1,(int)((polygon[(int)((i+1)*dx)]-min)*dy));
    }
else {g.drawLine(0,ysize/2,xsize-1,ysize/2);}
if (gridBoolean) {
  for (int j=50; j<512; j+=50) {
     for (int k=0; k<xsize; k+=10) g.drawLine(xsize*j/512, k, xsize*j/512,
k+2);
  }
} else {}
}

public void paint(Graphics g) {

if(paint=="black") g.setColor(new Color (70,0,0));
if(paint=="blue") g.setColor(new Color (0,0,70));
int x=getSize().width;
int y=getSize().height;
g.fillRect(0,0,x,y);
g.setFont(new Font("Helvetica",Font.PLAIN,10));
g.setColor(Color.white);
g.drawString(chartname,5,15);
plot(g);
}
} // ends Chart1

/************************************************
*                                      *
* Klasse XAxis                              *
*                                      *
************************************************/
class XAxis extends Panel {
  double xc;
  public XAxis() {
  } //ends constructor of XAxis
  public void setCursorX(double xc1) {
    xc=xc1;
  }
  public void paint(Graphics g) {
    int x=getSize().width;
    int y=getSize().height;
    g.setColor(new Color (220,220,220));
    g.fillRect(0,0,x,y);
    g.setColor(Color.black);
    g.drawLine(0, (int)y/6, x, (int)y/6);
    for (int i=0; i<512; i += 50) g.drawLine(x*i/512, 0, x*i/512,
(int)y/6);
    g.setFont(new Font("Helvetica",Font.PLAIN,9));
    g.drawString("0", 0, (int)y*8/10);
    for (int i=50; i<512; i += 50) g.drawString(String.valueOf(i),
((int)x*i/512)-6, (int)y*8/10);
    for (int i=0; i<512; i += 50) g.drawLine(x*i/512, y, x*i/512,
(int)y*7/8);
    g.setColor(Color.red);
    g.drawLine((int)xc,0,(int)xc,y);
  }
} // ends XAxis
/************************************************
```

```
*                                       *
* Klasse YAxis                                *
*                                       *
*****************************************************/
class YAxis extends Panel {
  int numbOfScale;
  double yc;
  public YAxis(int nos) {
    numbOfScale=nos;
  }
  public void setCursorY(double yc1) {
    yc=yc1;
  }
  public void setScales(int nos) {
    numbOfScale=nos;
  }
  public void paint(Graphics g) {
    int x=getSize().width;
    int y=getSize().height;
    g.setColor(new Color (220,220,220));
    g.fillRect(0,0,x,y);
    g.setColor(Color.black);
    g.setFont(new Font("Helvetica",Font.PLAIN,9));
    g.drawLine((int)x*6/7, 0, (int)x*6/7, y);
    if (numbOfScale>32) {
      for (int i=10; i<numbOfScale; i+=10) {
      g.drawLine((int)x*6/7, y*i/numbOfScale, x, y*i/numbOfScale);
      g.drawString(String.valueOf(i), 0, y*i/numbOfScale+4);
      }
    }
    else if (numbOfScale<=32 & numbOfScale>16) {
      for (int i=5; i<numbOfScale; i+=5) {
      g.drawLine((int)x*6/7, y*i/numbOfScale, x, y*i/numbOfScale);
      g.drawString(String.valueOf(i), 0, y*i/numbOfScale+4);
      }
    }
    else if (numbOfScale<=16) {
      for (int i=1; i<numbOfScale; i++) {
      g.drawLine((int)x*6/7, y*i/numbOfScale, x, y*i/numbOfScale);
      g.drawString(String.valueOf(i), 0, y*i/numbOfScale+4);
      }
    }
    g.setColor(Color.red);
    g.drawLine(0,(int)yc,x,(int)yc);
  }
} // ends YAxis

/*************************************************
*                                       *
* Klasse DataSet                        *
*                                       *
*************************************************/
class DataSet2 {

  static double[] data = new double[512];

  DataSet2() {
  }

  static double[] sinus() {
    for (int i=0; i<512; i++) {
      data[i] = Math.sin(((double)i)/10);
    }
```

```
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}

static double[] sin2freq() {
  int i;
  for (i=0; i<256; i++) {
    double x = ((double)i)/10;
    data[i] = Math.sin(x);
    if (x>8*Math.PI) break;
  }
  for (i=i; i<512; i++) {
    double x = ((double)i)/5;
    data[i] = Math.sin(x);
  }
  double r = range(data);
  for (int j=0; j<512; j++) {
    data[j] = data[j]/r;
  }
 return data;
}

static double[] sincos() {
  for (int i=0; i<512; i++) {
    data[i] = (Math.sin(((double)i)/10)) + 0.7*(Math.cos(((double)i)/4));
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}

static double[] sinPow2() {
  for (int i=0; i<512; i++) {
    data[i] = Math.sin(((double)i)/(0.005));
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}

static double[] sinPo2() {
  for (int i=0; i<512; i++) {
    data[i] = (Math.pow(i,2))*Math.sin(((double)i)/4);
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}

static double[] sinImpuls() {
  for (int i=0; i<512; i++) {
    data[i] = (Math.sin(((double)i)/10)) + (Math.cos(((double)i)/4));
    if (i==155) data[i] = -10;
    if (i==300) data[i] = 10;
```

```
    }
    double r = range(data);
    for (int i=0; i<512; i++) {
      data[i] = data[i]/r;
    }
   return data;
  }
  static double[] saege() {
    for (int i=0; i<512; i+=32) {
      for (int j=0; j<=16; j++) {
        data[i+j] = 2*j;
      }
      for (int j=17; j<32; j++) {
        data[i+j] = 68+(-2)*j;
      }
    }
    double r = range(data);
    for (int i=0; i<512; i++) {
      data[i] = data[i]/r;
    }
   return data;
  }

static double range(double[] d) {
double max, min, range;
max=-1.0e+14; min=-max;
for (int i=0; i<512; i++) {
  if(d[i]>max) max=d[i];
  if(d[i]<min) min=d[i];
}
range=max-min;
return range;
}

} // Ende Klasse DataSet

/************************************************
*                                      *
* Klasse ColorCodeNew                        *
*                                      *
************************************************/
class ColorCodeNew2 extends Panel {

int Nrow,Ncol;
String Title;
double[][] data;
int coding;
boolean gridAdded;

public ColorCodeNew2(int rows, int cols,String NameOfPlot, boolean gA){
//constructor
Nrow=rows;Ncol=cols;Title=NameOfPlot;
gridAdded = gA;
data = new double[Nrow][Ncol];
}

public void setGrid (boolean g) {gridAdded = g;}

public void setScales(int nos) {
Nrow=nos;
data = new double[Nrow][Ncol];
}
```

```
public void setMode(int Mode){
coding=Mode;
}

public void setData(double[][] daten){
int i,j;
for(i=0;i<Nrow;i++)
  for(j=0;j<Ncol;j++) data[i][Ncol-1-j]=daten[i][j];
}

public void plot(Graphics g) {
int xsize=getSize().width;
int ysize=getSize().height;

int i,j,color,red,green,blue;
double max,min,dy,dx;
max=-1.0e+14;min=-max;

for(i=0;i<Nrow;i++)  // determining maximum and minimum
   for(j=0;j<Ncol;j++)
      {if(data[i][j]>max) max=data[i][j];
       if(data[i][j]<min) min=data[i][j];
      }



for(i=0;i<Nrow;i++)
  for(j=0;j<Ncol;j++)
     {switch(coding)
      {case 1:
        color=(int)(255*(data[i][j]-min)/(max-min)); //color number
        break;
       case 2: color=(int)(255*(Math.exp(3.0*(data[i][j]-min)/(max-min))-
1.0)
                           /(Math.exp(3.0)-1));
               break;
       case 3: color=(int)(255*(Math.log(3.0*(data[i][j]-min)/(max-
min)+1.0)
                           /Math.log(4.0)));
       break;
       default: color=0;
      }


      red=255-color;
      green=(int)(255-2.0*Math.abs(128-color));
      if(green<0) green=0;
      blue=color;


      g.setColor(new Color(red,green,blue)); // color set
      g.fillRect((int)((j*xsize*1.0)/(Ncol*1.0)),
                 (int)((i*ysize*1.0)/(Nrow*1.0)),
                 (int)((xsize*1.0)/(Ncol*1.0))+1,
                 (int)((ysize*1.0)/(Nrow*1.0))+1);
      }

if (gridAdded) {
  g.setColor(Color.black);
  for (int l=50; l<512; l+=50) {
     for (int m=0; m<ysize; m+=10) g.drawLine(xsize*l/512, m, xsize*l/512,
m+2);
  }
```

```
    if (Nrow>32) {
      for (int n=10; n<Nrow; n+=10) {
      for (int m=0; m<xsize; m+=10) g.drawLine(m, ysize*n/Nrow, m+2,
ysize*n/Nrow);
      }
    }
    else if (Nrow<=32 & Nrow>16) {
      for (int n=5; n<Nrow; n+=5) {
      for (int m=0; m<xsize; m+=10) g.drawLine(m, ysize*n/Nrow, m+2,
ysize*n/Nrow);
      }
    }
    else if (Nrow<=16) {
      for (int n=1; n<Nrow; n++) {
      for (int m=0; m<xsize; m+=10) g.drawLine(m, ysize*n/Nrow, m+2,
ysize*n/Nrow);
      }
    }
}

}// ends plot

public void paint(Graphics g) {
plot(g);
}


}// ends ColorCode
```

Anhang D

Das Java-Applet zur diskreten Wavelet-Transformation

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class DWTApplet  extends     Applet
                        implements  ActionListener, ItemListener,
                                    AdjustmentListener,
                                    MouseMotionListener {

  // GUI-Variablen
  private Panel   inputPanel, controlPanel, mousePanel;
  private Panel   waveletChoicePanel, scalePanel,
                  tresholdPanel, buttonPanel,
                  dataPanel, noisePanel;
  private Choice  selectData, chooseWavelet, selectScale;
  private Scrollbar     treshold, noise;
  private TextField     treshTextField, mouseTextField, noiseTextField;
  private Label         inputLabel, waveletLabel,
                        scaleLabel, tresholdLabel, mouseLabel,
                        noiseLabel;
  private Button        apply, undo, add;

  Zeichnen4 signalChart   = new Zeichnen4("blue", "Signal");
  Zeichnen4 spectrumChart = new Zeichnen4("red", "Spectrum");
  double s=0;
  int length_filt, level=1;
  int fps;
  daubechies wavbox = new daubechies();
  boolean noiseAdded = false;

  double[] sigback = new double[512];
  double[] signal = new double[512];
  double[] spectrum = new double[512];
  double[] specback = new double[512];
  double[] rauschen = new double[512];
  double[] sigPlusNoise = new double[512];

  private String[] dataItems = {"Sinus", "Sinus 2Frequ", "sin + cos",
      "t²*sin", "sin(t²)", "Sinus + Impuls", "Sägezahn",};
  static final int sinus=0, sin2f=1, sincos=2, t2sin=3, sint2=4,
      sinimp=5, saege=6;
  private String[] waveletItems = {"Haar", "Daubechies4", "Daubechies6",
      "Daubechies8"};
  static final int haar=0, d4=1, d6=2, d8=3;

  public void init() {
    makeInputPanel();

    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    setLayout(gridbag);
    c.fill = GridBagConstraints.BOTH;
    // InputPanel**********
    c.gridx = 0;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 2;
    c.weightx = 0.0;
```

```
  c.weighty = 0.0;
  gridbag.setConstraints(inputPanel, c);
  add(inputPanel);
  // Signal**********
  c.gridx = 1;
  c.gridy = 0;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 1.0;
  c.weighty = 1.0;
  c.ipadx = 350;
  c.ipady = 150;
  gridbag.setConstraints(signalChart, c);
  add(signalChart);
  // Spectrum**********
  c.gridx = 1;
  c.gridy = 1;
  c.gridwidth = 1;
  c.gridheight = 1;
  c.weightx = 1.7;
  c.weighty = 1.0;
  c.ipadx = 350;
  c.ipady = 150;
  gridbag.setConstraints(spectrumChart, c);
  add(spectrumChart);
  spectrumChart.addMouseMotionListener(this);
  setBackground(new Color (230,230,230));
  setSize(700, 400);
   } // Ende init

// Behandler*******************************
public void actionPerformed (ActionEvent e) {
  if (e.getSource() == add) {
    for (int i=0; i<512; i++) {
    rauschen[i] = ((double)noise.getValue()/10) * Math.random();
    sigPlusNoise[i] = rauschen[i] + sigback[i];
    }
    drawSignal(sigPlusNoise);
    noiseAdded = true;
  }
  else if (e.getSource() == apply) {
    wipeout(level, fps);
    plotting();
    undo.setEnabled(true);
  }
  else if (e.getSource() == undo) {
    restore(level);
    plotting();
  }
}
int dataPicked, waveletPicked;
public void itemStateChanged (ItemEvent e) {
  Object picked = e.getSource();
  if (picked == selectData) {
  dataPicked = ((Choice) picked).getSelectedIndex();
  noiseAdded = false;
  switch (dataPicked) {
    case sinus :      sigback = DataSet1.sinus();
                      dataLoaded();
                break;
    case sin2f :      sigback = DataSet1.sin2freq();
                      dataLoaded();
                break;
```

```
      case sincos :     sigback = DataSet1.sincos();
                        dataLoaded();
                  break;
      case t2sin :      sigback = DataSet1.sinPo2();
                        dataLoaded();
                  break;
      case sint2 :      sigback = DataSet1.sinPow2();
                        dataLoaded();
                  break;
      case sinimp :     sigback = DataSet1.sinImpuls();
                        dataLoaded();
                  break;
      case saege :      sigback = DataSet1.saege();
                        dataLoaded();
                  break;
    }
    }
    else if (picked == chooseWavelet) {
    waveletPicked = ((Choice) picked).getSelectedIndex();
    if (noiseAdded) sigback = sigPlusNoise;
      switch (waveletPicked) {
        case haar :     processing(1);
                  break;
        case d4 : processing(2);
                  break;
        case d6 : processing(3);
                  break;
        case d8 : processing(4);
                  break;
      }
    }
    else if (picked == selectScale) {
    level = ((Choice) picked).getSelectedIndex() + 1;
    }
  }
  public void adjustmentValueChanged(AdjustmentEvent e) {
    Object picked = e.getSource();
    if (picked == noise) {
      noiseTextField.setText("   S/R :    10/" +
String.valueOf(noise.getValue()));
    }
    else if (picked == treshold) {
      fps = (int)treshold.getValue();
      treshTextField.setText("     " + fps);
    }
  }

  public void mouseMoved(MouseEvent e) {
    int x = spectrumChart.getSize().width;
    int y = spectrumChart.getSize().height;
    double xs = (double)e.getX() / (double) x;
    double ys = (double)e.getY() / (double) y;
    mouseTextField.setEnabled(true);
    mouseTextField.setText(" (" + (int)(512*xs) + " /" + (int)(100*ys) +
")");
    mouseTextField.setEnabled(true);
  }

  public void mouseDragged(MouseEvent e) {
  }

  // Methoden*************************************************
  public static void main(String args[]) {
```

```
  Frame f = new Frame("GridBag Layout Example");
  GridBagEx1 ex1 = new GridBagEx1();

  ex1.init();

  f.add("Center", ex1);
  f.pack();
  f.setSize(f.getPreferredSize());
  f.show();
}
public void drawSignal (double[] sig) {
  signalChart.setPlotData(sig);
  signalChart.paint(signalChart.getGraphics());
}
public void dataLoaded () {
  drawSignal(sigback);
  noise.setEnabled(true);
  add.setEnabled(true);
  chooseWavelet.setEnabled(true);
  selectScale.setEnabled(true);
  treshold.setEnabled(true);
  apply.setEnabled(true);
  undo.setEnabled(true);
  noise.setValue(0);
  noiseTextField.setText("   S/R :    10/0");
}
void makeInputPanel() {
  inputPanel = new Panel();
  inputPanel.setLayout (new GridLayout(16,1,0,0));
      inputLabel = new Label ("Choose Signal ");
      selectData = new Choice ();
    for (int i=0; i<dataItems.length; i++) {
        selectData.addItem(dataItems[i]);
    }
      selectData.addItemListener(this);
    inputPanel.add(inputLabel);
    inputPanel.add(selectData);
      noiseLabel = new Label ("Add Noise");
    noise = new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,11);
    noise.addAdjustmentListener(this);
    noise.setEnabled(false);
    noiseTextField = new TextField(" ",7);
    noiseTextField.setEditable(false);
    add = new Button("Add");
    add.setEnabled(false);
    add.addActionListener(this);
    inputPanel.add(noiseLabel);
    inputPanel.add(noise);
    inputPanel.add(noiseTextField);
    inputPanel.add(add);
    waveletLabel = new Label ("Choose Wavelet");
    chooseWavelet = new Choice();
    for (int i=0; i<waveletItems.length; i++) {
        chooseWavelet.addItem(waveletItems[i]);
    }
    chooseWavelet.setEnabled(false);
    chooseWavelet.addItemListener(this);
    inputPanel.add(waveletLabel);
    inputPanel.add(chooseWavelet);
    scaleLabel = new Label("Select Scale");
    scalePanel = new Panel();
    selectScale = new Choice();
    for (int i=1; i<10; i++) {
```

```
        selectScale.addItem(String.valueOf(i));
      }
      selectScale.setEnabled(false);
      selectScale.addItemListener(this);
      inputPanel.add(scaleLabel);
      inputPanel.add(selectScale);
      tresholdLabel = new Label("Set Treshold");
      treshold = new Scrollbar(Scrollbar.HORIZONTAL,100,1,0,101);
      treshold.addAdjustmentListener(this);
      treshold.setEnabled(false);
      treshTextField = new TextField(" ");
      treshTextField.setEditable(false);
      inputPanel.add(tresholdLabel);
      inputPanel.add(treshold);
      inputPanel.add(treshTextField);
    buttonPanel = new Panel();
    buttonPanel.setLayout (new FlowLayout(FlowLayout.CENTER,1,1));
      apply = new Button("Apply");
      apply.setEnabled(false);
      apply.addActionListener(this);
      undo = new Button("Undo");
      undo.setEnabled(false);
      undo.addActionListener(this);
    buttonPanel.add(apply);
    buttonPanel.add(undo);
     inputPanel.add(buttonPanel);
      mouseLabel = new Label ("Mouse Motion ");
      mouseTextField = new TextField (" ",7);
      mouseTextField.setEnabled(true);
      mouseTextField.setEditable(false);
    inputPanel.add(mouseLabel);
    inputPanel.add(mouseTextField);
  }

private void wipeout(int level,int tresh){
int i,start;
double norm;
start=1;
for(i=0;i<9-level;i++) start*=2;
norm=0.0;
for(i=start;i<2*start;i++) norm+=spectrum[i]*spectrum[i];
norm=Math.sqrt(norm/start);

for(i=start;i<2*start;i++)
  {specback[i]=spectrum[i];
   if(Math.abs(spectrum[i])<norm*tresh*0.05) {spectrum[i]=0.0;}
  }
}
private void restore(int level) {
int i,start;
start=1;
for(i=0;i<9-level;i++) start*=2;
for(i=start;i<2*start;i++) spectrum[i]=specback[i];
}
private void processing(int order) {
int i;
length_filt=wavbox.daub_init(order);
for(i=0;i<512;i++) spectrum[i]=sigback[i];
i=wavbox.decomp(spectrum,512,length_filt);
spectrumChart.setPlotData(spectrum);
spectrumChart.paint(spectrumChart.getGraphics());
signalChart.setPlotData(sigback);
signalChart.paint(signalChart.getGraphics());
```

```
}
private void plotting() {
int i;
spectrumChart.setPlotData(spectrum);
spectrumChart.paint(spectrumChart.getGraphics());
for(i=0;i<512;i++) signal[i]=spectrum[i];
i=wavbox.reconst(signal,512,length_filt);
signalChart.setPlotData(signal);
signalChart.paint(signalChart.getGraphics());
}
} // Ende Klasse Applet



/***********************************************
*                                     *
* Klasse Zeichnen4                          *
*                                     *
***********************************************/

class Zeichnen4 extends Panel {

  String paint,chartname;
  double[] polygon = new double[512];

  Zeichnen4 (String color,String title) {
    paint =color;
    chartname=title;
    for(int i=0; i<512; i++) polygon[i]=0.0;
  } // Ende Konstruktor Zeichenen*******************************

  public void setPlotData(double[] data) {
    for(int i=0; i<512; i++) polygon[i]=data[i];
  }

  public void plot(Graphics g) {
      int xsize=getSize().width;
      int ysize=getSize().height;

    int i;
    double max,min,dy,dx;
    max=-1.0e+14;min=-max;
    for(i=0;i<512;i++) {
       if(polygon[i]>max) {max=polygon[i];}
       if(polygon[i]<min) {min=polygon[i];}
    }
    dx=512.0/xsize;
    dy=ysize/(max-min);

    if((max-min)>1.0e-12) {
      for(i=0;i<xsize-1;i++) {
        g.drawLine(i,(int)((polygon[(int)(i*dx)]-min)*dy),
             i+1,(int)((polygon[(int)((i+1)*dx)]-min)*dy));
      }
    }
    else {
      g.drawLine(0,ysize/2,xsize-1,ysize/2);
    }
  }

  public void paint(Graphics g) {
    int x=getSize().width;
    int y=getSize().height;
```

```
    if (paint=="blue") {
      g.setColor(new Color(0,0,70));
      g.fillRect(0,0,x,y);
    }
    else if (paint=="red") {
      g.setColor(new Color(250,0,0));
      g.fillRect(0,0,x-x*127/128+1,y);
      g.setColor(new Color(200,0,0));
      g.fillRect(x-x*127/128,0,x*2/128+1,y);
      g.setColor(new Color(175,0,0));
      g.fillRect(x-x*63/64,0,x*1/64+1,y);
      g.setColor(new Color(150,0,0));
      g.fillRect(x-x*31/32,0,x*1/32+1,y);
      g.setColor(new Color(120,0,0));
      g.fillRect(x-x*15/16,0,x*1/16+1,y);
      g.setColor(new Color(80,0,0));
      g.fillRect(x-x*7/8,0,x*1/8+1,y);
      g.setColor(new Color(50,0,0));
      g.fillRect(x-x*3/4,0,x*1/4+1,y);
      g.setColor(new Color(10,0,0));
      g.fillRect(x/2,0,x/2+1,y);
    }

    g.setFont(new Font("Helvetica",Font.PLAIN,10));
    g.setColor(Color.white);
    g.drawString(chartname,5,15);
    g.setColor(Color.yellow);
    plot(g);
  }

} // Ende Zeichenen**************************************

/************************************************
*                                      *
* Klasse DataSet                       *
*                                      *
************************************************/
class DataSet1 {

  static double[] data = new double[512];

  DataSet1() {
  }
  static double[] sinus() {
    for (int i=0; i<512; i++) {
      data[i] = Math.sin(((double)i)/10);
    }
    double r = range(data);
    for (int i=0; i<512; i++) {
      data[i] = data[i]/r;
    }
   return data;
  }
  static double[] sin2freq() {
    int i;
    for (i=0; i<256; i++) {
      double x = ((double)i)/10;
      data[i] = Math.sin(x);
      if (x>8*Math.PI) break;
    }
    for (i=i; i<512; i++) {
      double x = ((double)i)/5;
```

```
    data[i] = Math.sin(x);
  }
  double r = range(data);
  for (int j=0; j<512; j++) {
    data[j] = data[j]/r;
  }
 return data;
}
static double[] sincos() {
  for (int i=0; i<512; i++) {
    data[i] = (Math.sin(((double)i)/10)) + (Math.cos(((double)i)/4));
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}

static double[] sinPow2() {
  for (int i=0; i<512; i++) {
    data[i] = Math.sin(Math.pow(((double)i)/5, 2));
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}
static double[] sinPo2() {
  for (int i=0; i<512; i++) {
    data[i] = (Math.pow(i,2))*Math.sin(((double)i)/4);
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}
static double[] sinImpuls() {
  for (int i=0; i<512; i++) {
    data[i] = (Math.sin(((double)i)/10)) + (Math.cos(((double)i)/4));
    if (i==150) data[i] = -10;
    if (i==300) data[i] = 10;
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
  }
 return data;
}
static double[] saege() {
  for (int i=0; i<512; i+=32) {
    for (int j=0; j<=16; j++) {
      data[i+j] = 2*j;
    }
    for (int j=17; j<32; j++) {
      data[i+j] = 68+(-2)*j;
    }
  }
  double r = range(data);
  for (int i=0; i<512; i++) {
    data[i] = data[i]/r;
```

```
    }
   return data;
  }

  static double range(double[] d) {
    double max, min, range;
    max=-1.0e+14; min=-max;
    for (int i=0; i<512; i++) {
      if(d[i]>max) max=d[i];
      if(d[i]<min) min=d[i];
    }
    range=max-min;
    return range;
 }
} // Ende Klasse DataSet
```

Literaturverzeichnis

Literatur zur Wavelet-Theorie

http://www.isca-louis.com/wavelets.html

http://www.wavelet.org/wavelet/tutorial

http://nt.eit.uni-kl.de/wavelet

FRANZ BACHMANN, WERNER BÄNI (1998) *Wavelets – ein neues Werkzeug der Signalverarbeitung*

C. SIDNEY BURRUS, RAMESH A. GOPINATH, HAITAO GUO (1998) *Introdction* TO *Wavelets and Wavelet Transforms*

CHARLES K. CHUI (1992) *An Introduction to Wavelets* Academic Press Limited, London

GORDON ERLEBACHER, M. YOUSUFF HUSSAINI, LELAND M.JAMESON (1996) *Wavelets Theory and Applications*

OTTO FORSTER, JOACHIM WEHLER (2000) *Fourier-Transformation und Wavelets*, LMU München

M. HOLSCHNEIDER (1995) *Wavelets An Analysis Tool*

BARBARA BURKE HUBBARD (1997) *Wavelets die Mathematik der kleinen Wellen* Birkenhäuser Verlag, Berlin

KLAUS JÄNICH (1993) *Lineare Algebra* 5.Auflage, Springer-Verlag, Berlin

GERALD KAISER (1994) *A Friendly Guide to Wavelets* Birkenhäuser, Boston

WOLFGANG KELLER (2000) *Lecture notes on wavelets* Universität Stuttgart

ALFRED KARL LOUIS, PETER MAAß, ANDREAS RIEDER (1998) *Wavelets: Theorie und Anwendungen* 2.Auflage, Teubner, Stuttgart

HOWARD L. RENIKOFF (1998) *Wavelet Analysis* Springer-Verlag, New York

Literatur zur Programmiersprache JAVA

www.javasoft.com

www.java.sun.com/docs/books/tutorial/

JUDITH BISHOP (2001) *Java lernen* 2.Auflage, Addison-Wesley-Verlag

DAVID FLANAGAN (2001) *Java Examples in a Nutshell*, O'Reilly-Verlag

MARKUS GUMBEL, MARCUS VETTER, CARLOS CÁRDENAS (2000) *Java Standard Libraries*, Addison-Wesley-Verlag

FRITZ JOBST (1996) *Programmieren in Java Carl* Hanser Verlag

WOLFGANG KÜCHLIN, ANDREAS WEBER (2000) *Einführung in die Informatik*, Springer Verlag

ALBRECHT WEINERT (2001) *Java für Ingenieure*, Fachbuchverlag Leipzig

LOTHAR ZEYER (2001) *Workshop Java 2*, Addison-Wesley-Verlag