TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

D I P L O M A R B E I T

Dynamische Fourier - Synthese zur Beschreibung musikalischer und physikalischer Zusammenhänge in Java

Ausgeführt am

Institut für Analysis und Scientific Computing

der Technischen Universität Wien

unter der Anleitung von **Prof. Mag. Dr. Gabriela Schranz-Kirlinger**

durch

David STADLER

Kapellenstraße 32, 2191 Atzelsdorf

_____ _____

Datum Unterschrift

Inhaltsverzeichnis

Danksagung

Ich möchte mich ganz herzlich bei all jenen bedanken, die mich bei der Entstehung dieser Diplomarbeit unterstützt haben.

Insbesondere bei Fr. Mag. Dr. Gabriela Schranz-Kirlinger möchte ich mich für die Möglichkeit, dieses Thema zu bearbeiten und für ihre Unterstützung im Rahmen dieser Arbeit bedanken.

Weiters möchte ich mich bei meiner Familie bedanken, die mir zu jeder schwierigen Stunde meines Studiums die richtige Unterstützung gegeben hat.

Zusammenfassung

Fourier - Analyse ermöglicht es auf einfache Weise, die periodische Fortsetzung einer Funktion zu finden. Hierfür ist fundamentales Wissen über die Beschaffenheit von Funktionen, sowie Integration notwendig.

Fouriers Theorie ist in vielen mathematischen und naturwissenschaftlichen Bereichen von Bedeutung: Akustik, Optik, Zahlentheorie, Statistik, Signalverarbeitung und Kryptographie sind nur einige Beispiele, wo sie Anwendung findet.

Je nach Anwendungsgebiet ist die Fourier - Analyse anders zu interpretieren. In dieser Arbeit wird Bezug auf die musikalischen und physikalischen Zusammenhänge bzgl. Grundschwingung und Oberschwingungen genommen. Dieser Zusammenhang ist auf die Überlagerung von Schwingungen mit ganzzahligem Vielfachen der Frequenz der Grundschwingung zurückzuführen.

Durch das in dieser Diplomarbeit in der Programmiersprache Java realisiertem Programm **FourierApp** ist dieser Zusammenhang gut erkennbar und ermöglicht durch Änderung der Fourier - Koeffizienten die Auswirkungen auf das überlagerte Signal optisch wie akustisch zu untersuchen.

Die Diplomarbeit *Dynamische Fourier - Synthese zur Beschreibung musikalischer und physikalischer Zusammenhänge in Java* umfasst alle notwendigen Grundlagen um Funktionenanalyse und -synthese auszuführen und beinhaltet ein Java- Programm, um diese praktisch durchzuführen.

In der abschließenden Diskussion wird neben allgemeinen Problemen der Abhandlung auch auf den Einsatz im Schulbetrieb eingegangen.

1 Einleitung

1.1 Motivation

Die Fourier - Reihenentwicklung ist im Lehrplan der österreichischen berufsbildenden höheren Schulen genauso enthalten, wie im Studienplan von (technischer) Mathematik und naturwissenschaftlichen Studien. Natürlich steht in der Schule die Konstruktion von Signalen ohne tiefem Verständnis der Fourier - Theorie im Vordergrund. Dennoch ist die Anwendung selbst relativ schnell zu erlernen, da bloß ein Grundverständnis von Funktionen und Integration verlangt wird und Simulationssysteme bzw. Berechnungssysteme zeigen, dass das errechnete Ergebnis dem gewollten periodischen Signal entspricht. Für mich war dieses Erlebnis einer der Höhepunkte der Mathematik in der Oberstufe.

Im Musikstudium wurde mir gelehrt, wie Klangfarben entstehen, was die Begriffe Grundton, Obertöne und Schwebung bedeuten. Dabei sah ich nicht unmittelbar den Zusammenhang zur Fourier - Theorie, aber im Laufe des Mathematik - Studiums erkannte ich, dass es sich nur um die Überlagerung von Signalen mit ganzzahligem Vielfachen der Grundfrequenz dreht.

Diesen Zusammenhang von Grundton, Oberton und dem überlagerten Signal möchte ich in dieser Arbeit erarbeiten und grafisch darstellen. Folglich soll diese Diplomarbeit die Bereiche Musik, Physik, Mathematik und Informatik verbinden:

- In Musik / Physik wird auf die harmonische Schwingung, Intensität, Schall, stehende Wellen, Klangfarbe, Oberton, Überlagerung und Spektraldarstellung eingegangen (siehe Kapitel 2.1).
- In Mathematik wird auf notwendige Elemente der Fourier - Analysis (periodische Funktion, stückweise Stetigkeit, Fourier - Reihenentwicklung, Konvergenz der Fourier - Reihe) Bezug genommen (siehe Kapitel 2.2).
- Zur grafischen Darstellung wird schließlich in der Programmiersprache Java ein Programm (**FourierApp**) konstruiert (siehe Kapitel 2.3).

Im Zentrum der Abhandlung steht die Fourier - Analyse, insbesondere die Fourier - Synthese, wobei sich dieser Begriff auf die Zusammensetzung einer Funktion bzgl. ihrer Einzelkompenenten bezieht.

Zum Visualisieren entschloss ich mich, ein Java- Applet zu programmieren, da Applets alle notwendigen grafischen Erfordernisse erfüllen und internetfähig sind.

1.2 Ziel der Arbeit

Ziel der Arbeit ist es **nicht**, Fourier - Reihen zu berechnen, da es dafür bereits eine Vielzahl von Hilfsmitteln gibt. Stattdessen soll das Programm die Möglichkeit bieten, ausgehend von einem Grundsignal die Fourier - Koeffizienten zu ändern. Die Auswirkungen einer solchen Parametermodifikation werden dann sowohl grafisch als auch akustisch demonstriert. Hierfür sollen dem Anwender des Programms als Ausgangspunkt drei Möglichkeiten gegeben sein:

1. Er soll ein vorprogrammiertes Signal (Sinus-, Kosinus-, Rechteck-, Dreieck-, Sägezahnsignal) auswählen können.
2. Er kann eine selbst- ermittelte Fourier - Reihe eingeben.
3. Er soll das Signal löschen können.

Nach einer gewählten Methode, kann die Änderung der Fourier - Koeffizienten durchgeführt werden.

1.3 Aufbau der Arbeit

Nach Behandlung aller notwendigen Grundlagen (die in 1.1 angeführt sind) werde ich den Aufbau des Java- Applets präsentieren. Hierfür wurde die Entwicklungsumgebung „eclipse“[1] verwendet. Nach der Programmierung folgen nähere Erläuterungen zur Software begleitet von einer Bedienungsanleitung. Schließlich will ich zur Diskussion stellen, welche Probleme aufgetreten sind und wo das Programm Anwendung finden kann.

[1] Erhältlich unter `http://www.eclipse.org/`

2 Grundlagen

Um eine geeignete Basis für die Realisierung zu schaffen, werden nun die notwendigsten Grundlagen der Physik, Musik, Mathematik und Informatik erläutert.

2.1 Grundlagen Physik / Musik

In unserer Umwelt gibt es kaum einen Ort, an dem völlige Stille herrscht. Überall ist man, ob gewollt oder nicht, von Schallwellen umgeben. Wenn man Musik hört, so ist das meist gewollt und man erwartet hauptsächlich Töne, die die Ästhetik des Stücks, der Synfonie, der Ballade usw. betonen. Musik stellt neben ihrem ästhetisch, emotionalen Aspekt aber auch ein komplexes mathematisches Themengebiet da.

In den nun folgend angeführten Grundlagen der Physik und Musik wird auf die wichtigsten Begriffe zum Arbeiten im Bereich der Akustik eingegangen.

2.1.1 Die harmonische Schwingung

Als *Schwingung* oder *periodische Bewegung* bezeichnet man den zeitlichen Verlauf einer Zustandsänderung um einen Ruhepunkt, welcher sich in regelmäßigen Zeitabschnitten wiederholt. Die Amplitude gibt die maximale Auslenkung vom Ruhepunkt an und die Frequenz gibt an, wie oft sich diesr Vorgang pro Sekunde wiederholt.

Die Schwingung ist für die Physik und Musik von größter Bedeutung und als wichtiger Spezialfall soll hier die *harmonische* Schwingung angeführt sein, da sie auf einfache Weise die im Weiteren verwendeten Ausdrücke in Zusammenhang bringt.

Eine Schwingung wird als *harmonisch* bezeichnet, wenn die Rückstellgröße (z. B. die rückstellende Kraft) proportional zur Auslenkung ist [23]. Abbildung 2.1 soll die wichtigsten Begriffe einer harmonischen Schwingung visualisieren.

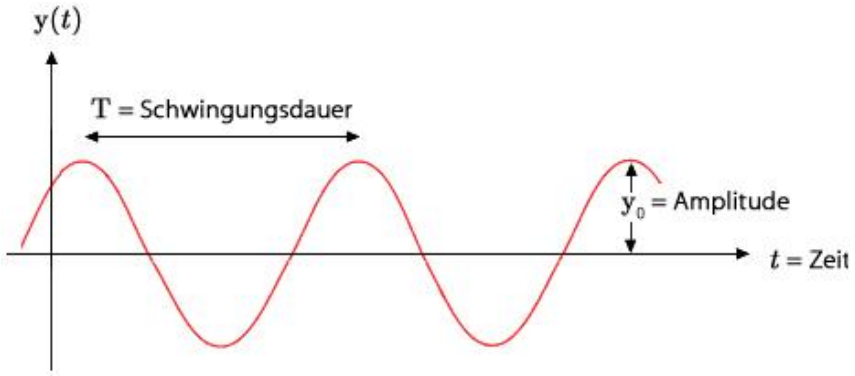

Abbildung 2.1: Darstellung des zeitlichen Verlaufs einer harmonischen Schwingung

Die Schwingungsdauer wird auch Periodendauer T genannt. Die Anzahl der Schwingungen pro Zeiteinheit wird als Frequenz f bezeichnet. Es gilt folgender Zusammenhang:

$$f = \frac{1}{T}$$

Menschen sind in der Lage Frequenzen zwischen etwa 15 und 20 000 Hertz zu hören. Diesen Frequenzbereich nennt man *Hörschall*. Niedrige Frequenz werden als Infraschall, höhere Frequenzen ab 20 kHz als Ultraschall bezeichnet. Die SI[1]- Einheit Hertz definiert hierbei die Anzahl der Schwingungen pro Sekunden. Es gilt:

$$1\text{Hz} = \frac{1}{s}$$

Die Amplitude kann als größte Entfernung des schwingenden Körpers von der Ruhelage angesehen werden. Von unserem Ohr wird ein Ton umso lauter empfunden, je höher die Amplitude ist.

Eine Schwingung lässt sich beschreiben durch:

$$y(t) = y_0 \sin(2\pi f t + \varphi_0),$$

wobei y_0 die Amplitude, f die Frequenz und φ_0 die Anfangsphase der Schwingung bestimmt.

Durch Einführung der Kreisfrequenz $\omega = 2\pi f$ kann die Schwingung etwas einfacher beschrieben werden:

$$y(t) = y_0 \sin(\omega t + \varphi_0)$$

[1]SI ist die Abkürzung von *Systeme international d'unites* und bezeichnet das internationale Einheitensystem [14].

Neben der oben angeführten ungedämpfen Schwingung gibt es noch die

- gedämpfte Schwingung,
- freie, erzwungene (oder fremderregte), selbsterregte und parametererregte Schwingung,
- lineare und nichtlineare Schwingung und
- Schwingungen mit einem odere mehreren Freiheitsgraden [23].

Auf diese Spezialfälle wird aber in dieser Arbeit nicht eingegangen.

Die Schwingung ist eine Funktion der Zeit, bei der sonst keine physikalische Komponente per Definition auftritt, d.h. es wird keine Energie transportiert, umgewandelt etc. Erst bei der Welle kommt neben der Zeit die zusätzliche Komponente des Ortes hinzu, wobei unser Raum durch Ortskoodinaten eindeutig beschrieben werden kann. Jeder Punkt unseres Raums ist ein schwingungsfähiges System, welches an benachbarte Raumpunkte gekoppelt ist (Moleküle der Luft, von Festkörpern usw.). Eine Schwingung, welche an ein Medium wie beispielsweise Luft gebunden ist und die enthaltenen Moleküle in Bewegung bringt, kann sich als Welle fortpflanzen. Beispiele hierfür sind

- elektromagnetische Wellen,
- Wasserwellen,
- Licht und
- Schallwellen [26].

2.1.2 Intensität

Die Leistung, die pro Flächeneinheit durch eine Fläche tritt, wird als *Intensität* bezeichnet. Alternativ kann die Intensität auch dadurch charakterisiert werden, dass als diese die Entfernung bezeichnet wird, in der ein Ton von seiner Quelle noch hörbar ist. Gemessen wird die Tonintensität in Dezibel (dB). Das Dezibel kennzeichnet den dekadischen Logarithmus des Verhältnisses zweier gleichartiger Leistungsgrößen. Daher wird eine Zunahme der Lautstärke von etwa fünf Dezibel vom menschlichen Ohr bereits als doppelt so laut empfunden. Abbildung 2.2 soll Aufschluss darüber geben, welches Geräusch, welcher Ton ca. welche Schallleistung erzeugt. [22]

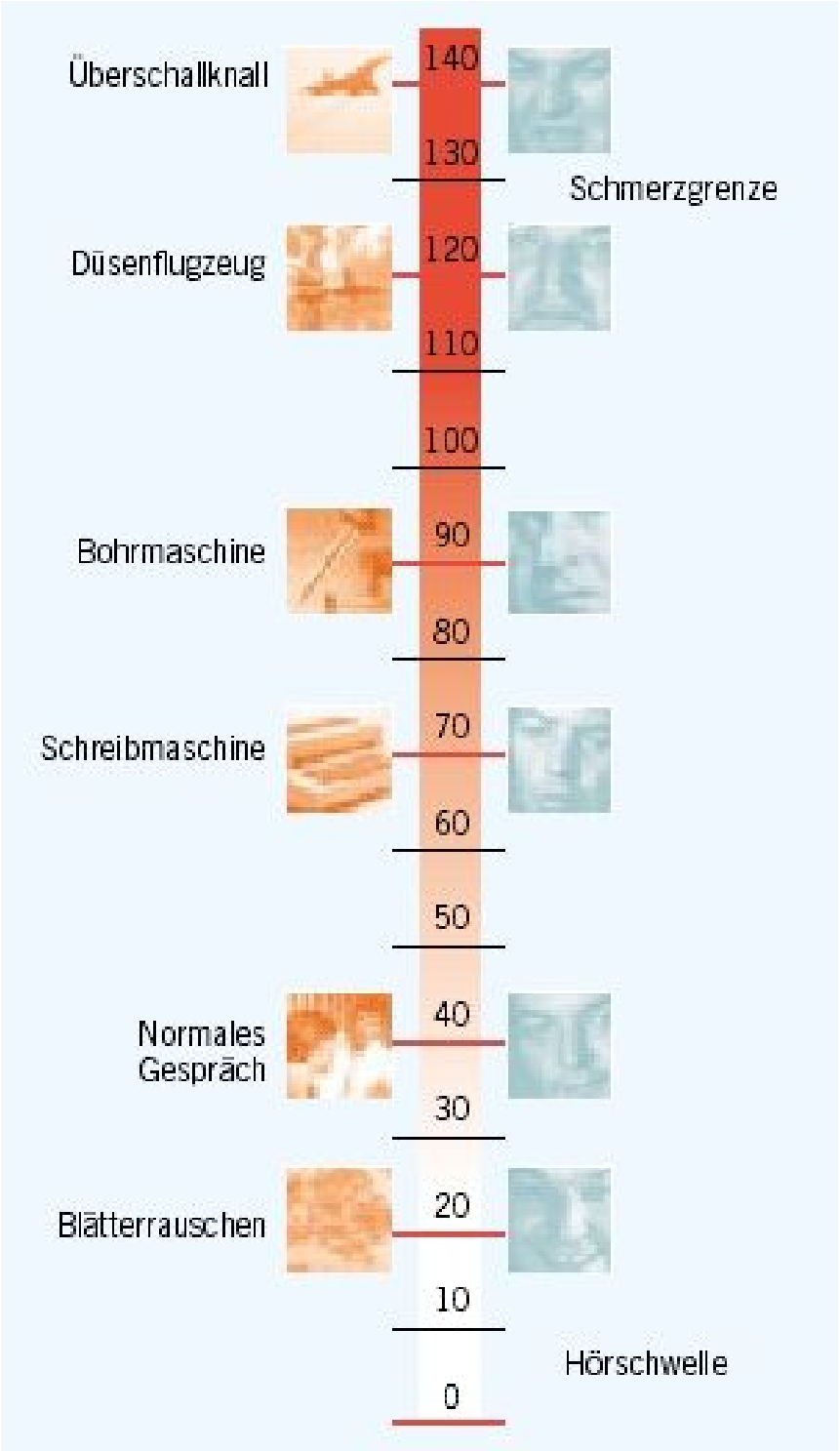

Abbildung 2.2: Beispiele für Schalldruckpegel in dB

2.1.3 Schall

Als *Schall* bezeichnet man (longitudinale) Wellen, welche vom Mensch (und Tier) mit dem Gehörsinn wahrgenommen werden können. Diese Wellen breiten sich in einem elastischen Medium, etwa in Gasen, Flüssigkeiten oder Festkörpern in Form von kleinen Druck- oder Dichteschwankungen aus. Man unterscheidet zwischen Störschall, welcher etwa bei Baustellenlärm auftritt, und Nutzschall, wie er etwa in der Musik auftritt.

Die Schallgeschwindigkeit hängt vom vorhandenem Medium ab. In trockener Luft bei 0°C beträgt diese $331,6\frac{m}{s}$, bei 20°C $343\frac{m}{s}$. Die Ausbreitung der Schallwellen erfolgt in Längsrichtung und diese sind somit *longitudinale* Wellen, d.h. die einzelnen Moleküle der Luft bewegen sich parallel zur Ausbreitungsrichtung und geben beim Stoß etwas Energie an das Nachbarmolekül ab. Wie in Abbildung 2.3 dargestellt, bewegen sich im Unterschied dazu bei der Transversalwelle (die nur in festen Stoffen auftreten können [36]) die Moleküle senkrecht zur Ausbreitungsrichtung, d.h. die Schwingung erfolgt senkrecht zur Ausbreitungsrichtung [24].

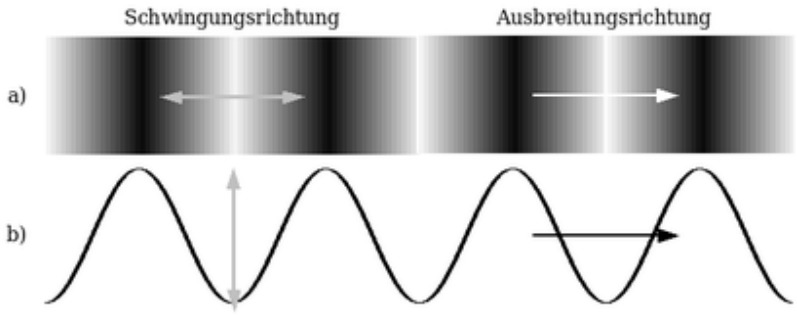

Abbildung 2.3: Schwingungs- und Ausbreitungsrichtung einer Longitudinalwelle (a) und einer Transversalwelle (b)

Wenn auf einem Musikinstrument ein Ton gespielt wird, so kann dieser durch Tonhöhe, Intensität und Klangfarbe eindeutig charakterisiert werden. Physikalisch betrachtet entspricht die Tonhöhe der gespielten Frequenz, die Intensität der Amplitude und die Klangfarbe der harmonischen Zusammensetzung (der Wellenform). [21][32]

2.1.4 Stehende Wellen einer Saite

Es sei eine Saite der Länge L und Masse m an den festen Punkten P und Q verankert und mit einer gegebenen Kraft T gespannt, welche beliebig geändert werden kann. Abbildung 2.4 veranschaulicht diesen Sachverhalt.

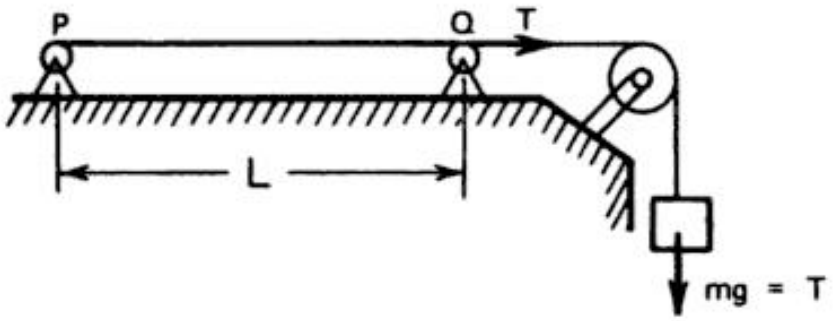

Abbildung 2.4: Verankerte Saite

Wenn nun diese Saite durch Zupfen an einem beliebigen Punkt in Schwingung versetzt wird, so breiten sich zwei Querwellenimpulse nach links und rechts aus, welche an den Ankerpunkten P und Q reflektiert werden. Ein positiver Impuls kommt dabei als negativer Impuls zurück und umgekehrt. Nach kurzer Zeit bewegen sich schließlich viele solcher Impulse in beiden Richtungen auf der Saite hin und her, solange, bis die induzierte Energie in Form von Schallwellen an das Medium abgegeben wurde.

Das Bild, dass durch die in Bewegung befindlichen Impulse entsteht, ist das einer scheinbar stehenden Welle und es kann gezeigt werden, dass stehende Wellen die einzig- mögliche Schwingungsform für eine Saite mit befestigten Enden (hier P und Q) sind. Diese Aussage hat zur Folge, dass unter allen stehenden Wellen nur solche möglich sind, welche als Knoten P und Q besitzen, die also zwischen P und Q hineinpassen. Wenn man l_N als halbe Wellenlänge einer einzelnen Schwingung betrachtet, so muss hierfür die Gleichung

$$L = nl_N = n\frac{\lambda}{2}, \quad n \in \mathbb{N}$$

gelten, wobei als Wellenlänge λ, der kleinste Abstand zweier Punkte gleicher Phase einer Welle bezeichnet [25]. Umgeformt erhält man, dass dabei nur Wellenlängen

$$\lambda_n = \frac{2L}{n}, \quad n \in \mathbb{N}$$

gestattet sind. Betrachtet man den Zusammenhang

$$\lambda = \frac{1}{f}\sqrt{\frac{T}{d}},$$

wobei T die Spannung (in Newton) und d die lineare Dichte (in $\frac{kg}{m}$), also die Masse pro Längeneinheit angibt, so erhält man für die möglichen Schwingungen:

$$f_n = \frac{1}{\lambda_n}\sqrt{\frac{T}{d}} = \frac{n}{2L}\sqrt{\frac{T}{d}} = nf_1 \tag{2.1}$$

Die niedrigst- mögliche Frequenz erhält man mit $n = 1$:

$$f_1 = \frac{1}{2L}\sqrt{\frac{T}{d}} = nf_1$$

Diese bezeichnet man als Grundfrequenz. In Formel (2.1) ist ersichtlich, dass nur Frequenzen möglich sind, die ganzzahlige Vielfache der Grundfrequenz haben. Die Schwingungen, die durch diese Frequenzen resultieren, nennt man *Harmonische* von f_1. [37].

2.1.5 Klangfarbe

Beim Einstimmen eines Orchesters wird üblicherweise der Kammerton A mit einer Frequenz von 440Hz herangezogen (die Frequenz kann von Orchester zu Orchester etwas variieren). Dabei ist das Ziel alle Instrumente auf eine Frequenz zu kalibrieren. Weiters ist es vorstellbar, dass beinahe alle Instrumente bei Bedarf dieselbe Amplitude erzeugen können. Trotzdem können sie eindeutig durch die Klangfarbe unterschieden werden. Während ein reiner Sinuston[2] ohne andere Frequenzen als solches eindeutig wahrgenommen werden kann, klingt beispielsweise eine Trompete, welche den gleichen Ton spielt, komplett anders, da bei diesem Instrument beispielsweise neben dem Grundton sehr viele Obertöne (Harmonische) des Grundtons mitklingen (siehe 2.1.6). Die Intensität der Obertöne bestimmt dabei die Klangfarbe des gehörten Tons.

2.1.6 Oberton

In der Musik sind verschiedene Klangfarben von Instrumenten dadurch gekennzeichnet, dass sie unterschiedliche Anteile von Vielfachen der Grundfrequenz besitzen. Diese Töne nennt man *Obertöne*[3]. Bei der schwingenden Saite (etwa einer Gitarrensaite) erkennt man auch den Zusammenhang mit der Saitenlänge: wird die Saite halbiert, respektive durch einen Finger am Griffbrett bei der Hälfte der Saite, so erklingt die doppelte Frequenz des Grundtons.

Beispiel: Die tiefste Saite einer Konzertgitarre ist die E- Saite. Wird diese am 12. Bund praktisch in der Hälfte geteilt, so erklingt wieder ein E, diesmal aber eine Oktave höher, was physikalisch einer Frequenzverdopplung entspricht. Wenn die Saite wieder in der Hälfte geteilt werden würde, so erhält man wieder ein E[4]. Würde man die Saite bzgl. der Ausgangslänge dritteln, so würde dies zu einer Frequenzverdreifachung des Grundtons führen, musikalisch entspricht das der Quint zum Grundton, bei E wäre dies das H [18]. Abbildung 2.5 beschreibt die harmonischen Obertöne einer idealisierten schwingenden Saite.

[2]Ein reiner Sinuston ist ohne technischen Hilfsmittel kaum produzierbar. Einzig die Stimmgabel liefert einen sehr ähnlichen Klang.

[3]Bei Blechblasinstrumenten werden diese Obertöne auch Naturtöne genannt.

[4]Die doppelte Teilung einer Gitarrensaite ist praktisch kaum realisierbar

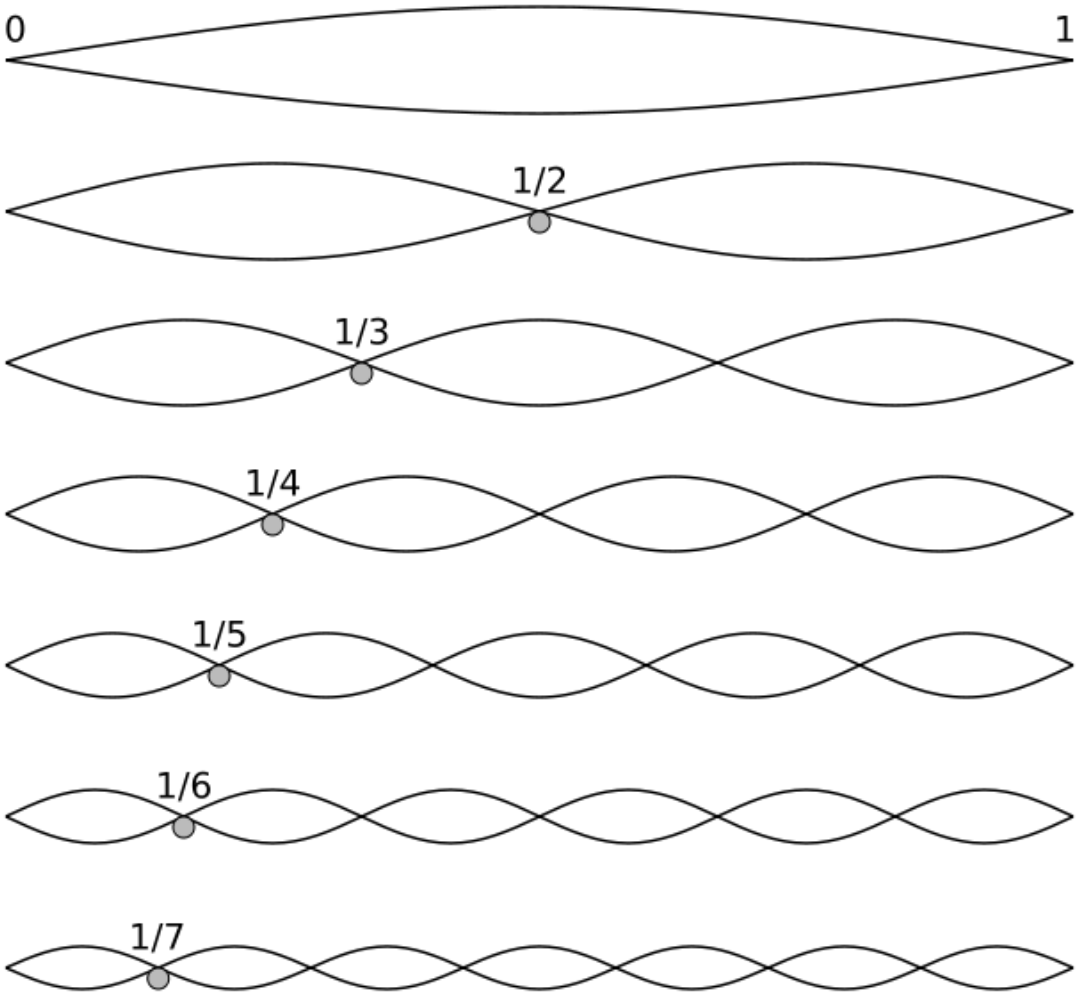

Abbildung 2.5: harmonische Obertöne einer idealisierten Saite

Folgende Tabelle soll für den Grundton C den Zusammenhang zwischen den Frequenzen und den zugehörigen Tönen repräsentieren:

Vergleich mit Grundton	Intervall zum vorhergehenden Ton	Frequenzverhältnis zum vorhergehenden Ton	Beispiel	Frequenz (Hz)
Grundton			C	66
doppelte Frequenz	Oktave	2:1	c	132
dreifache Frequenz	Quint	3:2	g	198
vierfache Frequenz	Quart	4:3	c'	264
fünffache Frequenz	große Terz	5:4	e'	330
sechsfache Frequenz	kleine Terz	6:5	g'	396
siebenfache Frequenz		7:6	siehe Naturseptime	462
achtfache Frequenz		8:7	c"	528
neunfache Frequenz	großer Ganzton	9:8	d"	594
zehnfache Frequenz	kleiner Ganzton	10:9	e"	660
elffache Frequenz		11:10	siehe Alphorn-Fa	726
zwölffache Frequenz		12:11	g"	792
dreizehnfache Frequenz		13:12		858
vierzehnfache Frequenz		14:13		924
fünzehnfache Frequenz		15:14	h"	990
sechzehnfache Frequenz	kleine Sekunde	16:15	c"'	1056

Tabelle 2.1: Frequenz/Ton- Zusammenhang, Quelle: http://de.wikipedia.org/wiki/Oberton

Obertöne können nur im Zusammenhang mit Grundtönen auftreten. Da ein einzelner aber nur im geringerem Ausmaß den Klang bildet, ist es kaum als Mensch möglich, den Oberton eines Klanges herauszufiltern. Allerdings kann man sich dazu eines Tricks bedienen:

Auf dem Klavier können Obertöne eines Tones hörbar gemacht werden, indem man die Tasten eines Akkords aus der Obertonreihe sanft niederdrückt, ohne dass die Hämmer die Saite berühren und dann den Grundton im Bassbereich kurz und stark anschlägt. Die Obertöne erzeugen nun eine Resonanz auf den ungedämpften Saiten der niedergedrückten Tasten, die man deutlich hören kann [18].

2.1.7 Überlagerung

Die Überlagerung von zwei oder mehreren Wellen nach dem Superpositionsprinzip wird als *Interferenz* bezeichnet. Diese tritt beispielsweise bei elektromagnetischen Wellen in der Optik, Funktechnik und Quantenmechanik auf [13]. In der Musik ergibt das Überlagern (oder Superponieren) von verschiedenen Frequenzen einen Akkord (üblicherweise Zwei-, Drei- oder Vierklang). Ein Dur- Dreiklang besteht etwa aus Grundton, (großer) Terz und Quint:

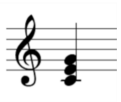

Abbildung 2.6: Dur- Dreiklang

Das Prinzip der *linearen Superposition* sagt aus, dass die resultierende Auslenkung bei vorhandenen Schwingungen in einem gemeinsamen Medium (wie beispielsweise Schallwellen in der Luft) einfach der algebraischen Summen der Einzelauslenkungen entspricht.

Schwingungen gleicher Frequenz und Richtung

Zunächst sei der Fall von Superposition zweier (oder mehrerer) Schwingungen gleicher Richtung und Frequenz, aber mit unterschiedlicher Phase angeführt([36]):

Seien

$$y_1 = b_1 \sin(\omega t + \varphi_1)$$
$$y_2 = b_2 \sin(\omega t + \varphi_2)$$

zwei Schwingungen mit $\varphi_1 - \varphi_2 = \Delta\varphi$ als Phasenunterschied. Ist insbesondere $\Delta\varphi = 0°$, so gehen die Bewegungen beider Schwingungen zu gleichen Zeiten und in gleicher Richtung durch die Nulllage und erreichen zu gleichen Zeiten auch die größte Auslenkungen (Abb. 2.7 (a)).

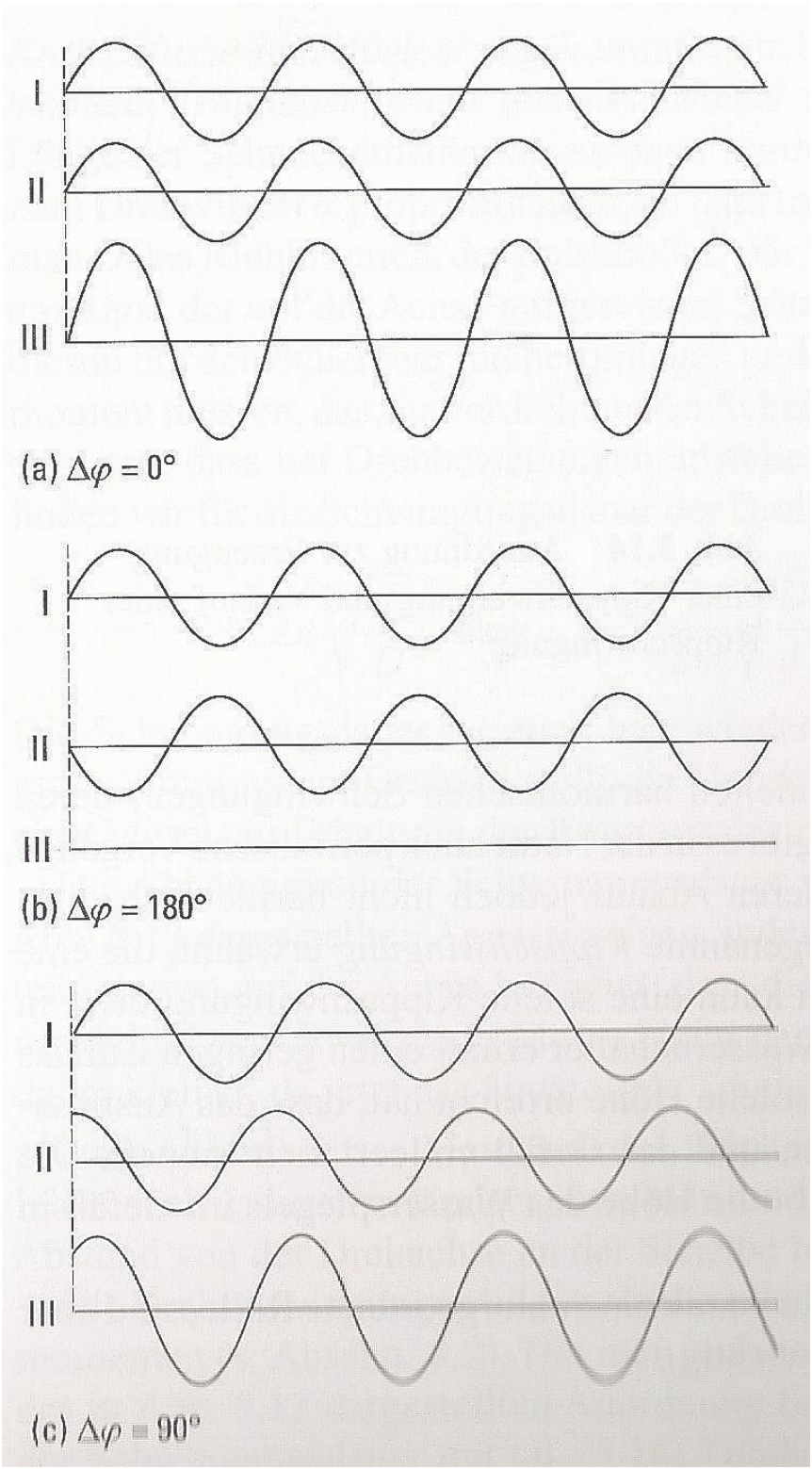

Abbildung 2.7: Zusammensetzung zweier Sinusschwingungen I und II gleicher Frequenz und Amplitude zu einer resultierenden Schwingung III bei verschiedener Phasendifferenz $\Delta\varphi$: (a) $\Delta\varphi = 0°$, (b) $\Delta\varphi = 180°$, (c) $\Delta\varphi = 90°$

Ist $\Delta\varphi = \pi = 180°$, so gehen beide Schwingungen zwar zu gleichen Zeiten durch die Nulllage, jedoch in verschiedene Richtungen und löschen sich dadurch bei Superposition aus (vgl. Abb. 2.7 (b)). Dazwischen sind alle anderen Fälle möglich. Abbildung 2.7 (c) zeigt den Fall, dass $\Delta\varphi = \frac{\pi}{2} = 90°$ ist.

Im wichtigsten Fall, der auch in dieser Arbeit im Vordergrund steht, verhält sich jede Schwingung so, als ob die andere (oder anderen) nicht existieren würden, dass heißt, dass sich zusammengesetzte Schwingungen nicht gegenseitig stören. Man spricht in diesem Zusammenhang auch von ungestörter Überlagerung, das bedeutet es werden die Amplituden der Einzelschwingungen addiert:

$$y = y_1 + y_2 = b_1 \sin(\omega t + \varphi_1) + b_2 \sin(\omega t + \varphi_2)$$

Zusammenfassend lässt sich feststellen, dass bei Überlagerung zweier Schwingungen gleicher Frequenz und Schwingungsrichtung eine Schwingung resultiert, die wieder dieselbe Frequenz und Schwingungsrichtung, allerdings eine andere Amplitude besitzt (abhängig von den Amplituden der Ausgangsschwingungen und deren Phasendifferenz).

Schwingungen verschiedener Frequenz aber gleicher Richtung

Es wird nun der Fall betrachtet, dass die zu untersuchenden Schwingungen dieselbe Richtung, allerdings unterschiedliche Frequenzen besitzen:

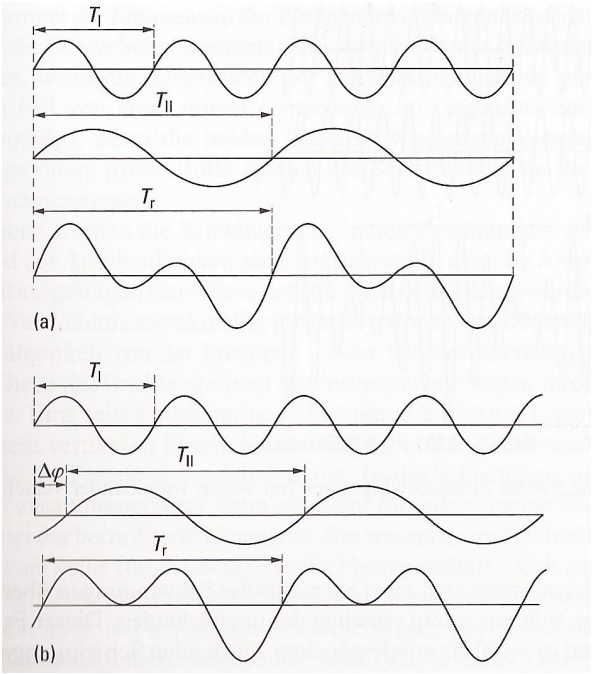

Abbildung 2.8: Zusammensetzung von Sinusschwingungen mit unterschiedlicher Frequenz: (a) $\Delta\varphi = 0°$, (b) $\Delta\varphi \neq 0°$

In Abbildung 2.8 (a) ist ersichtlich, dass die Phasendifferenz $\Delta\varphi$ zwischen den beiden Schwingungen null ist. Die Resultierende ist wieder eine periodische Schwingung mit der Periodendauer $T_r = T_{II}$ der niederfrequenten Schwingung. Die Funktionenform lässt den unmittelbaren Zusammenhang zu den Ausgangsschwingungen wie in den vorigen Beispielen nicht unmittelbar erahnen. Noch komplizierter wird die Resultierende, wenn wie in Abbildung 2.8 (b) die Phasenlage zusätzlich nicht gleich ist. Diese besitzt nun neben einer unterschiedlichen Funktionenform vorallem auch eine andere Periodendauer T_r.

Man kann zeigen, dass diese Periodendauer aus dem größten gemeinsamen Teiler des Zählers und Nenners der Zahlen ermittelt wird, welche das Frequenzverhältnis zwischen den beiden Schwingungen angibt. Wenn sich die Frequenzen zweier Schwingungen beispielsweise im Verhältnis 9:2 zueinander verhalten, so besitzt die resultierende Überlagerte eine doppelt so große Periodendauer, wie die Niederfrequente der beiden Ausgangsfrequenzen (vgl. Abb. 2.9).

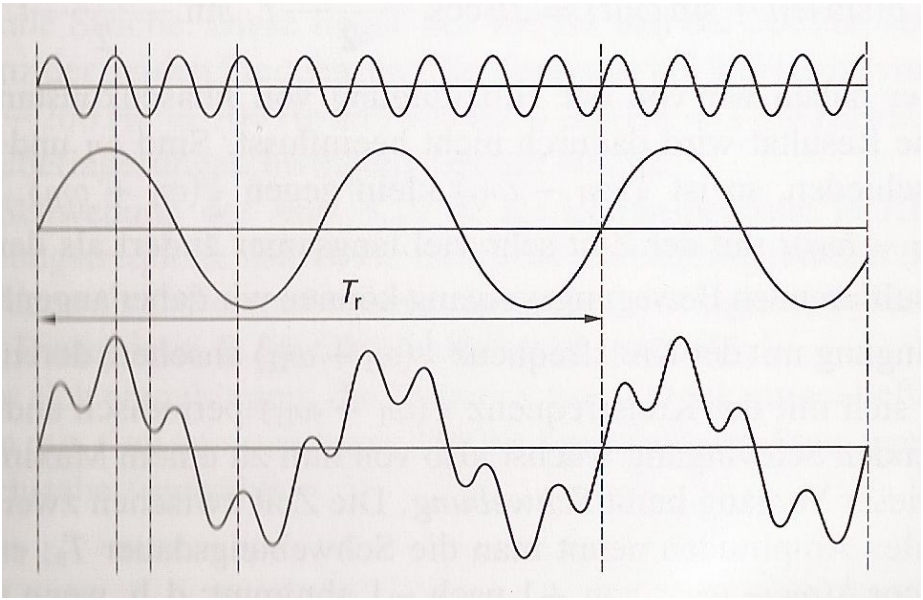

Abbildung 2.9: Zusammensetzung von Sinusschwingungen mit Frequenzverhältnis 9:2

Sind die Frequenzen der Ausgangsschwingungen inkommensurabel[5], so ist die resultierende Bewegung überhaupt nicht mehr periodisch.

[5]In der Mathematik heißen zwei reelle Zahlen a und b *inkommensurabel* (lat. nicht zusammen messbar), wenn sie keine ganzzahligen Vielfache einer geeigneten dritten reellen Zahl c sind, also keinen gemeinsamen Teiler besitzen [10].

2.1.8 Spektraldarstellung von Schwingungen

Schwingungen können auf sehr anschauliche Weise als Funktion der Zeit dargestellt werden. Bei den mathematischen Grundlagen wird auf diese Darstellung auch wieder eingegangen werden, es soll aber an dieser Stelle erwähnt sein, das Schwingungensamplituden auch in Abhängigkeit von der Frequenz visualisiert werden können. Man erhält dadurch die Spektraldarstellung dieser Schwingung. Als Beispiel sei hier eine Rechteckschwingung angeführt (vgl. Abb. 2.10).

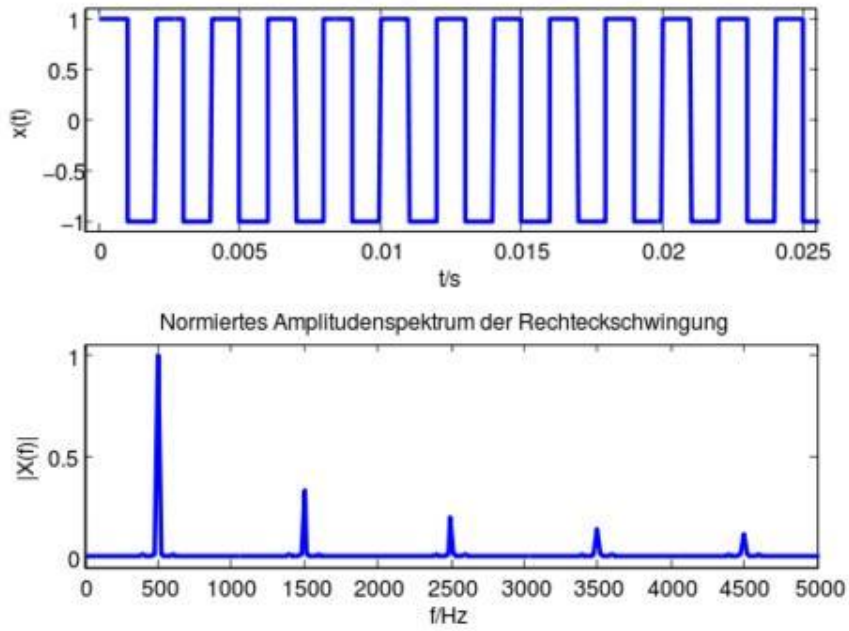

Abbildung 2.10: Spektraldarstellung einer Rechteckschwingung der Grundfrequenz 500 Hz

In der programmierte Software können die Amplituden für einzelne Frequenzen variiert und die dadurch Resultierenden betrachtet und angehört werden.

2.2 Grundlagen Mathematik

Joseph Fourier[6] stieß beim Versuch, ein Wärmeleitungsproblem zu lösen,[7] auf ein neues Problem, welches die Entwicklung von Funktionen in Reihen aus cos- und sin-Termen beinhaltete.

Auf periodische/stückweise stetige Funktionen, die die Voraussetzung für die Anwendung und Definition der Fourier - Reihe bilden, soll im Weiteren eingegangen werden.

2.2.1 Periodische Funktionen

Eine Funktion $f(t)$ heißt *periodisch* mit der Periode P ($\in \mathbb{R}$), wenn $\forall t : f(t+P) = f(t)$ gilt und P dabei den kleinst- möglichen Wert annimmt.

$\sin(t)$ hat beispielsweise die Periode 2π, da $\sin(t + 2\pi) = \sin(t)$ gilt. Weitere Beispiele sind die Rechteck-, Dreieck- und Sägezahnfunktion (vgl. 2.2.9).

2.2.2 Stückweise stetige Funktionen

Eine Funktion $f(t)$ heißt *stückweise stetig* in einem Intervall I, wenn gilt [39, S.21]:

- Das Intervall kann in eine endliche Anzahl von Teilintervallen zerlegt werden. $f(t)$ ist in diesen Teilintervallen stetig.
- Die Grenzwerte bei Annäherung von x gegen die Endpunkte des Teilintervalls sind endlich.

Folgende Funktion besitzt zwei Unstetigkeitsstellen. Da diese aber in drei Teilintervalle zerlegt werden kann, in denen $f(t)$ stetig ist, ist die Funktion stückweise stetig auf dem ganzen Intervall $[a, b]$. Abbildung 2.11 zeigt diese stückweise stetige Funktion mit zwei Unstetigkeitsstellen auf dem Intervall $[a, b]$.

[6] französischer Mathematiker und Physiker, 1768-1830

[7] Er formulierte dabei dieses Probleme als Gleichungssystem partieller Differentialgleichungen [39].

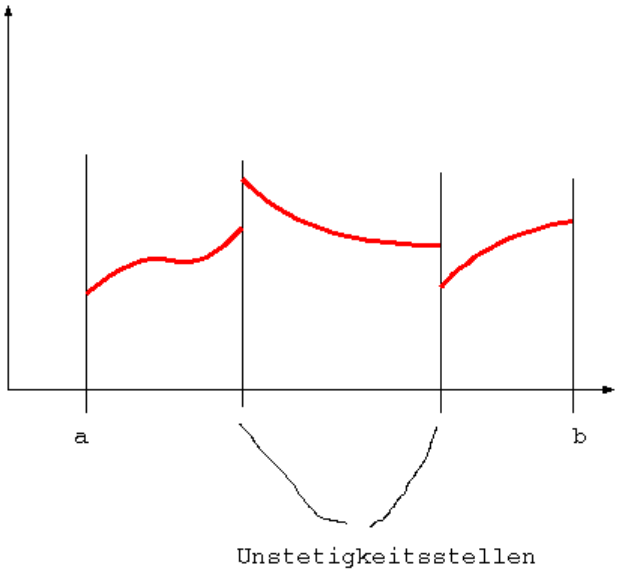

Abbildung 2.11: Stückweise stetige Funktion

2.2.3 Fourier - Analysis

Der Begriff der *Fourier - Analysis* wird in der Literatur leider nicht einheitlich gebraucht. So findet man bei der Recherche im wesentlichen drei Varianten, wenn man von Fourier - Analysis spricht:

- Das ist einerseits die Entwicklung einer in einem gegebenen Intervall gegebenen, periodischen Funktion in eine Fourier - Reihe.
- Andererseits spricht man in diesem Zusammenhang auch von der kontinuierlichen Fourier - Transformation, bei der es um die Zerlegung kontinuierlicher, aperiodischer Funktionen in ein kontinuierliches Spektrum geht.
- Schließlich kann es sich bei der Fourier - Analysis um die diskrete Fourier - Transformation handeln, bei der nur diskrete, äquidistante Zeitpunkte in einem endlichen Intervall bekannt sind [7].

Zusammengefasst erhält man also bei allen drei Varianten unterschiedliche Definitionsmengen, Periodizitäten und Frequenzspektren von $f(t)$:

Variante	Definitionsmenge von f	Periodizität von f	Frequenzspektrum
Fourier-Reihe	kontinuierliches Intervall	periodisch	diskret
Kontinuierliche Fourier-Transformation	kontinuierlich	aperiodisch	kontinuierlich
Diskrete Fourier-Transformation	diskret, endlich	periodisch	diskret, endlich

Tabelle 2.2: Charakteristika der verschiedenen Teilbereich der Fourier - Analysis, Quelle: http://de.wikipedia.org/wiki/Fourier-Analysis

In dieser Arbeit wird der Schwerpunkt auf die erste erwähnte Variante geworfen.

2.2.4 Fourier - Synthese

Im vorigen Kapitel wurden die kontinuierliche und diskrete Fouriertransformation erwähnt. Zu all diesen Transformationen existiert eine inverse Transformation. In Technik, Physik und Mathematik spricht man in diesem Kontext vom Zerlegen einer Funktion in ihr Spektrum. Den Prozess des Darstellens einer Ausgangsfunktion durch dessen Spektrum wird als Fourier - Synthese bezeichnet. Dieser Begriff tritt häufig im Zusammenhang mit diskreter und schneller- Fourier - Transformation (FFT, Fast- Fourier - Transformation) auf [7]. In dieser Arbeit ist aber auch von Fourier - Synthese die Rede, wenn es um die Zusammensetzung einer Funktion bzgl. ihrer Einzelkompenenten (harmonische Schwingungen) geht.

2.2.5 Die Fourier - Reihe

Sei $f(t)$ eine auf dem Intervall $(-L, L)$ stückweise stetig definierte Funktion und habe die Periode $2L$ ($f(t+2L) = f(t)$), so hat die Fourier - Reihe zu $f(t)$ die Gestalt

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos(\frac{n\pi t}{L}) + b_n \sin(\frac{n\pi t}{L}) \tag{2.2}$$

wobei die Fourier - Koeffizienten a_n und b_n gegeben sind durch

$$\begin{aligned} a_n &= \frac{1}{L}\int_{-L}^{L} f(t)\cos(\frac{n\pi t}{L})dt \\ b_n &= \frac{1}{L}\int_{-L}^{L} f(t)\sin\frac{n\pi t}{L}dt \\ n &= 0, 1, 2, \cdots \end{aligned} \tag{2.3}$$

Wenn $f(t)$ die Periode $2L$ hat, kann man die Koeffizienten a_n und b_n auch bestimmen aus

$$\begin{aligned} a_n =& \frac{1}{L}\int_c^{c+2L} f(t)\cos(\frac{n\pi t}{L})dt \\ b_n =& \frac{1}{L}\int_c^{c+2L} f(t)\sin\frac{n\pi t}{L}dt \\ n =& 0,1,2,\cdots \end{aligned} \tag{2.4}$$

wobei c beliebig gewählt werden kann. Es sei an dieser Stelle betont, dass (2.2) die einzige Reihe ist, der $f(t)$ entspricht [39, S.23].

Der konstante Term $\frac{a_0}{2}$ stellt das Mittel von $f(t)$ über eine Periode dar und berechnet sich aus

$$\frac{a_0}{2} = \frac{1}{2L}\int_{-L}^{L} f(t)dt \tag{2.5}$$

Wenn T der Periodendauer und $\omega = \frac{2\pi}{T}$ entspricht, erhält man die Fourier - Reihe und ihre Koeffizienten in einfacherer Form:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n\cos(n\omega t) + b_n\sin(n\omega t))$$

$$a_n = \frac{2}{T}\int_c^{c+T} f(t)\cos(n\omega t)$$

$$b_n = \frac{2}{T}\int_c^{c+T} f(t)\cos(n\omega t)$$

Der Gleichanteil $\frac{a_0}{2}$ berechnet sich aus

$$\frac{a_0}{2} = \frac{1}{T}\int_c^{c+T} f(t)$$

Fourier zeigte, dass eine, von mehreren harmonischen Schwingungen zusammengesetzte Resultierende, welche als solche wieder eine harmonische Schwingung darstellt, stets eindeutig in die Summe von harmonischen Teilschwingungen (trigonometrische Funktionen) zerlegt werden kann[8]. Anwendung findet die Fourier - Reihenentwicklung in vielen Bereichen der Technik und Musik. Man ist dadurch in der Lage beliebige Schwingungsformen zu konstruieren bzw. umgekehrt jede beliebige Schwingungsform in ihre Spektralkomponenten umzuwandeln (siehe 2.1.8).

[8] Zwar ist die Fourier - Analyse wie in 2.2.3 auch auf aperiodische Bewegungen anwendbar, für diese Arbeit, die einen Bezug zur Musik haben soll, wird aber von diesem allgemeinem Fall abgesehen.

2.2.6 Berechnung von Fourier - Reihen

Das Berechnen von Fourier - Reihe ist ein relativ langwieriges Prozedere, von welchem, außer zu Schulungszwecken, in Zeiten des Computerzeitalters immer mehr abgesehen werden kann. Derzeit ist aber noch das Berechnen von Fourier - Reihen im Lehrplan der 4. Klasse HTL vorgesehen [30]. Es sei hier erwähnt, wie sich die Berechnung von Fourier - Reihen bei ungeraden bzw. geraden Funktionen vereinfacht.

(Un)gerade Funktionen

Eine Funktion $f(x)$ wird als *ungerade* bezeichnet, wenn $f(-x) = -f(x)$ gilt. x^3 und $\sin(x)$ sind Beispiele für ungerade Funktionen (siehe Abbildung 2.12).

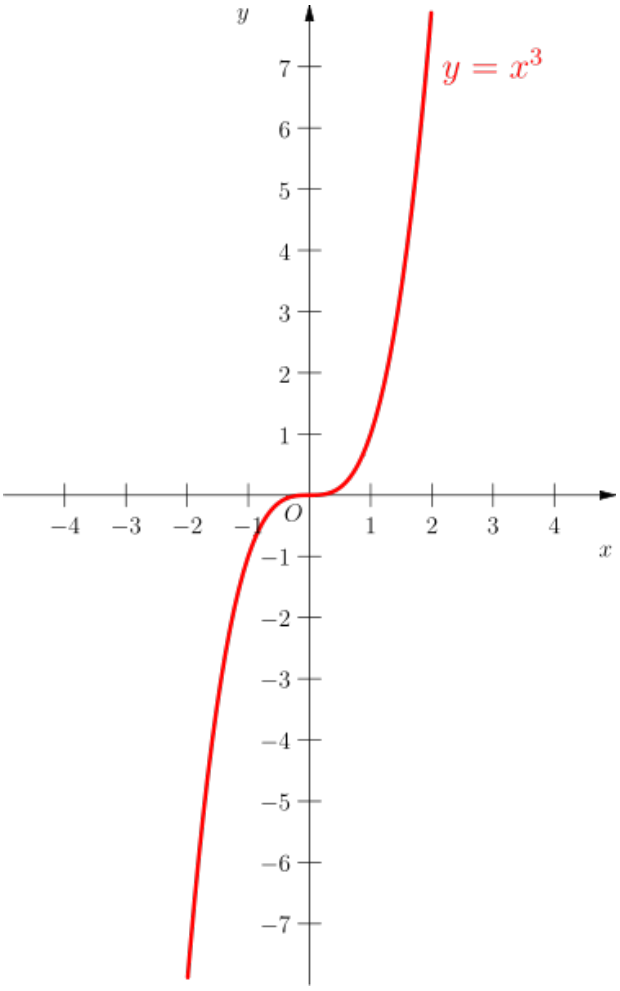

Abbildung 2.12: Ungerade Funktion

Eine Funktion $f(x)$ wird als *gerade* bezeichnet, wenn $f(-x) = f(x)$ gilt. x^2 und $\cos(x)$ sind Beispiele für gerade Funktionen (siehe Abbildung 2.13).

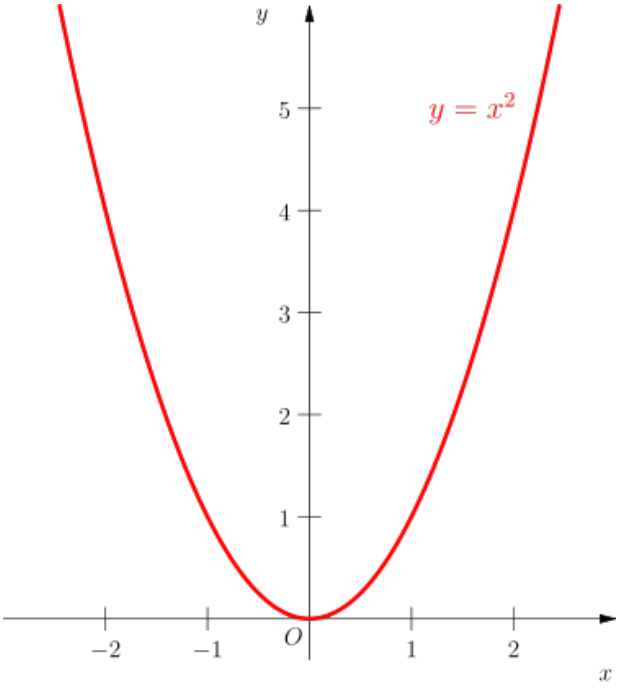

Abbildung 2.13: Gerade Funktion

Die Berechnung der Fourier - Reihe vereinfacht sich bei ungeraden bzw. geraden Funktionen dadurch, als dass bei ungeraden Funktionen nur Sinus- Ausdrücke ($a_n = 0$) und umgekehrt bei geraden Funktionen nur Kosinus- Ausdrücke auftreten ($b_n = 0$). [39, S.22]

2.2.7 Geschichte der Fourier - Reihe

Leonhard Euler[9], Joseph- Louis Lagrange[10] und Jakob I. Bernoulli[11] errechneten bereits im 18.Jahrhundert die Fourier - Reihen für einige Funktionen. Fourier postulierte im 19.Jahrhundert in seiner Arbeit *Theorie analytique de la chaleur*, dass es sogar für alle Funktionen möglich ist, eine Fourier - Reihe zu konstruieren. Augustin Louis Cauchy[12] und Niels Henrik Abel[13] lehnten diese Behauptung allerdings ab. Peter Gustav Lejeune Dirichlet[14] konnte 1829 beweisen, dass Fouriers Behauptung zumindest für Lipschitz- ste-

[9] Deutscher Mathematiker, 1707-1783
[10] Italienischer Mathematiker und Astronom, 1736-1813
[11] Schweizer Mathematiker und Physiker, 1654-1705
[12] Französischer Mathematiker, 1789-1857
[13] Norwegischer Mathematiker, 1802-1829
[14] Deutscher Mathematiker, 1805-1859

tige Funktionen zutrifft. 1876 fand Paul Du Bois-Reymond[15] eine stetige Funktion, deren Fourierreihe divergiert. Im 20. Jahrhundert konnte schließlich Lennart Carleson[16] beweisen, dass es für stetige oder stückweise stetige Funktionen immer konvergente Fourierreihen gibt, wenn man den Konvergenzbegriff etwas abschwächt [8].

2.2.8 Konvergenz der Fourier - Reihe

Konvergenzaussagen über beliebige Fourier - Reihen beschäftigten wie in 2.2.7 dargelegt die Mathematik über einige Jahrhunderte. Es seien an dieser Stelle in kompakter Form zwei wichtige Konvergenzaussagen über Fourier - Reihen angeführt.

Satz von Dirichlet

Der Satz von Dirichlet bezieht sich auf die punktweise Konvergenz[17] einer Fourier - Reihe gegen ihre Ausgangsfunktion. Dirichlet zeigte, dass diese Aussage genau dann gegeben ist, wenn die zu entwickelnde Funktion 2π- periodisch und differenzierbar ist.

Wenn $f(t)$ 2π- periodisch und stetig differenzierbar ist, dann konvergiert die Fourier - Reihe sogar gleichmäßig[18] gegen die Ausgangsfunktion.

Satz von Fejer

Leopold Fejer[19] zeigte, dass das arithmetische Mittel der Partialsummen der Fourierreihe einer stetigen, 2π-periodischen Funktion gleichmäßig gegen die Funktion konvergiert [8].

Eine Folgerung aus dem Satz von Fejer ist der Weierstraß'sche[20] Approximationssatz über die gleichmäßige Approximation stetiger Funktionen durch Polynome [35, S.82].

2.2.9 Beispiele

Drei der wichtigsten Funktionenformen, die aufgrund ihrer Symmetrie relativ einfach berechnet werden können sind die Rechteck-, Dreieck- und Sägezahnfunktion. Ausführlich soll hier die Rechteckfunktion behandelt werden, da anhand dieser auch das entwickelte Programm erklärt wird. Die anderen beiden Funktionen werden analog ermittelt.

---

[15] Deutscher Mathematiker, 1831-1889

[16] Schwedischer Mathematiker, *1928

[17] Man nennt eine Funktionenfolge f_n *punktweise kovergent*, wenn $f_n(x)$ $\forall x$ aus dessen Definitionsbereich gegen den Funktionswert $f(x)$ einer Grenzfunktion f konvergiert [20].

[18] Man nennt eine Funktionenfolge f_n *gleichmäßig konvergent* gegen eine Grenzfunktion, wenn diese unabhängig vom Funktionsargument gegen die Grenzfunktion konvergiert [9].

[19] Ungarischer Mathematiker, 1880-1959

[20] Karl Weierstraß, deutscher Mathematiker, 1815-1897

Rechteckfunktion

Sei $f(t)$ gegeben als ungerade, 2π-periodische Funktion, mit Amplitude 1:

$$f(t) = -1, \quad -\pi < t < 0$$
$$f(t) = 1, \quad 0 < t < \pi$$

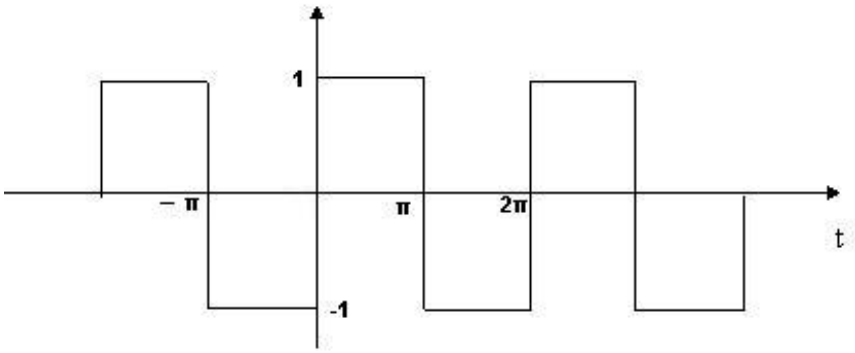

Abbildung 2.14: Rechteckfunktion

Da die Funktion ungerade ist, sind die Fourier - Koeffizienten $a_n = 0$. Da keine Verschiebung in Ordinatenrichtung vorliegt, ist der Gleichanteil $\frac{a_0}{2}$ ebenso 0. Die Berechnung der Fourier - Reihe ergibt nun:

$$\begin{aligned} f(t) =& \frac{4}{\pi}\left(\sin t + \frac{1}{3}\sin(3t) + \frac{1}{5}\sin(5t) + \frac{1}{7}\sin(7t) + \cdots\right) \\ =& \frac{4}{\pi}\sum_{n=1}^{\infty}\frac{\sin\left((2n-1)\omega t\right)}{2n-1} \end{aligned} \tag{2.6}$$

Man sieht bei der Berechnung der Fourier - Reihe, dass die Rechteckfunktion durch Überlagerung von unendlich- vielen ungeraden harmonische Oberschwingungen zur Grundschwingung entsteht. Die Amplitude dieser Oberschwingungen nimmt dabei mit steigender Frequenz ab [8]. Abbildung 2.15 visualisiert diesen Zusammenhang.

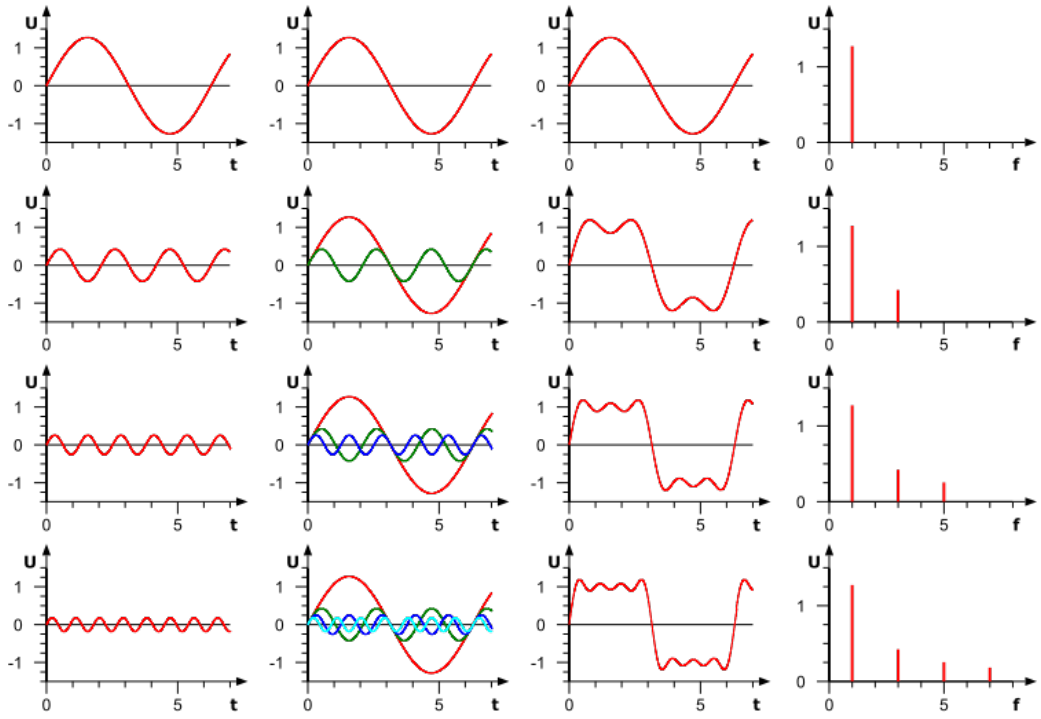

Abbildung 2.15: Synthese der Rechteckfunktion

In der Grafik (erste Spalte) ist ersichtlich, welche Schwingung hinzugefügt wird. In der zweiten Spalte sind alle zu überlagernden Schwingungen eingezeichnet, was in der dritten Spalte passiert. Die vierte Spalte gibt das Amplitudenspektrum der superpositionierten Schwingung wieder.

Um praktisch besser mit Fourier - Reihen hantieren zu können, wird an dieser Stelle eine etwas allgemeinere Form der Reihe für die Funktion angegeben. h bezeichnet die Amplitude der Funktion, $\omega = \frac{2\pi}{T}$ mit T als Periodendauer.

$$\begin{aligned} f(t) =& \frac{4h}{\pi}\left(\sin\omega t + \frac{1}{3}\sin(3\omega t) + \frac{1}{5}\sin(5\omega t) + \frac{1}{7}\sin(7\omega t) + \cdots\right) \\ =& \frac{4h}{\pi}\sum_{n=1}^{\infty}\frac{\sin\left((2n-1)\omega t\right)}{2n-1} \end{aligned} \tag{2.7}$$

Wenn man diese Funktion um $\frac{T}{4}$ nach links verschiebt, so wird aus dieser ungeraden Funktion eine gerade Funktion und die Fourier - Reihe wird gebildet durch:

$$\begin{aligned} f(t) =& \frac{4h}{\pi}\left(\cos\omega t - \frac{1}{3}\cos(3\omega t) + \frac{1}{5^2}\cos(5\omega t) - \frac{1}{7^2}\cos(7\omega t) + \cdots\right) \\ =& \frac{4h}{\pi}\sum_{n=1}^{\infty}(-1)^{n-1}\frac{\cos\left((2n-1)\omega t\right)}{(2n-1)^2}. \end{aligned} \tag{2.8}$$

Dreieckfunktion

Sei $f(t)$ gegeben als ungerade T-periodische Funktion, mit Amplitude h:

$$\begin{aligned} f(t) &= t, \quad -\frac{T}{4} < t < \frac{T}{4} \\ f(t) &= -t, \quad \frac{T}{4} < t < \frac{3T}{4} \end{aligned}$$

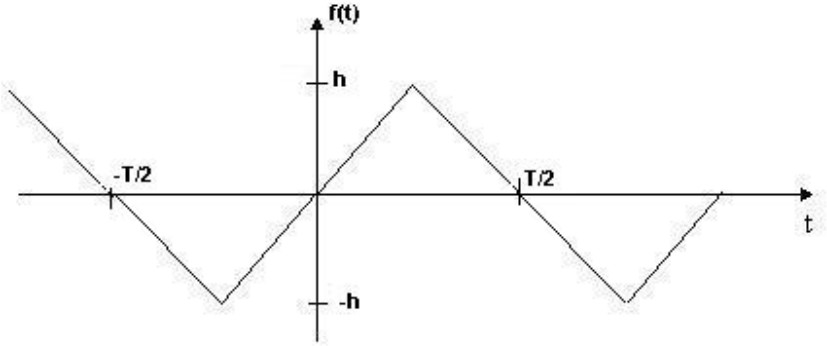

Abbildung 2.16: Dreieckfunktion

Die zugehörige Fourier - Reihe ist dann ($a_0 = a_n = 0$):

$$\begin{aligned} f(t) =& \frac{4h}{\pi}\left(\sin\omega t - \frac{1}{3^2}\sin(3\omega t) + \frac{1}{5^2}\sin(5\omega t) - \frac{1}{7^2}\sin(7\omega t) + \cdots\right) \\ =& \frac{4h}{\pi}\sum_{n=1}^{\infty}(-1)^{n-1}\frac{\sin\left((2n-1)\omega t\right)}{(2n-1)^2}. \end{aligned} \tag{2.9}$$

Wenn man diese Funktion um $\frac{T}{4}$ nach links verschiebt, so wird aus dieser ungeraden Funktion eine gerade Funktion mit vereinfachter Fourier - Reihe:

$$\begin{aligned} f(t) =& \frac{4h}{\pi}\left(\cos\omega t + \frac{1}{3^2}\cos(3\omega t) + \frac{1}{5^2}\cos(5\omega t) + \frac{1}{7^2}\cos(7\omega t) + \cdots\right) \\ =& \frac{4h}{\pi}\sum_{n=1}^{\infty}\frac{\cos\left((2n-1)\omega t\right)}{(2n-1)^2}. \end{aligned} \tag{2.10}$$

Sägezahnfunktion (steigend)

Sei $f(t)$ gegeben als ungerade T-periodische Funktion, mit Amplitude h:

$$f(t) = t, \quad -\frac{T}{2} < t < \frac{T}{2}$$

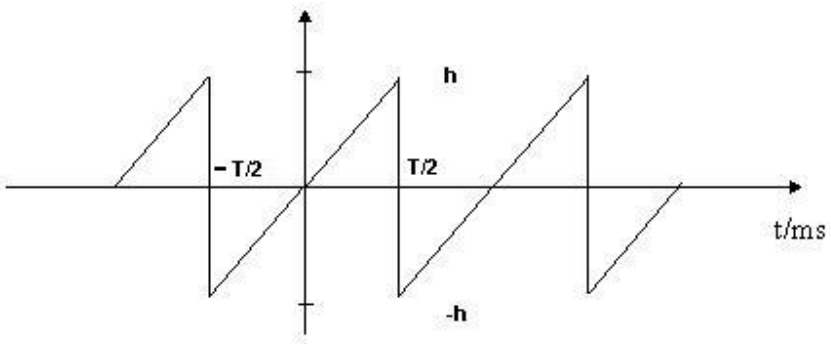

Abbildung 2.17: Sägezahnfunktion

Die zugehörige Fourier - Reihe berechnet sich dann zu ($a_0 = a_n = 0$):

$$\begin{aligned} f(t) =& -\frac{2h}{\pi}\left(\sin\omega t + \frac{1}{2}\sin(2\omega t) + \frac{1}{3}\sin(3\omega t) + \cdots\right) \\ =& -\frac{2h}{\pi}\sum_{n=1}^{\infty}\frac{\sin\left(n\omega t\right)}{n}. \end{aligned} \tag{2.11}$$

2.3 Grundlagen Informatik

Der im vorherigen Kapitel dargestellte Sachverhalt soll in dieser Arbeit in ein Java- Programm (**FourierApp**) implementiert werden. Ziel ist es dabei, mittels einer graphischen Benutzeroberfläche, die sich durch Koeffizientenmodifikation ergebenden Schwingungsänderungen zu veranschaulichen. Das notwendige Know- How wird in diesem Kapitel dargelegt.

2.3.1 Java- Grundlagen

Java ist eine von Sun Microsystems entwickelte objektorientierte[21] Programmiersprache, mit welcher u.a. einfach Animationen und Interaktionen im Internet realisiert werden können [34, S. 15]. Es zeichnet sich durch einen leichteren Gebrauch als ähnliche Programmiersprachen wie C++ oder C# aus, da der Sprachumfang kleiner ist und beispielsweise Mehrfachvererbungen nicht möglich sind [16].

Da es für die Entwickler von Java ein Hauptanliegen ist, Plattformunabhängigkeit und Internet- Fähigkeit zu vereinen, erzeugt der Java- Compiler aus dem Quellcode einen sogenannten Bytecode, welche unabhängig von der Rechnerarchitektur auf jedem Rechner gelesen werden kann, wenn auf diesem Rechner die Java Virtual Machine (JVM) vorhanden ist. Diese ist heute für jedes Betriebssystem verfügbar und meist auch installiert.

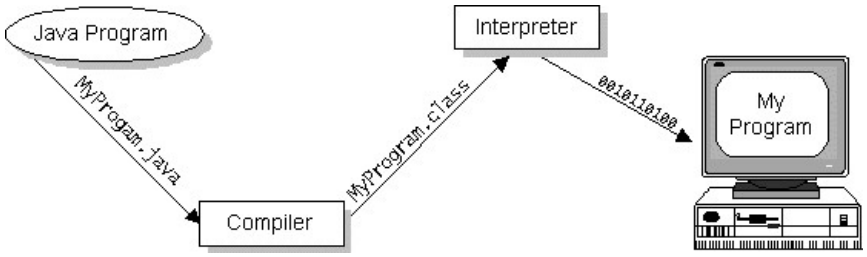

Abbildung 2.18: Übersetzung und Ausführung eines Java- Programms

Abbildung 2.18 veranschaulicht die Übersetzung und Ausführung eines Java- Programms. Dabei wandelt der Compiler den Quelltext in den Bytecode („MyProgram.class“) um und die JVM übernimmt die Rolle des Interpreters, indem sie das class- File in die auf dem Betriebssystem verstandene Maschinensprache übersetzt.

Java- Programmierung

Ein Java- Programm kann in jedem beliebigen Text- Editor oder jeder Entwicklungsumgebung geschrieben werden. Um kompiliert werden zu können, muss hierbei die Endung

[21] Die Grundidee des objektorientierten Programmierens ist die, Daten und Funktionen, die darauf angewandt werden können, eng in einem sogenannten Objekt zusammenzufassen [19].

„.java“ verwendet werden. Das Kompilieren kann dann zum Beispiel dadurch passieren, dass in der Konsole (Aufruf von „cmd“) folgender Befehl ausgeführt wird:

javac Beispiel.java

Aufgerufen wird das kompilierte Programm (Beispiel.class- File) schließlich mit:

java Beispiel.class

Um einen ersten Eindruck über eine Programmiersprache zu erhalten, wird üblicherweise auf einfachste Weise der Text „Hello World“ ausgegeben. In Java passiert dies so, dass zunächste eine Datei mit dem (frei- gewählten) Namen „HelloWorld.java“ erzeugt wird und in dieser folgender Quelltext eingefügt wird:

```
public class HelloWorld {

    public static void main (String[] args) {

        System.out.println(„Hello World!“);

    }
}
```

In der ersten Zeile wird durch „public class HelloWorld“ bekanntgegeben, dass hier eine Klasse beginnt (durch geschwungene Klammern wird diese begrenzt). Eine *Klasse* ist gewissermaßen der Plan eines Objekts. Man stelle sich beispielsweise den Bauplan eines Hauses vor, mit dem verschiedene Häuser gebaut werden können, aber alle als Ausgangspunkt diesen Bauplan haben. Eine Klasse beschreibt analog dazu ein Objekt, stellt aber kein eigenes Objekt dar. Wenn im Weiteren ein solches Objekt instanziert[22] wird (man spricht auch von Instanzen der Klasse), so steht es der Applikation zur Verfügung und erlaubt das Verwenden der Funktionen, Variablen etc. des Objekts bzgl. der Klasse [29].

Als nächstes wird durch „public static void main (String[ ] args)“ bekanntgegeben, dass hier die Funktion (oder Methode) „main“ beginnt[23]. Diese ist bei jeder Applikation notwendig und wird aufgerufen, sobald die Applikation aufgerufen wurde. Das Schlüsselwort „public“ sagt aus, dass von „außen“ (zum Beispiel wenn eine Instanz in einem anderen Programmteil / in eineranderen Klasse erstellt worden ist) auf diese Funktion zugegriffen werden darf. Durch Verwendung von „static“ wird festgelegt, dass die Klasse ohne Objektinstanz existieren kann, „void“ bedeutet, dass die Funktion keinen Rückgabewert hat.

[22] Als *Instanziierung* bezeichnet man in der objektorientierten Programmierung das Erzeugen eines Objekts einer bestimmten Klasse. [11]

[23] Die Funktion „main“ wird immer mit den Schlüsselwörtern „public static void“ eingeleitet..

Die Parameter der Funktion werden im String- Array[24] „arg“ gesammelt und können beim Programmaufruf bestimmt werden, wenn zum Beispiel die Beispiel.class- Datei durch

java Beispiel.class Parameter1 Parameter2 $\cdots$

aufgerufen wird.

In der main- Funktion wird schließlich die Ausgabe von „Hello World“ durch den Befehl „System.out.println(„Hello World!“);“ erzwungen. „println“ ist die Funktion zum Ausgeben. Dem Compiler muss dabei auch die Hierarchie des Befehls angegeben werden. Verschiedene Ebenen werden durch einen Punkt abgegrenzt.

Jedes Java- Programm kann eine oder mehrere Klassendefinitionen enthalten. Wenn eine Instanz von einer Klasse (allerdings nicht diejenige, die in der die main- Methode vorkommt) gebildet werden soll, so passiert das mit dem Befehl „new“.

Eine einfache Klasse ist beispielsweise gegeben durch[25]:

```
public class Point {

   public int x, y, color;

   public void moveTo(int newX, int newY) {

      x = newX;
      y = newY;

   }

   public void moveRel(int dX, int dY) {

      x += dX;
      y += dY;

   }
```

Dieses Beispiel enthält drei Integer[26] „x“,“y“ und „color“, welche die Koordinaten und die Farbe eines Punktes speichern. Die beiden Funtionen dienen zum absoluten und relativen Verschieben des Punktes [38].

[24] Mit Hilfe eines Arrays ode Feldes werden Daten eines Datentyps im Speicher so abgelegt, dass diese Daten über einen Index wieder angesprochen werden können [5].

[25] Eine Speicherung des Quelltextes in der Datei „Point.java“ ist notwendig.

[26] Mit *Integer* werden in der Informatik Objekte bezeichnet, welche ganze Zahlen speichern können [12].

In einer anderen Klasse kann nun mittels

Point myPoint;
myPoint=new Point();

ein neues Objekt mit Namen „myPoint“ instanziert werden. Zugriff erhält man auf die Variable „color“ etwa durch

myPoint.color

Soll der Punkt nach $\langle 2, 3 \rangle$ verschoben werden, so muss folgender Befehl ausgeführt werden:

myPoint.moveTo(2,3);

Soll derPunkt relativ um ein Pixel in x- und y-Richtung verschoben werden, so muss folgender Befehl ausgeführt werden:

myPoint.moveRel(1,1);

Zu den aktuellen Pixel- Koordinaten wir ein Pixel dazuaddiert.

2.3.2 Applets

Wie im vorigen Kapitel gezeigt wurde, kann ein Java- Programm unter anderem als eigenständig- laufendes Programm (Applikation) erstellt werden. Eine weborientierte Java-Implementierung stellt das Applet dar. Während eine Applikation ohne die Kontrolle eines Browser ausgeführt werden kann, ist ein *Applet* ein Programm, welches explizit in einem Webbrowser ausgeführt wird und die Interaktion mit dem Anwender / der Anwenderin erlaubt, ohne Daten über die Leitung zum Server versenden zu müssen [15]. Applets sind aus Sicherheitsgründen der uneingeschränkte Zugriffe auf Ressourcen des Rechners untersagt. Deshalb laufen diese in einer gesicherten Umgebung („Sandbox“), für welche der Anwender / die Anwenderin selber die Sicherheitsrichtlinien festlegen kann. Um die Herkunft eines Applets eindeutig zu bestimmen, kann dieses mit einer digitale Signatur versehen werden, damit durch Überprüfung dieser die Vertraulichkeit in Frage gestellt werden kann [33].

Java- Applets werden in der Regel von HTML- Seiten aufgerufen. Ein Webseite, in der ein Applet aufgerufen wird, kann etwa folgende Form haben:

```
<html>
<head>
<title>Java-Applet</title>
</head>
<body>
<applet code=„applet.class“ width=„640“ height=„200“>
</applet>
</body>
</html>
```

Auf die einzelnen Tags (welche durch spitze Klammern begrenzt sind) wird hier nicht eingegangen (siehe hierfür etwa [1]). Wenn nun diese html- Datei im Browser geöffnet wird, so erscheint ein (640x200 Pixel großes) Fenster, in welchem das Applet ausgeführt wird.

Applet- Programmierung

Bei Applets wird im Gegensatz zu Applikationen keine Methode „main“ verwendet, die bei Programmstart aufgerufen wird. Bei der Applet- Programmierung sind folgende Methoden zu verwenden:

- init() - Diese Funktion wird wenn das Applet erstmals in den Browser geladen wird genau einmal aufgerufen.
- start() - Die Methode „start“ wird aufgerufen, wenn das Applet sichtbar ist.
- paint() - Ermöglicht das Zeichnen im Applet und wird aufgerufen, wenn ein Neuzeichnen des Applets notwendig ist (wenn etwa das Browser- Fenster verschoben wird).
- stop() - Wird aufgerufen, wenn das Applet nicht sichtbar ist.
- destroy() - Die Funktion „destroy“ wird aufgerufen, wenn das Applet geschlossen wird [28].

Applet- Programmierung

Applet- Programmierung ist der Applikations- Programmierung sehr ähnlich, bis auf den Umstand, dass keine „main“- Funktion vorhanden ist. Um das „Hello World“- Beispiel zu realisieren benötigt man nur die paint- Funktion, da diese beim Aufruf des Applets aufgerufen wird:

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

public class HelloApp extends Applet {

   private String s = „Hello World!“;

   public void paint (Graphics g) {

      g.drawString (s, 25, 25);

   }
}
```

Mit dem Befehl „import“ werden zunächst Bibliotheken eingefügt, um den Befehlssatz des Compilers zu erweitern. Bei der Klassen- Definition wird „extends Applet“ angefügt: wenn ein Applet programmiert werden soll, ist dieser Zusatz notwendig. Die nächste Zeile definiert einen String („Hello World!“) und in der darauffolgenden Funktion paint[27] wird der String durch den Befehl „drawString“ ausgegeben.

Für Java existieren mehrere Sprachpakete für die Programmierung grafischer Benutzeroberflächen (z.B. Swing, AWT). In dieser Arbeit wird AWT verwendet, da in diesem die für die Implementierung notwendigen Sprachbefehle vollständig vorhanden sind.

[27] der Parameter „Graphic g“ ist notwendig, um auf das Zeichen- Objekt zuzugreifen

3 Realisierung

3.1 Forderungen

Um das Ziel der flexiblen Fourier - Analyse/Synthese zu erreichen, werden folgende Forderungen an die Software gestellt:

Es soll

- eine Grundfrequenz einstellbar sein,
- eine Signalform ausgewählt werden können, die dann visualisiert wird,
- einerseits ausgehend vom konstanten Signal $y(t) = 0$ durch Änderung der Fourier - Koeffizienten,
- andererseits durch Eingabe der Fourier - Reihe (durch Eingabe der Fourier - Koeffizient, Gleichanteil und Dämpfungsfaktor[1]) ein Signal generiert werden können.

Weiters soll es jederzeit während des Programmablaufs möglich sein, das Signal anzuhören, die Anzahl der Fourier - Koeffizienten zu erhöhen oder diese abzulesen und zu ändern.

[1] vgl. 3.3.4

3.2 Bedienungsanleitung des Programms FourierApp

Die Software **FourierApp** ist online unter

`http://web.student.tuwien.ac.at/~e0325815/dipa/FourierApp.html`

verfügbar. Getestet wurde unter *Mozilla Firefox 3.6.17* und *Microsoft Internet Explorer 6.0*. Unter Firefox ist ein Neustart des Browsers notwendig, wenn die Webseite aktualisiert wird. Dies hängt wahrscheinlich mit den Sicherheitseinstellung des Browsers zusammen.

Nach Aufruf der Webseite erscheint folgendes Bild:

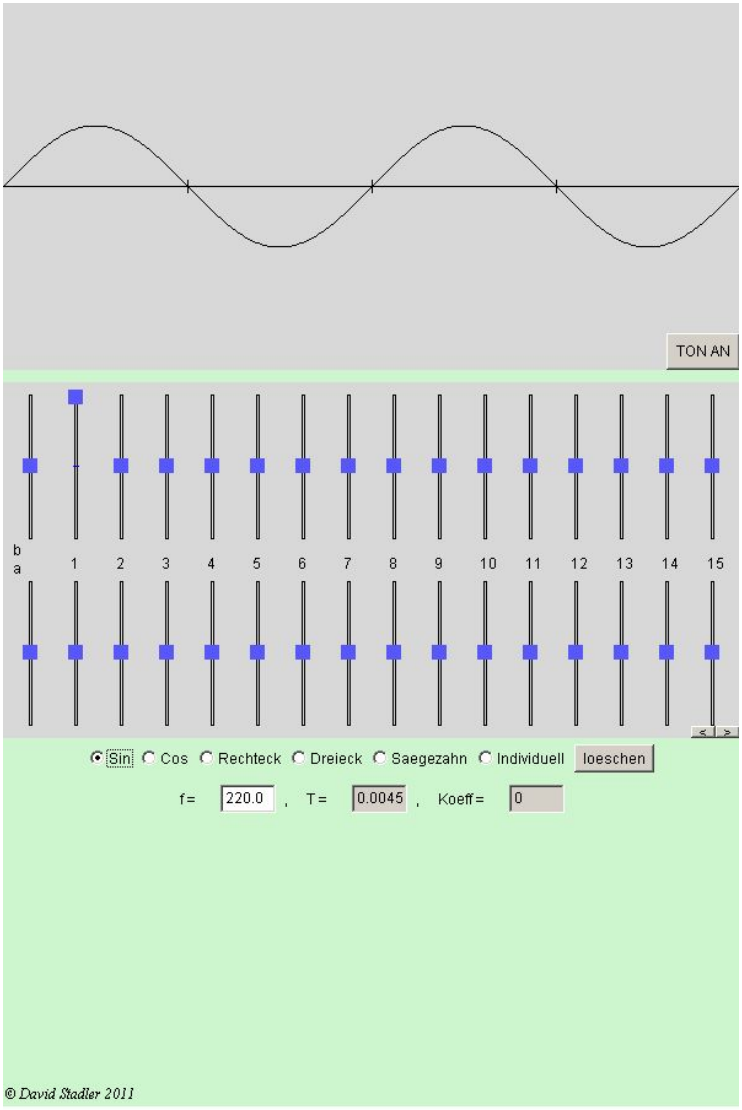

Abbildung 3.1: Ausgangsgrafik

Die Benutzeroberfläche ist gedrittelt. Im ersten Drittel ist der Graph einer Funktion und ein Button „TON AN“ zu sehen. Im zweiten Drittel ist eine Schiebereglersteuerung mit zwei kleinen Buttons rechts unten und im letzten Drittel sind verschiedene Checkboxen, Textfelder und ein Button mit Namen „loeschen“.

3.2.1 Graph- Bereich

In diesem Bereich wird der Graph dargestellt. Bis auf den Button „TON AN“ sind dem Benutzer / der Benutzerin hier keine Interaktionsmöglichkeiten gegeben. Wird der Button „TON AN“ betätigt, hört man das dargestellte Signal per Soundkarte und Audio- System. Per erneutem Drücken verstummt das Signal wieder.

3.2.2 Regler- Bereich

In diesem Bereich ist dargestellt, welche Gewichtung der aktuelle Fourier - Koeffizient im Signal hat, wenn man den Mauszeiger über die Schieberegler zieht: das Textfeld mit Namen „Koeff“ nimmt den jeweiligen Wert an (siehe 3.2.3). Als Bereich ist hier $[-1, 1]$ zulässig. Per Mausklick auf einen Regler und anschließendem Ziehen der Maus kann der jeweilige Fourier - Koeffizient geändert werden. Dies wirkt sich unmittelbar auf die dargestellte Schwingung aus.

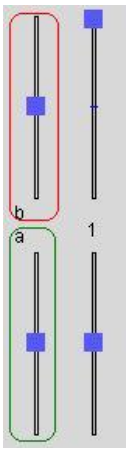

Abbildung 3.2: Niedrige Fourier - Koeffizienten

Die Nummerierung gibt den jeweiligen Fourierkoeffizienten an. a_0, welcher nicht beschriftet ist, aber in Abbildung 3.2 grün markiert ist, definiert den Gleichanteil, darüber (in Abb. 3.2 rot gekennzeichnet) ist eine fiktiver Fourier - Koeffizient „b_0“ vorhanden, mit welchem die Amplitude geändert werden kann.

Rechts unten sind zwei kleine Buttons vorhanden, mit denen die Anzahl der Fourier - Koeffizienten geändert werden kann.

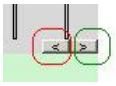

Abbildung 3.3: Koeffizientenanzahl ändern

Der rot- umrandete Button in Abbildung 3.3 dient der Verringerung, der grün- umrandete der Erhöhung der Fourier - Koeffizienten.

3.2.3 Eingabe- Bereich

Die erste Zeile des Eingabe- Bereichs ermöglicht die Auswahl einer Funktion durch Auswahl einer Checkbox, die den Namen dieser Funktion hat. Per Klick auf eine Checkbox wird das Signal im Graph- Bereich sichtbar und die Fourier - Koeffizienten werden im Regler- Bereich angepasst.

Wenn ein Signal ausgewählt und nachträglich ein Fourier - Koeffizient variiert wird, so wird automatisch die Checkbox „Individuell“ ausgewählt.

Darunter geben drei Textfelder den Wert der Frequenz[2], Periodendauer und, wenn die Maus über einen Schieberegler positioniert ist, den Wert des Fourier - Koeffizienten an, wobei der Anwender / die Anwenderin nur die Frequenz ändern kann, die Periodendauer wird automatisch ermittelt.

Per Druck auf „loeschen“ werden alle Fourier - Koeffizienten auf 0 gesetzt und es erscheint ein neues Fenster:

3.2.4 Erweiterter Eingabe- Bereich

Im erweiterten Eingabe- Bereich kann der Anwender / die Anwenderin selbst Fourier - Synthese vollziehen. Durch manuelle Eingabe der berechneten Fourier - Reihe kann diese visualisiert bzw. akustisch wiedergegeben werden.

[2]Die Anzeige im Graph- Bereich ist so konzipiert, dass Sinus- Signale zwischen 50 und einigen 1000Hz, sowie Amplituden bis etwa 1.5 gut dargestellt werden können.

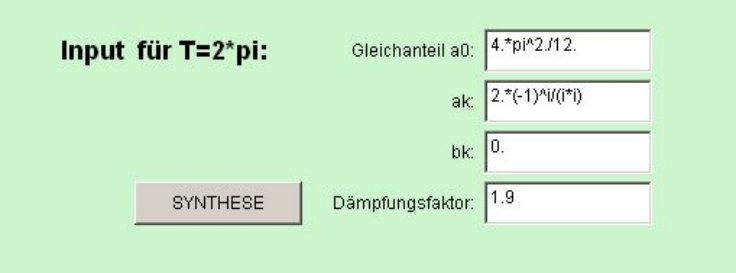

Abbildung 3.4: Erweiterter Eingabe- Bereich

Im ersten Textfeld von Abbildung 3.4 ist der Gleichanteil einzugeben, im zweiten/dritten Textfeld der Anteil des jeweiligen Fourier - Koeffizienten und im vierten Textfeld kann ein Dämpfungsfaktor eingestellt werden, damit Signale mit zu hoher Amplitude besser dargestellt werden können.

Nachdem alle Eingaben getätigt wurden, kann per Druck auf „SYNTHESE“ das Signal generiert werden.

Die Syntax zur Eingabe ist folgende:

Wert	Kommentar
1, 2.0, 3.3, 1/2 ...	Eine Zahl ist als ganze Zahl, Kommazahl oder in Form eines Bruches zu schreiben.
^	Hochzahlen sind mit vorangehenden ^ zu bezeichnen.
pi	Die Konstante π muss klein geschrieben werden.
i	Als Laufvariable muss „i“ verwendet werden.
+ - * /	Addieren, Subtrahieren, Multiplizieren und Dividieren.
( )	Gruppierungen werden mit runden Klammern bewerkstelligt.

3.3 Implementierung der Software

Um die Software **FourierApp** möglichst flexibel anwenden zu können und die volle Stärke des Applets auszulasten, ist grundsätzlich eine Implementierung ins World Wide Web notwendig. Als Programmiersprache zur Realisierung eignet sich daher Java, da der vom Compiler erzeugte Bytecode, der im Weiteren in einer speziellen Laufzeitumgebung ausgeführt wird, plattformunabhängig und dadurch für alle geeigneten Rechnerarchitekturen und Betriebssysteme exekutierbar ist (vgl. 2.3.1) [17].

3.3.1 Das Java- Grundgerüst

Die Software besteht aus zwei Klassen FourierApp.java (siehe S.59) und FourierHelp.java(siehe S. 94). In FourierApp.java wird das Applet implementiert, FourierHelp.java dient der Speicherung der Fourier - Koeffizienten und der Ton- Wiedergabe. Grundlage für die Software ist ein bereits publizierter Sinusoszillator [31]. Von diesem Programm wurde das Grundgerüst übernommen und für diese Arbeit erweitert und angepasst.

FourierApp.java

Die wichtigsten Variablen in FourierApp.java:

- Checkbox sinus, cosinus, $\cdots$ - Die Checkboxen zur Auswahl einer vorprogrammierten Signalform.
- Button loeschen - Dieser Button dient zum Löschen des Signals.
- Button soundButton - Mit diesem Button kann die Tonausgabe gestartet werden.
- Button syntheseButton - Wenn auf diesen Button gedrückt wird, so werden die manuell eingegebenen Fourier - Koeffizienten in eine Fourier - Reihe umgewandelt und als Graph dargestellt.
- Button mehrKoeff/wenigerKoeff - Hiermit kann die Anzahl der Fourier - Koeffizienten geändert werden.
- TextField frequenz, periodenDauer, koeff, indiGleichanteil, indiKoeff_ak, indiKoeff_bk, indiZoom: Textfelder in denen Text eingegeben werden kann, beispielsweise für die Frequenz, Gleichanteil etc.
- double freq - Speichert die Grundfrequenz.
- int reglerAbstandVonRand, x1, xRegler, x0, y2, ya, yb, yc, yd, yac, ddh = 10, h, h2, d1, dOffs, maxx, ycd, yyc, yaaa, ybbb, yccc, yddd, polX[ ], polY[ ], groesse, groesse2, yOffset, double yVal[ ] - Diese Variablen und Konstanten dienen hauptsaechlich der Positionierung der einzelnen Panels. Sie werden im Quelltext näher erläutert.

- int koeffAnz - Mit dem Integer „koeffAnz“ wird die Anzahl der Fourier - Koeffizienten bestimmt.
- int SIN,COS - Die beiden Variablen „SIN“ und „COS“ werden gesetzt, wenn sich der Mauszeiger bei den Reglern im Bereich der Sinus- Koeffizienten (b_k) oder Cosinus-Koeffizienten (a_k) befindet.
- FourierHelp f - Die Instanz „f“ vom Objekt „FourierHelp“ dient der Speicherung der Fourierkoeffizienten und der Tonausgabe.
- Panel p0,p1,p2,p3 - Im Panel[3] „p0“ werden „p1“,“p2“ und „p3“ eingebettet. Panel „p1“ dient zur Auswahl der Signalform, bzw. zum Löschen der Fourierkoeffizienten. Panel „p2“ dient zur Anzeige der Ausgabefrequenz, Periodendauer und des Gewichts der Fourierkoeffizienten. Panel „p3“ wird erst sichtbar, wenn „loeschen“ gedrückt wurde und der Anwender / die Anwenderin errechnete Fourier - Koeffizienten eintragen möchte.
- double PI = Math.PI - Die Konstante π.

Die wichtigsten Methoden in FourierApp.java:

- public void init() - Wie in 2.3.2 erwähnt, wird diese Methode aufgerufen, wenn das Applet gestartet wird, d.h. diese Funktion dient der Initialisierung des Applets. Hiebei werden grundlegende Einstellungen der Panels wie Größe und Position definiert. Weiters werden sämtliche grafischen Elemente und Interaktionsobjekte in den Panels hinzugefügt und für das Interagieren mit „addActionListener“[4] bereitgestellt.
- private void neustart() - Die Funktion „neustart“ wird ausgeführt, wenn fundamentale Änderungen z.B. in der Frequenz oder in der Koeffizientenanzahl vorgenommen wurden. Es wird uberprüft, welche Funktion ausgewählt ist (Rechteck-, Dreieck-, Sägezahnfunktion...), es werden die Funktionswerte errechnet und die Reglersteuerung neugestartet.
- private void ursprung() - Mit „ursprung“ wird die Reglersteuerung neugezeichnet. Dies wird z.B. benötigt, wenn ein anderes Signal ausgewählt wird und dadurch die Schieberegler angepasst werden müssen.
- public void paint(Graphics g) - „paint“ wird immer dann aufgerufen, wenn neugezeichnet werden muss, z.B. wenn das Fenster verdeckt war und wieder im Vordergrund angezeigt wird.

[3] Ein *Panel* ist eine Klasse zur Aufnahme anderer AWT-Komponenten (anderer grafische Elemente) [2].
[4] „addActionListener“ ist ein Ereignisempfänger, der beispielsweise auf Maustastenklicks in Buttons reagieren kann [2].

- public boolean action() - Die Funktion „action“ wird aufgerufen, sobald die enter-Taste gedrückt wird. In diesem Fall wird die Frequenz auf den Wert geändert, der im Textfield eingegeben wurde.
- public void itemStateChanged() - Die Methode „itemStateChanged“ wird aufgerufen, wenn eine andere als aktuelle ausgewählte Checkbox ausgewählt wurde.
- public void actionPerformed(ActionEvent arg0) - „actionPerformed“ wird aufgerufen, wenn ein Button gedrückt wurde.
- private String config(String text) - Die Funktion „config“ dient zum Parsen des übergebenen Textes. Es generiert aus dem String „i^j“ den String „pow(i,j)“, der in java verarbeitet werden kann.
- private void zeichneGraf() - „zeichneGraf“ definiert den Grafen neu. Dies inkludiert die Farbwahl, Berechnung der Fourierkoeffizienten und zeichnen der zugehörigen Fourier - Reihe.
- public boolean mouseMove(Event e, int x, int y) - Wird aufgerufen, wenn der Mauszeiger bewegt wird und dabei keine Maustaste gedrückt wurde.
- public boolean mouseDown(Event e, int x, int y) - Wird aufgerufen, wenn die Maustaste gedrückt wurde.
- public boolean mouseDrag(Event e, int x, int y) - Wird aufgerufen, wenn der Mauszeiger mit gedrückter Taste bewegt wurde.
- public boolean mouseUp(Event e, int x, int y) - Wird aufgerufen, wenn die Maustaste losgelassen wurde.
- public void update(Graphics g) - Die Methode „update“ wird aufgerufen, wenn ein Befehl zum neuzeichnen aufgerufen wird, z.B. durch repaint.
- private void prepareGBC() - „prepareGBC“ dient zur Voreinstellung der danachfolgenden Positionierung der Panels bei der Initialisierung.
- private void regler() - Die Funktion „regler“ dient zum Zeichnen der einzelne Regler der Schiebereglersteuerung.
- private void letsPlay() - Die Methode „letsPlay“ wird aufgerufen, um zu ermitteln, welche Checkbox akutell ausgewählt ist. Entsprechend werden die Fourierkoeffizienten im Objekt „f“ verschieden gesetzt.
- private void berechneDim() - Berechnet alle notwendigen Koordinaten für Panels, Schiebereglergrafik etc. (siehe Abbildung 3.6)
- public void destroy() - Die Funktion „destroy“ wird aufgerufen, wenn das Applet geschlossen wird. Eine allenfalls laufende Tonausgabe wird abgebrochen.

FourierHelp.java

Die wichtigsten Variablen in FourierHelp.java:

- int periode - Das Integer „periode“ speichert die Periodendauer.
- byte[ ] wiedergabe - Das byte- Array „wiedergabe“ dient der Tonausgabe des aktuell angewählten Signals.
- double amplitude - Die Variable „amplitude“ speichert die Amplitude der Funktion.
- double[ ] yWerte - Das double- Array „yWerte“ speichert die y- Werte des akutellen Signals.
- double omega=2.*Math.PI/periode - Definition des Kreisfrequenz.
- double wt - Ist eine Hilfsvariable fuer die Kreisfrequenz omega
- int maxKoeff = 100 - Hier wird das obere Limit der Fourier - Koeffizienten definiert.
- double[ ] ak, bk - Deklaration der Arrays, welche die a_ks und b_ks speichern.

Die wichtigsten Methoden in FourierHelp.java:

- public void grundFrequenz(double grundf) - Wenn in der Klasse „FourierApp“ die Ausgangsfrequenz geändert wird, so wird die Funktion „grundFrequenz“ aufgerufen. Hierbei wird die Periodendauer neu errechnet und die Arrays „yWerte“ und „wiedergabe“ neu deklariert. Weiters werden diese durch die Funktion „berechne“ definiert und die Tonausgabe beginnt neu.
- public void set_ak(int i,double val) - In der Funtion „set_ak“ werden die Fourierkoeffizienten (a_ks) in Abhängigkeit von Index und Wert gespeichert.
- public void set_bk(int i,double val) - In „set_bk“ werden analog die Fourierkoeffizienten (b_ks) gespeichert.
- public double get_ak(int i) - In der Funtion „get_ak“ werden die Fourierkoeffizienten (a_ks) in Abhängigkeit vom Index wiedergeholt.
- public double get_bk(int i) - In „get_bk“ werden analog die Fourierkoeffizienten (b_ks) wiedergeholt.
- public void start() - Wenn die Funktion „start()“ aufgerufen wird, soll die Tonausgabe beginnen.

- public void stop() - Wenn die Funktion „stop()“ aufgerufen wird, soll die Tonausgabe beendet werden.
- public void berechne() - Fügt alle uebergebenen a_ks und b_ks zusammen und erstellt ein Array „yWerte“ mit den entsprechenden, transformierten Werten von diesen Fourierkoeffizienten.
- public static byte int2ulaw(int ch) - Dient der Umwandlung der aktuellen Information in ein sound- Objekt.

3.3.2 Grafikelemente und Panelbereich

In Abbildung 3.5 sind die in 3.3.1 erwähnten Grafik- Elemente ersichtlich.

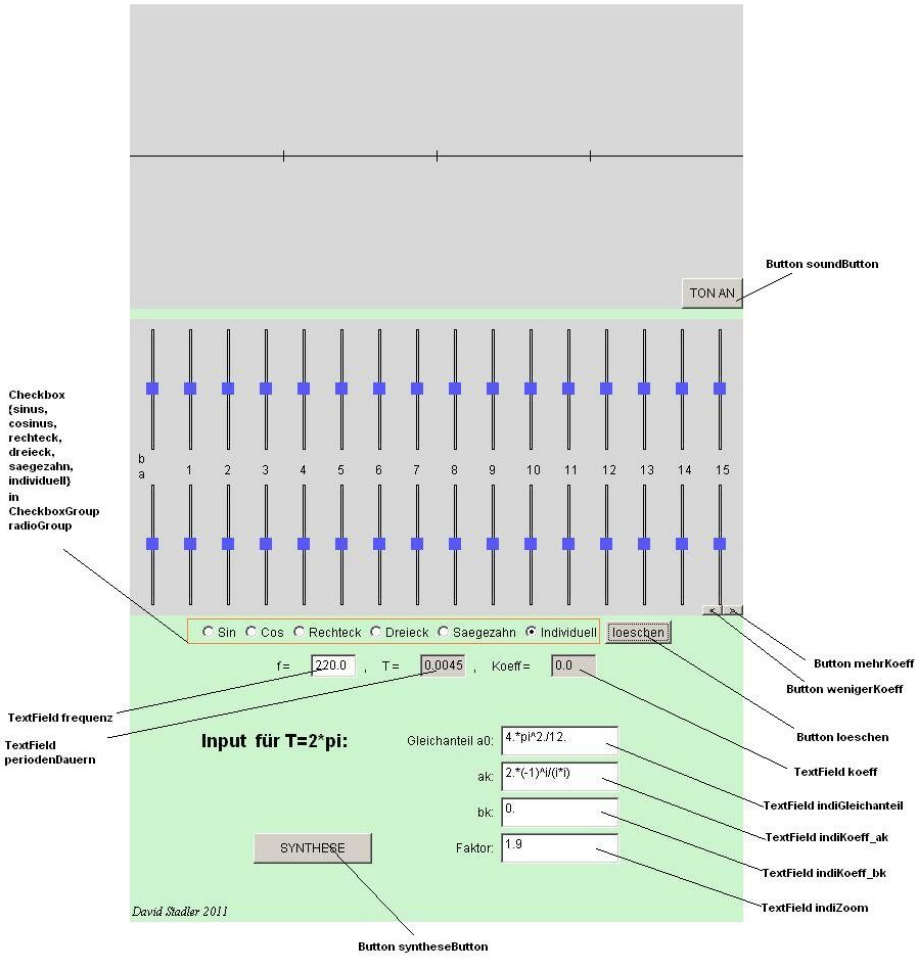

Abbildung 3.5: Interaktionselemente im Programm

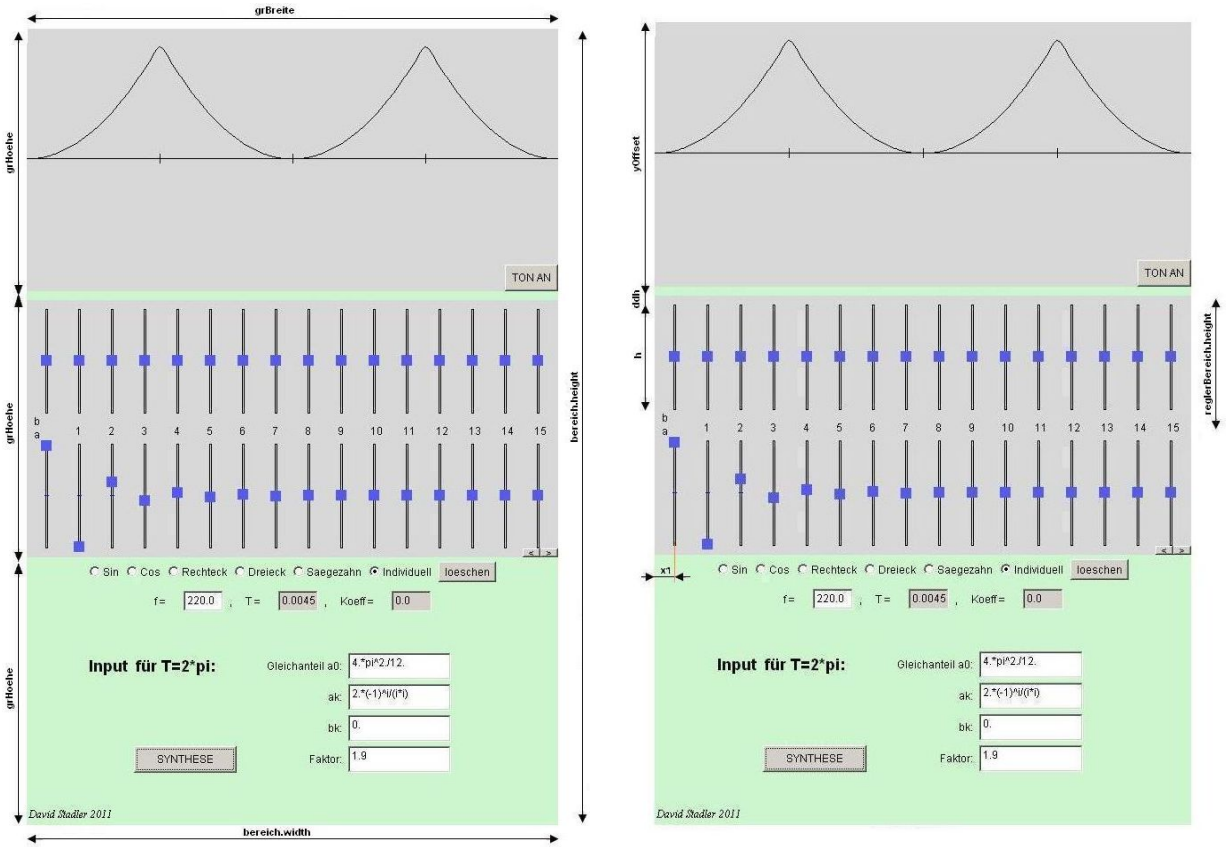

Abbildung 3.6: Variablen zur Darstellung von Grafikelementen im Hauptpanel

Die Abbildungen in 3.6 sollen die wichtigsten Variablen zur Positionierung im Hauptpanel wiedergeben.

Für die Schieberegler werden schließlich folgende Variablen verwendet:

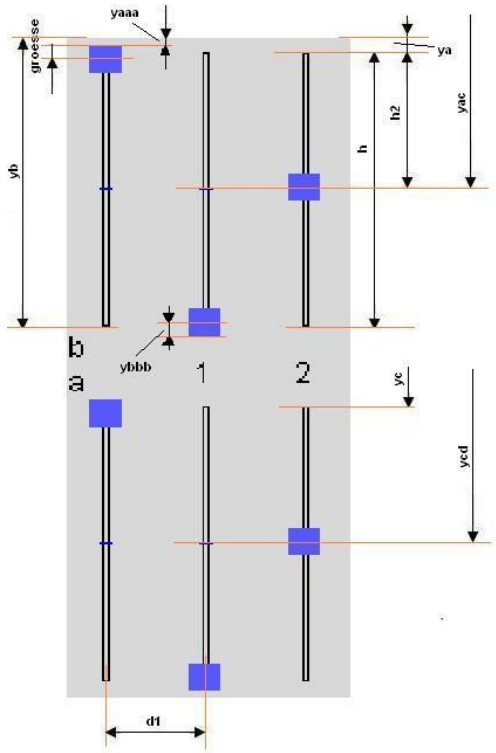

Abbildung 3.7: Variablen zur Darstellung von Grafikelementen im Steuerungspanel

3.3.3 Implementierung der Fourier - Analyse in Java

Bei der Rechteckfunktion lautete die Fourier - Reihe (vgl. 2.2.9):

$$f(t) = \frac{4h}{\pi}\left(\sin\omega t + \frac{1}{3}\sin(3\omega t) + \frac{1}{5}\sin(5\omega t) + \frac{1}{7}\sin(7\omega t) + \cdots\right)$$

Durch einen konstanten Faktor, multipliziert mit der Summe der Fourier - Koeffizienten $\sin\omega t, \frac{1}{3}\sin(3\omega t), \frac{1}{5}\sin(5\omega t)\cdots$ wird die Funktion gebildet. h definiert die Amplitude, $w = 2\pi f$. Man sieht hier, dass die Reihe nur von ungerade sin- Koeffizienten gebildet wird, allerdings mit unterschiedlicher Amplitude, da der Sinus mit dem Kehrwert des jeweiligen Index' des Fourier - Koeffizienten multipliziert wird. Die Spektralverteilung ist in Abbildung 2.10 dargestellt.

In der Software werden die Fourier - Koeffizienten der Rechteckfunktion im Wesentlichen durch folgenden Teil des Quelltexts berechnet:

```
for(int i=0;i<=koeffAnz;i++){
   if(i%2==1)
      f.set_bk(i,4.0*0.77/PI*1/i);
   else
      f.set_bk(i,0.);
   f.set_ak(i,0.);
}
```

Die erste Zeile definiert eine for- Schleife[5], die solange ausgeführt wird, bis die Laufvariable „i“ den Wert von koeffAnz erreicht hat. „i“ definiert den Index des jeweiligen Fourier - Koeffizienten und wird bei jedem Durchlauf um 1 erhöht. Damit nur die ungeraden Koeffizienten angesprochen werden, wird eine if- Anweisung auf i angewandt, welche nur True ist, wenn $i\%2 = 1$ ist (% steht für das Modulosymbol[6] [4]. In diesem Fall nimmt der behandelte Fourier - Koeffizient den Wert $\frac{Konstante}{i}$ an. h wurde mit 0.77 angenommen, damit die Grafik im weiteren gut dargestellt werden kann. Alle anderen Koeffizienten werden auf 0 gesetzt.

Alle Fourier - Koeffizienten sind nun im Objekt „f“ (als Instanz von „FourierHelp“) gespeichert. Es folgt als nächsten Schritt das neuzeichnen der Reglersteuerung. Dies passiert mit der Methode „ursprung“ (vgl. 3.3.1). Durch repaint, wird schließlich die Grafik neu gezeichnet. Dabei wird zunächst die Funktion „update“ gestartet, die ihrerseits „zeichneGraf“ aufruft. In „zeichneGraf“ wird „f.berechne()“ aufgerufen, was die Berechnung der Fourier - Reihe nach sich zieht. Dabei wird das Array „yWerte“ mit folgender Syntax zusammengestellt:

[5] Eine For-Schleife ist eine Struktur, mit der man eine Gruppe von Anweisungen (einen Block) mit einer bestimmten Anzahl von Wiederholungen ausführen kann [6].

[6] Die mathematische Operation „Modulo“ (auch „Division mit Rest“) besagt, dass es zu zwei Zahlen n und $m \neq 0$ eindeutig bestimmte Zahlen a und b gibt für die gilt: $n = a \cdot m + b\,, \quad 0 \leq b < m$ [3].

```
for (int j = 1; j < maxKoeff; j++)
   y += (ak[j]*Math.cos(wt*j)+bk[j]*
      Math.sin(wt*j));
```

Diese Schleife wird für jeden einzelnen x- Wert eine Periode (gespeichert als Variable „periode“) durchgeführt und „y“ wird schließlich in das Array „yWerte“ durch

```
yWerte[i]=y;
```

übergeben. Standardmäßig besteht dieses Array aus 150 Einträgen. Nun kann die Funktion mittels „bildgraf.drawPolygon“ gezeichnet werden.

Sinus-, Kosinus-, Dreieck- und Sägezahnfunktion werden analog konstruiert.

3.3.4 Fourier - Synthese in Java

Zur individuellen Eingabe von Fourier - Koeffizienten gibt es in **FourierApp** vier Textfelder:

1. Gleichanteil a0 - Hier kann der Gleichanteil eingegeben werden (z.B. 4.*pi^2./12.)
2. ak - Hier können in allgemeiner Form die Fourier - Koeffizienten a_k eingegeben werden. Beispiel: 2.*(-1)^i/(i*i)
3. bk - Analog zu ak.
4. Dämpfungsfaktor - Damit die Funktion besser sichtbar ist, kann diese hier gestaucht oder gestreckt werden.

Wenn alle Daten eingegeben wurden, kann per Druck auf „SYNTHESE“ die Fourier - Reihe konstruiert werden und wird sogleich angezeigt. Das passiert im Prinzip genauso wie in 3.3.3, nur wird keine vorgefertige Fourier - Reihe verwendet.

3.3.5 Kommentar zum musikalischen Einsatz

Von musikalischer Seite interessant ist der Zusammenhang von Abbildung 2.1. Hier ist ersichtlich, dass wenn man die ersten fünf und den siebten Oberton auf einmal klingen lässt einen Dur- Dreiklang über zwei Oktaven hören würde. Im Programm wird dieser Fall so bewerkstelligt, dass man das Signal löscht und b2[7]- b6 bzw. b8 durch Erhöhung der Amplitude auf 1 klingen lässt. Dieser Sachverhalt ist in Abbildung 3.8 dargestellt.

[7] b1 ist der Grundton

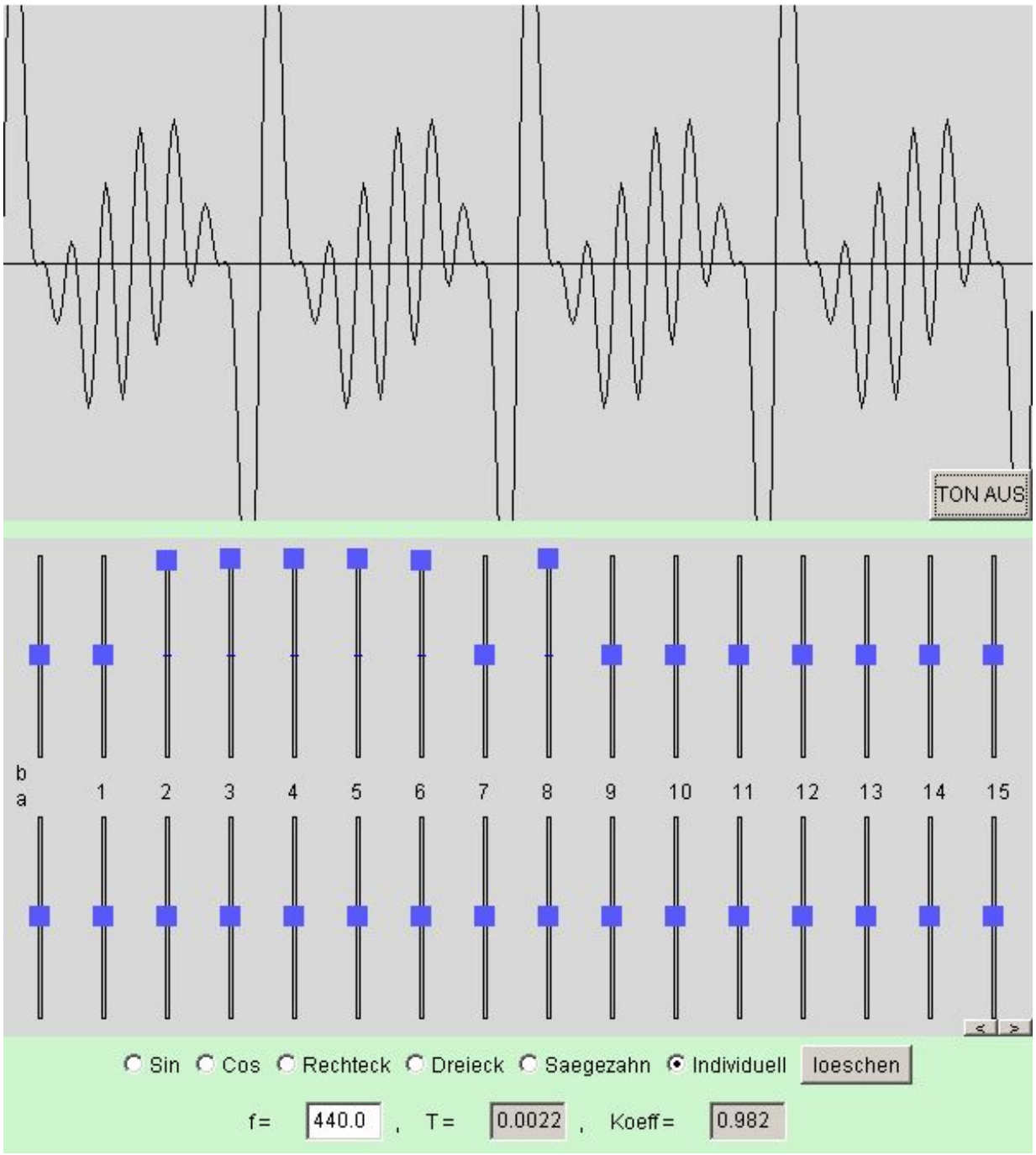

Abbildung 3.8: Dur- Dreiklang über 2 Oktaven

3.3.6 Einbinden der Software in eine Website

Um das Programm in eine Website einzubinden, sind folgende Schritte notwendig:

- Zu Beginn sind die beiden java- Dateien, sowie die jar- Packages jep-2.4.1.jar und rt.jar in den selben Ordner zu kopieren, wie in Abbildung 3.9 dargestellt.

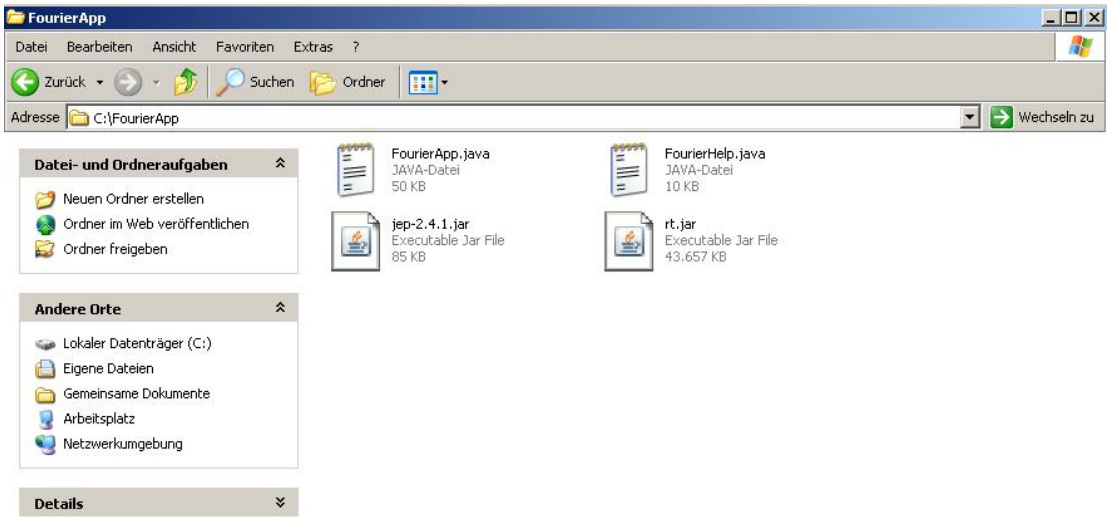

Abbildung 3.9: Alle relevanten Dateien in einem Ordner

- Danach müssen die beiden java- Dateien mit den notwendigen Packages kompiliert werden. In der Kommandozeile muss dabei in den relevanten Ordner gewechselt werden und schließlich der Befehl, welcher in Abbildung 3.10 dargestellt ist, aufgerufen werden [27].

```
C:\FourierApp>dir
 Volume in Laufwerk C: hat keine Bezeichnung.
 Volumeseriennummer: 0025-176E

 Verzeichnis von C:\FourierApp

31.03.2011  15:20    <DIR>          .
31.03.2011  15:20    <DIR>          ..
31.03.2011  15:23            14.314 FourierApp.class
29.03.2011  15:08            50.260 FourierApp.java
31.03.2011  15:23             2.508 FourierHelp.class
25.03.2011  23:37             9.226 FourierHelp.java
30.04.2007  11:44            86.296 jep-2.4.1.jar
12.11.2010  19:45        44.704.363 rt.jar
               6 Datei(en)     44.866.967 Bytes
               2 Verzeichnis(se),    869.789.696 Bytes frei

C:\FourierApp>javac -cp jep-2.4.1.jar;rt.jar *.java

C:\FourierApp>_
```

Abbildung 3.10: Kompilieren in der Kommando- Zeile

Es sind nun, wie in Abbildung 3.11 ersichtlich, alle relevanten Dateien in dem davor verwendeten Ordner.

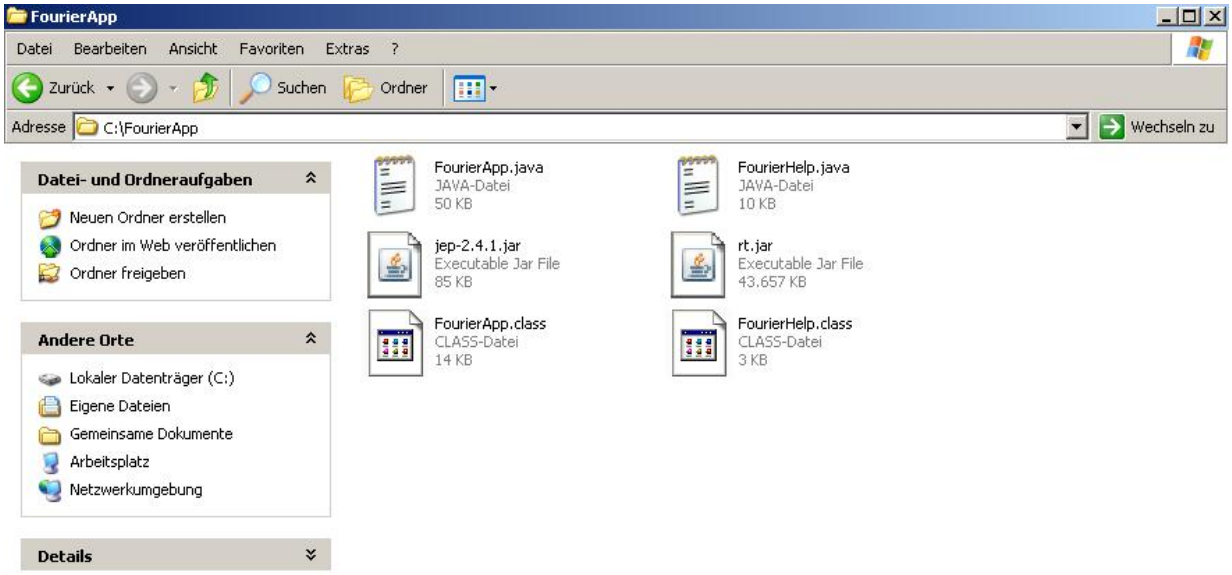

Abbildung 3.11: Alle relevanten Dateien in einem Ordner

- Als nächstes wird nun eine Webseite erstellt, in dem das Applet erscheinen soll. Diese Webseite kann zum Beispiel „FourierApp.html“ heißen. Der Quelltext in diesem File ist:

  ```
  <html>
  <body>
  <p><applet code = FourierApp.class
  archive="jep-2.4.1.jar,rt.jar"width=600 height=900>
  </applet></p>
  </body>
  </html>
  ```

  Durch Öffnen der erstellten html- Seite kann kontrolliert werden, ob die vorangegangen Schritte funktioniert haben.

- Als nächsten Schritt können nun alle Dateien im Ordner per FTP- Client[8] auf eine Website kopiert werden. Als Beispiel diene an dieser Stelle die Website des Autors[9].

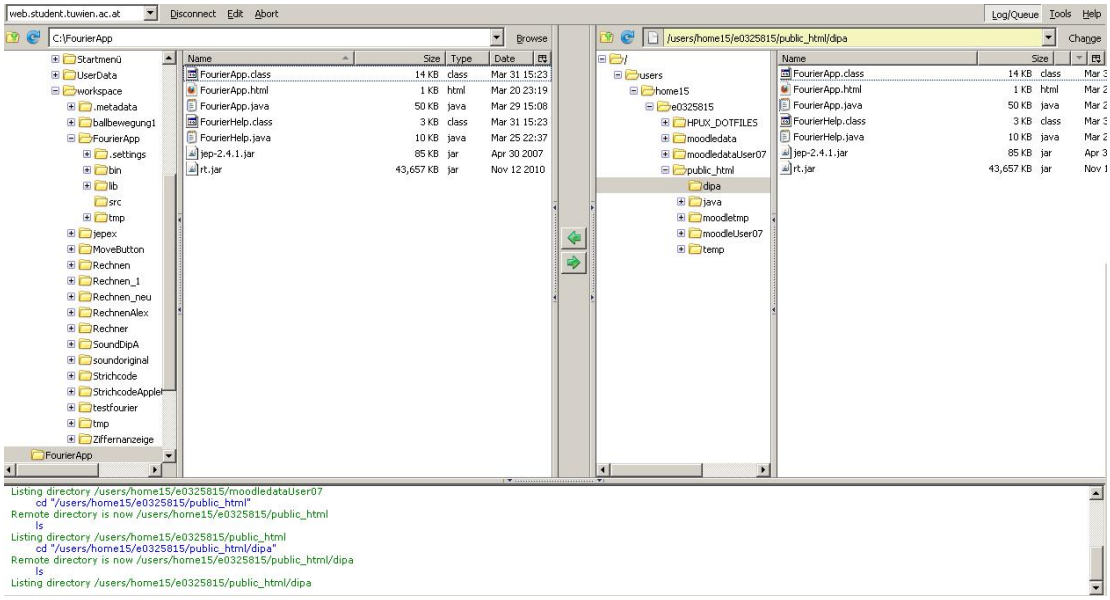

Abbildung 3.12: Alle relevanten Dateien in einem Ordner 2

Es brauchen für das Übertragen der Dateien nur Ausgangs- und Zielordner ausgewählt werden und dann die zu übertragenden Dateien selektiert werden (vgl. Abb. 3.12). Per Druck auf den grünen Pfeil, welche nach rechts zeigt, werden die Dateien auf die Website kopiert.

- Durch Öffnen von `http://web.student.tuwien.ac.at/~e0325815/dipa/FourierApp.html` kann nun das Applet gestartet werden.

[8] Der Autor verwendet hierfür den gratis- verfügbaren FTP- Client „FireFTP“, erhältlich unter `http://www.chip.de/downloads/fireFTP_15579287.html`.

[9] Erreichbar unter `http://web.student.tuwien.ac.at/~e0325815/`.

4 Diskussion

Der Zusammenhang zwischen Grundton und Obertönen eines Signals kann anhand der Gewichtung des jeweiligen Fourier - Koeffizienten dargestellt werden. Physik / Musik und Mathematik gehen konform, wenn es um die Ermittlung der Obertöne geht: es handelt sich dabei im Prinzip nur um Frequenzvervielfachung und Amplitudenänderung. Entsprechend einfach ist auf mathematische Weise auf die Fourier - Koeffizienten zu schließen. In der Programmiersprache Java ist dies gut umzusetzen, da ausreichend viele mathematische Prozeduren zur Verfügung stehen.

Der Aufwand zur Realisierung der Software **FourierApp** war recht hoch[1], da es mein Anliegen war, eine grafisch ansprechende Benutzeroberfläche zu entwickeln. Ich entschied mich gegen statische Positionierung und organisierierte die Panels abhängig von der verfügbaren Größe des Browser- Fensters. Der Aufwand an zu verwendenden Variablen stieg dadurch stark an, aber die Software ist dadurch flexibler einsetzbar.

FourierApp kann dazu verwendet werden, dynamisch die Fourier - Koeffizienten einer Funktion zu ändern. Die Änderungen der Funktion sind unmittelbar ersichtlich. Durch die akustische Begleitung sind Änderungen auch hörbar. Die Anzeige im Graph- Bereich ist so konzipiert, dass Sinus- Signale zwischen 50 und einigen 1000Hz, sowie Amplituden bis etwa 1.5 gut dargestellt werden können. Für einen eventuellen Einsatz im Schulunterricht ist diese Information unabdingbar, da natürlich 1000Hz[2] nicht problemlos in realer Weise dargestellt werden können[3].

Das österreichische Schulsystem befindet sich im Umbruch. Neben mehreren Schulversuchen und Lehrplanänderungen werden 2014 die ersten SchülerInnen nach einer neuen Reifeprüfungsordnung ihre Schulausbildung abschließen. Die Fourier - Theorie wird vorallem in technischen höheren Schulen unterrichtet und hierfür könnte die Software eingesetzt werden. Konkret kann das Programm zur Überprüfung einer errechneten Fourier - Reihe verwendet werden. Weiters können Rechteck-, Dreieck- und Sägezahnfunktion auf ihre Fourier - Koeffizienten untersucht werden. Anschaulich kann damit den SchülerInnen auch die akustische Veränderung bei Variation der unteren Fourier - Koeffizienten gezeigt werden. Es entstehen dadurch Dur- Akkorde. Von Interesse ist auch, den hörbaren Unterschied zwischen etwa Rechtecksignal und Dreiecksignal zu untersuchen. Das gleichzeitig grafische

[1] Mehr als 50% der reinen Arbeitszeit.

[2] 1000Hz entsprechen einer Wellenlänge von $\lambda = \frac{c}{f} = 34.3cm$, c bezeichnet hier Schallgeschwindigkeit

[3] Die dargestellten Wellenlängen sind abhängig von Bildschirmauflösung, verwendeten Browser $\cdots$.

und akustische Darstellen bei Änderung der Fourier - Koeffizienten visualisiert zudem den Zusammenhang zwischen Klangfarbe und Signalform.

Literaturverzeichnis

[1] http://de.selfhtml.org.
Zugriff: 05.06.2011.

[2] http://de.wikibooks.org/wiki/Java_Standard:_Grafische_Oberflächen_mit_AWT. Zugriff: 07.06.2011.

[3] http://de.wikipedia.org/wiki/Division_mit_Rest.
Zugriff: 07.06.2011.

[4] http://de.wikipedia.org/wiki/Division_mit_Rest.
Zugriff: 07.06.2011.

[5] http://de.wikipedia.org/wiki/Feld_(Datentyp).
Zugriff: 05.06.2011.

[6] http://de.wikipedia.org/wiki/For-Schleife.
Zugriff: 06.06.2011.

[7] http://de.wikipedia.org/wiki/Fourieranalyse.
Zugriff: 05.06.2011.

[8] http://de.wikipedia.org/wiki/Fourierreihe.
Zugriff: 04.06.2011.

[9] http://de.wikipedia.org/wiki/Gleichmäßige_Konvergenz.
Zugriff: 05.06.2011.

[10] http://de.wikipedia.org/wiki/Inkommensurabilität.
Zugriff: 04.06.2011.

[11] http://de.wikipedia.org/wiki/Instanziierung.
Zugriff: 05.06.2011.

[12] http://de.wikipedia.org/wiki/Integer_(Datentyp).
Zugriff: 05.06.2011.

[13] http://de.wikipedia.org/wiki/Interferenz_(Physik).
Zugriff: 02.06.2011.

[14] http://de.wikipedia.org/wiki/Internationales_Einheitensystem.
Zugriff: 07.06.2011.

[15] http://de.wikipedia.org/wiki/Java-Applet.
Zugriff: 05.06.2011.

[16] http://de.wikipedia.org/wiki/Java_(Programmiersprache).
Zugriff: 05.06.2011.

[17] http://de.wikipedia.org/wiki/Java_(Programmiersprache).
Zugriff: 12.04.2011.

[18] http://de.wikipedia.org/wiki/Oberton.
Zugriff: 02.06.2011.

[19] http://de.wikipedia.org/wiki/Objektorientierte_Programmierung.
Zugriff: 05.06.2011.

[20] http://de.wikipedia.org/wiki/Punktweise_Konvergenz.
Zugriff: 04.06.2011.

[21] http://de.wikipedia.org/wiki/Schall.
Zugriff: 02.06.2011.

[22] http://de.wikipedia.org/wiki/Schallintensität.
Zugriff: 02.06.2011.

[23] http://de.wikipedia.org/wiki/Schwingung.
Zugriff: 04.06.2011.

[24] http://de.wikipedia.org/wiki/Transversalwelle.
Zugriff: 02.06.2011.

[25] http://de.wikipedia.org/wiki/Wellenlänge.
Zugriff: 04.06.2011.

[26] http://de.wikipedia.org/wiki/Welle_(Physik).
Zugriff: 04.06.2011.

[27] http://download.oracle.com/javase/1.4.2/docs/tooldocs/windows/javac.html. Zugriff: 19.03.2011.

[28] http://www.cafeaulait.org/course/week5/28.html.
Zugriff: 06.06.2011.

[29] http://www.highscore.de/java/einfuehrung/variablen.html\#variablen_klassen_und_objekte. Zugriff: 05.06.2011.

[30] `http://www.htl.at/fileadmin/content/Lehrplan/HTL/ELEKTROTECHNIK_Anlage_1.1.3_302-97.pdf`. Zugriff: 05.06.2011.

[31] `http://www.phy.ntnu.edu.tw/ntnujava/index.php?topic=1060.0`. Zugriff: 07.06.2011.

[32] `http://www.studentshelp.de/p/referate/02/2285.htm`. Zugriff: 02.06.2011.

[33] D. Abts. *Grundkurs JAVA*. Friedr. Vieweg + Sohn Verlagsgesellschaft mbH, Braunschweig/ Wiesbaden, 1999.

[34] J. Decemeber. *JAVA- Einführung und Überblick*. Markt+Technik Buch- und Software-Verlag GmbH, Haar bei München, 1996.

[35] W. Kaballo. *Grundkurs Funktionalanalysis*. Spektrum Akademischer Verlag, Heidelberg, 2011.

[36] C. Schaefer L. Bergmann. *Lehrbuch der Experimentalphysik, Band 1*. Walter de Gruyter GmbH, Berlin, 2009.

[37] J. G. Roederer. *Physik und psychoakustische Grundlagen*. Springer- Verlag, New York Heidelberg Berlin, 1977.

[38] R.Singer S. Middendorf. *Java- Programmierhandbuch und Referenzen*. dpunkt- Velag für digitale Technologie, Hüthig GmbH, Heidelberg, 1999.

[39] M. R. Spiegel. *Fourier- Analysis*. McGraw- Hill Book Company GmbH, New York, 1990.

Abbildungsverzeichnis

Anhang

Source Code (FourierApp.java)

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import org.nfunk.jep.*;


public class FourierApp extends Applet implements ItemListener,
  ActionListener{

  /* Hier erfolgt zunächst die Deklaration der Variablen.
  Variablen, Konstanten und alle anderen Objekt, die hier erwaehnt
  werden sind in der kompletten Klasse bekannt und können von
  allen Funktionen verwendet werden.*/

  //Zunaechst werden die verschiedenen Objekte der Panels deklariert:
  private Checkbox sinus, cosinus, rechteck, dreieck, saegezahn,
    individuell;
  private CheckboxGroup radioGroup;
  private Button loeschen;

  private Button soundButton;
  private Button syntheseButton;

  private Button mehrKoeff;
  private Button wenigerKoeff;

  private TextField frequenz, periodenDauer, koeff, indiGleichanteil,
    indiKoeff_ak, indiKoeff_bk, indiZoom;

  //Der folgenden double Wert soll die Standardfrequenz bei
```

```
//Programmstart definieren.
private double freq = 220.;

//Entsprechend ist die Periodendauer mit 1/freq definiert.
private double perDauer = 1/freq;
private int grBreite, grHoehe;

//Die Images "bild" und "achs" inkludieren die Objekte für die
//grafische Ausgabe des Grafen und der Schieberegler.
Image bild, regl;
Graphics bildgraf, regler;

//"Dimension" ist ein Typ, welcher die Hoehe und die Breite enthaelt.
Dimension bereich, reglerBereich, grafBereich;

//Mit folgendem Objekt "gbc" vom Typ GridBagConstraint koennen
//Voreinstellungen fuer die Darstellung der einzelnen Panels
//gemacht werden.
private GridBagConstraints gbc;

//Die folgenden Variablen und Konstanten dienen hauptsaechlich
//der Positionierung der einzelnen Panels. Sie werden im
//Quelltext naeher erlaeutert.
private int reglerAbstandVonRand=40, x1=reglerAbstandVonRand,
  xRegler, x0, y2, ya, yb, yc, yd, yac, ddh = 10, h, h2;
private int d1, dOffs=5, maxx, ycd, yyc, yaaa, ybbb, yccc, yddd;
private int polX[], polY[], groesse=6, groesse2=12,yOffset;
private int koeffAnz=15;
private int Id=0,temp;
private double yVal[];

//Die beiden Variablen "SIN" und "COS" werden gesetzt, wenn sich
//der Mauszeiger bei den Reglern im Bereich der Sinus-
//Koeffizienten (bk) oder Cosinus- Koeffizienten (ak) befindet.
private final int SIN=0,COS=1;
//Die folgenden Variablen dienen zur Darstellung des Grafen.
private double amplitude;

//Die folgende boolean- Variable wird gesetzt, wenn die Maustaste
//gedrueckt wurde.
private boolean gedrueckt=false;

//tmp wird zur Berechnung der Fourierkoeffizienten herangezogen
```

```
private double tmp;

//Die Instanz f vom Objekt FourierHelp dient hauptsaechlich der
//Speicherung der Fourierkoeffizienten und der sound- Ausgabe
FourierHelp f;

/*Im Panel p0 werden p1,p2 und p3 eingebettet.
 * Panel p1 dient zur Auswahl der Signalform, bzw. zum loeschen
 *  der Fourierkoeffizienten.
 * Panel p2 dient zur Anzeige der Ausgabefrequenz, Periodendauer
 *  und des Gewichts der Fourierkoeffizienten.
 * Panel p3 wird erst sichtbar, wenn loeschen gedrueckt wurde
 *  und der User errechnet Fourier- Koeffizienten eintragen moechte.*/
private Panel p0,p1,p2,p3;

//Die Konstante PI.
private double PI = Math.PI;

//Farbbestimmung:
private Color hintergrund = new Color(0xCEF6CE),
              schiebereglerfarbe = new Color(0x58, 0x58, 0xFA),
              kurvenhintergrund = new Color(0xD8D8D8),
              reglerhintergrund = new Color(0xD8D8D8);

//Das JEP- Objekt zur Auswertung des mathematischen Ausdrucks.
//JEP ist ein Parser fuer regulaere Ausdruecke (Strings)
//20.3.2011: http://www.singsurf.org/djep/html/index.html
static JEP myParser = new JEP();



/** Die Funktion init dient der Initialisierung des Applets.
 * Diese Funktion wird unmittelbar nach Start des Applets
 * gestartet. Hiebei werden grundlegen Einstellungen der Panels
 * wir Groesse und Position definiert. Weiters werden saemtliche
 * grafischen Elemente und Interaktionsobjekte in den Panels
 * hinzugefuegt und fuer das Interagieren mit "addActionListener"
 * bereitgestellt.
 *
 * @param: -
 * @return: -
 * @author: David Stadler
 * @date: 22.3.2011
```

```
 */
public void init() {
  //21.3.:http://stackoverflow.com/questions/5193971/
  //applet-permission-denied
  //Damit das Applet Sicherheitsbestimmung nicht zum trotz

  //Das Hauptpanel soll als null- Layout verwendet werden,
  //d.h. zu- setzende Objekte muessen per Pixelkoordinaten in
  //diese Panel ("this") gesetzt werden.
  this.setLayout(null);

  //Bestimmung der Groesse des Panels
  this.setSize(600, 900);
  //this.setBackground(Color.LIGHT_GRAY);
  this.setBackground(hintergrund);

  //Folgendes Panel soll die Steuerung ermoeglichen und in der
  //unteren Fensterhaelfte sein. In diesem werden die anderen
  //Panels (p1,p2,p3) eingefuegt (siehe Beschreibung bei der
  //Deklaration).
  p0 = new Panel();

  /* 22.3.: http://www.dpunkt.de/java/Programmieren_mit_Java/
   * Oberflaechenprogrammierung/82.html
   * Das GridBagLayout basiert wie das GridLayout auf einem
   * Gitter. Allerdings ist der GridBagLayout-Manager erheblich
   * flexibler. Er bietet unter anderem folgende Moeglichkeiten:
     * Eine Komponente kann sich über einen Anzeigebereich von
     * mehreren Gitterzellen erstrecken. Spalten und Zeilen
     * koennen unterschiedlich breit bzw. hoch sein. Eine
     * Komponente kann die ihr zugewiesenen Gitterzellen voll
     * ausfüllen oder aber in ihrer normalen Groesse dargestellt
     * werden.Wenn sie in ihrer normalen Groeße dargestellt wird,
     * kann angegeben werden, wie sie innerhalb der zugewiesenen
     * Gitterzellen angeordnet wird. Falls die Groesse des
     * Containers vom Benutzer verändert wird, kann angegeben
     * werden, zu welchen Anteilen die Hoehen-/Breitenänderung
     * auf die einzelnen Gitterzeilen und -spalten verteilt wird.
     * So kann beispielsweise spezifiziert werden, dass sich
     * Aenderungen der Hoehe eines Fensters ausschließlich auf
     * die Hoehe einer bestimmten Gitterzeile auswirken und alle
     * anderen Gitterzeilen eine feste Hoehe haben.
    */
```

```
p0.setLayout(new GridBagLayout());

p1 = new Panel();
p1.setLayout(new FlowLayout());

p2 = new Panel();
p2.setLayout(new FlowLayout());

p3 = new Panel();
//Im Panel p3 wird GridLayout verwendet, da keine dynamische
//Anordnung wie bei GridBagLayout notwendig ist
//(GridLayout(Zeilen, Spalten, horizontaler Abstand,
//vertikaler Abstand zwischen den Zellen)).
p3.setLayout(new GridLayout(0,5,5,5));

//Hier werden die Checkboxen definiert. CheckboxGroup beinhaltet
//alle weiteren Checkboxen, es kann also immer nur genau eine
//Checkbox der CheckboxGroup angewaehlt sein.
radioGroup = new CheckboxGroup();
//"Sin" soll zu beginn ausgewaehlt sein -> state= true
sinus = new Checkbox("Sin", radioGroup, true);
cosinus = new Checkbox("Cos", radioGroup, false);
rechteck = new Checkbox("Rechteck", radioGroup, false);
dreieck = new Checkbox("Dreieck", radioGroup, false);
saegezahn = new Checkbox("Saegezahn", radioGroup, false);
individuell = new Checkbox("Individuell", radioGroup, false);

//Der Button "loeschen" dient zum loeschen des Grafen (alle
//Fouierkoeffizienten werden auf 0 gesetzt).
loeschen = new Button("loeschen");

//Der soundButton- Button ermoeglicht das hoeren des aktuell
//dargestellten Grafen.
soundButton = new Button("TON AN");

//Mittels des Synthese- Buttons, kann eine haendisch errechnet
//und eingegebene Fourierreihe optisch als Graf dargestellt
//werden.
syntheseButton = new Button("SYNTHESE");

//Die beiden Buttons "mehrKoeff" und "wenigerKoeff" dienen zur
//Erhoehung der zu- ermittelnden und darzustellenden
//Fouierkoeffizienten
```

```
mehrKoeff = new Button(">");
wenigerKoeff = new Button("<");

//Damit bei Druck auf den Checkboxen das Programm reagieren
//kann, ist es notwendig diese per "addItemListener" zu
//veroeffentlichen. Wenn gedrueckt wird, wird die Funktion
//"itemStateChanged" aufgerufen.
sinus.addItemListener(this);
cosinus.addItemListener(this);
rechteck.addItemListener(this);
dreieck.addItemListener(this);
saegezahn.addItemListener(this);
individuell.addItemListener(this);

//Damit bei Druck auf den Buttons das Programm reagieren kann,
//ist es notwendig diese per "addActionListener" zu
//veroeffentlichen. Wenn gedrueckt wird, wird die Funktion
//"actionPerformed" aufgerufen.
loeschen.addActionListener(this);
soundButton.addActionListener(this);
syntheseButton.addActionListener(this);
mehrKoeff.addActionListener(this);
wenigerKoeff.addActionListener(this);

//Hinzufuegen der Checkboxen zum Panel p1.
p1.add(sinus);
p1.add(cosinus);
p1.add(rechteck);
p1.add(dreieck);
p1.add(saegezahn);
p1.add(individuell);
p1.add(loeschen);

//Hinzufuegen der Textfelder und Labels zum Panel p2.
p2.add(new Label("f = "));
p2.add(frequenz = new TextField(String.valueOf(freq),3));
p2.add(new Label(",      T = "));
p2.add(periodenDauer
    = new TextField(String.valueOf(perDauer),3));
//PeriodenDauer haengt von Frequenz ab und soll ich nicht
//veraendert werden koennen
periodenDauer.setEditable(false);
p2.add(new Label(",      Koeff = "));
```

```
p2.add(koeff = new TextField("0",3));

//Ab hier beginnen die Voreinstellung fuer das Panel p3.
//Die Labels txt1 und txt2 dienen zum Darstellen des Texts
//("Input für T=2*pi:") und zwar fett und in Schriftgroesse 18
Label txt1 = new Label("Input", Label.RIGHT);
Label txt2 = new Label("für T=2*pi:", Label.LEFT);
txt1.setFont(new Font("Helvetica", Font.BOLD, 18));
txt2.setFont(new Font("Helvetica", Font.BOLD, 18));

//Hinzufuegen der vorher definiert Labels und Textfelder
//zum Panel p3
p3.add(txt1);
p3.add(txt2);

p3.add(new Label("Gleichanteil a0:", Label.RIGHT));
p3.add(indiGleichanteil = new TextField("4.*pi^2./12.",12));
p3.add(new Label(""));
p3.add(new Label(""));
p3.add(new Label(""));
p3.add(new Label("ak:", Label.RIGHT));
p3.add(indiKoeff_ak = new TextField("2.*(-1)^i/(i*i)",20));
p3.add(new Label(""));
p3.add(new Label(""));
p3.add(new Label(""));
p3.add(new Label("bk:", Label.RIGHT));
p3.add(indiKoeff_bk = new TextField("0.",20));
p3.add(new Label(""));
p3.add(new Label(""));
p3.add(syntheseButton);
p3.add(new Label("Dämpfungsfaktor:", Label.RIGHT));
p3.add(indiZoom = new TextField("1.9",3));

//PeriodenDauer haengt von Frequenz ab und soll ich nicht
//veraendert werden koennen
koeff.setEditable(false);

//Instanzieren des Hilfsobjekts f zur Speicherung der
//Fourierkoeffizienten und sound- Wiedergabe.
f = new FourierHelp(freq);

//Die Funktion berechneDim() dient zu ermitteln der
//verfuegbaren Groessen und Positionen fuer die
```

```
//Schiebereglersteuerung, Grafenbereich etc.
berechneDim();



/******Ausgaben*********
System.out.println("yOffset= " + yOffset);
System.out.println("d1= " + d1);
System.out.println("x0= " + x0);
System.out.println("x1= " + x1);
System.out.println("h= " + h);
System.out.println("y2= " + y2);
System.out.println("ya= " + ya);
System.out.println("yaaa= " + yaaa);
System.out.println("yb= " + yb);
System.out.println("ybbb= " + ybbb);
System.out.println("yac= " + yac);
System.out.println("yc= " + yc);
System.out.println("yccc= " + yccc);
System.out.println("yd= " + yd);
System.out.println("yddd= " + yddd);
System.out.println("ycc= " + ycd);
System.out.println("maxx= " + maxx);
System.out.println("reglerBereich.height= " +
  reglerBereich.height);
**************/


//Berechnen der Masze fuer die Grafik, 3 Arrays mit Groesse
//der Anzahl der Pixel des grafBereichs
polX = new int[grafBereich.width];
polY = new int[grafBereich.width];
//polY2 = new int[grafBereich.width];

//polX[]=0,2,4,6,8,10,.... Das sind die Pixel und bestimmen
//die Aufloesung des Grafen.
for(int i=0;i<grafBereich.width;i++)
  polX[i] = i*2;
//das letzte und erste Element der Arrays polY und polY2
//nimmt den Wert der halbe Hoehe des grafBereichs an
polY[grafBereich.width-1]=grafBereich.height/2;
polY[0]=grafBereich.height/2;
```

```
//System.out.println(java.util.Arrays.toString(polX));

//Das Panel p0 wird nach den beiden Grafiken positioniert
//und soll die Dimension Gesamtbreitex250 haben.
p0.setBounds(0, 600, bereich.width, 250);

//aus (9.3.2011) http://webcache.googleusercontent.com/
//search?q=cache:wes9OEDCTpoJ:www.java-forum.org/awt-swing-swt/
//100429-buttons-positionieren-gridbaglayout.html+
//gridlayout+platzieren+java+add&cd=1&hl=de&ct=clnk&gl=at&
//client=firefox-a&source=www.google.at
//gbc ist der Container, der alle Information bzgl. Groesse,
//Position etc. der aktuellen Zelle enthaelt.
gbc = new GridBagConstraints();

/*natural height, maximum width (21.3.http://download.oracle.com/
 * javase/tutorial/uiswing/layout/gridbag.html)
 * 22.3.: http://www.dpunkt.de/java/Programmieren_mit_Java/
 * Oberflaechenprogrammierung/82.html
 * Im Element GridBagConstraints.fill kann spezifiziert werden,
 * ob und wie die Komponente ihren Anzeigebereich ausfuellt:
 * GridBagConstraints.NONE: Die Komponente wird in ihrer
 * bevorzugten Größe dargestellt.
 * GridBagConstraints.HORIZONTAL: Die Komponente wird so ver-
 * breitert, dass sie die Breite des Anzeigebereichs voll ausfuellt.
 * GridBagConstraints.VERTICAL: Die Hoehe der Komponente wird so
 * veraendert, dass sie die Höhe des Anzeigebereichs voll ausfuellt.
 * GridBagConstraints.BOTH: Die Hoehe und die Breite der
 *  Komponente werden so veraendert, dass sie den Anzeigebereich
 * voll ausfuellt.
 */
    gbc.fill = GridBagConstraints.HORIZONTAL;


    //21.3.: der Abstand der Komponenten von den Seiten wird
    //jeweils auf 2 Pixel gesetzt
    //gbc.insets = new Insets(2, 2, 2, 2);
    gbc.insets = new Insets(0,0,0,0);

    //Einfuegen des Panels p1 in die zweite Zeile des Panels p0.
    //Die Funktion prepareGBC dient fuer Voreinstellungen der
    //Platzierung
```

```
        //prepareGBC(int x, int y, int width, int height),
        //x...Spalte, y...Zeile,
        //width...Anzahl der Zeilen, height...Anzahld der Spalten
        prepareGBC(0, 0, 6, 1);
    p0.add(p1, gbc);

    //gbc.weightx = 1;
        //gbc.weighty = 3.5;
    //Einfuegen des Panels p2 in die dritte Zeile:
        prepareGBC(0, 1, 6, 1);
    p0.add(p2, gbc);

    //22.3.: http://www.dpunkt.de/java/Programmieren_mit_Java/
    //Oberflaechenprogrammierung/82.html
    //Zur Aufteilung des Platzes gibt es die Funktionen weightx
    //und weighty. Sie ermoeglichen eine gewichtete Verteilung
    //des vorhandenen Platzes auf die einzelnen Gitterzellen.
    gbc.weightx = 1;
        gbc.weighty = 1;

        //Mein persoenliches Logo wird links unten platziert.
        //\u00a9 ist das Copyright- Symbol.
    Label txt3 = new Label("\u00a9 David Stadler 2011");
    txt3.setBounds(0, bereich.height-20, 200, 20);
    //txt3.setFont(new Font("Helvetica", Font.BOLD, 10));
    txt3.setFont(new Font("Serif", Font.ITALIC, 12));
    this.add(txt3);

    //p0, welches die Panels p1 und p2 (p3 wird erst spaeter
    //hinzugefuegt) enthaelte, wird in das Hauptpanel angefuegt.
    this.add(p0);

    //Dummy- Label, damit Formatierung konsistent bleibt und bei
    //Zuschaltung des Panels p3 keine Verschiebung passiert.
    prepareGBC(0, 2, 6, 1);
    p0.add(new Label(""), gbc);

    //Positionierung der Buttons "soundButton"(dargestellt als
    //"TON AN"), "mehrKoeff" (dargestellt als ">") und
    //"wenigerKoeff" (dargestellt als "<").
    soundButton.setBounds(grafBereich.width-60,
        grafBereich.height-30, 60, 30);
    mehrKoeff.setBounds(bereich.width-20, yddd*2+2*ya, 20, 10);
```

```
    wenigerKoeff.setBounds(bereich.width-40, yddd*2+2*ya, 20, 10);

    //Hinzufuegen obiger Buttons zum Hauptpanel.
    this.add(soundButton);
    this.add(mehrKoeff);
    this.add(wenigerKoeff);

    //Neuzeichnen aller Komponenten:
    this.repaint();

    //Setzen der Grundfrequenz im zuvor instanzierten Objekt f
    //vom Typ "FourierHelp"
    f.grundFrequenz(freq);

    //Nachdem an dieser Stelle alle notwendigen Variablen definiert
    //wurden und die Panels bereit sind, erfolgt der erst Start,
    //bei dem Sinus dargestellt wird (wurde zuvor als einzige
    //Checkbox angewaehlt).
    neustart();

    //Initialisierung von myParser, dass dieser Konstanten wie
    //pi etc. verwenden kann:
    myParser.addStandardFunctions();
      myParser.addStandardConstants();

  }

  /** Die Funktion berechneDim berechnet alle notwendigen
   * Koordinaten fuer Panels, Schiebereglergrafik etc.
   * (siehe Grafiken)
   *
   * @param: -
   * @return: -
   * @author: David Stadler
   * @date: 9.3.2011
   */
  private void berechneDim() {
    //Zunaechst wird die Gesamtgroesse des Hauptpanels ermitteln:
    bereich = this.getSize();
    //System.out.println("this.getSize()= " + this.getSize());

    //Berechnung der Groesse fuer den dargestellten Grafen und der
    //Reglersteuerung (soll drittel Panel ausfuellen):
```

```
grBreite = bereich.width;
grHoehe = bereich.height/3;

//Dimensionen von Regler und Grafenbereich ermitteln
grafBereich = new Dimension(grBreite, grHoehe);
//Der Reglerbereich soll aus zwei Bereichen bestehen
//(oberer und unterer Reglerbereich), deshalb wird hier die
//halbe Hoehe des komplett- verfuegbaren herangezogen.
reglerBereich = new Dimension(grBreite-reglerAbstandVonRand,
    (grHoehe-dOffs)/2);

//9.3.2011: http://webcache.googleusercontent.com/
//search?q=cache:eqEUQzxUxDIJ:straub.as/java/awt/
//Graphik.html+createimage+java&cd=12&hl=de&ct=clnk&gl=at&lr=
//lang_de&client=firefox-a&source=www.google.at
//Die Funktion createImage erstellt ein OffscreenImage.
regl = createImage(grafBereich.width, grafBereich.height);
bild = createImage(grafBereich.width, grafBereich.height);

//9.3.2011: http://webcache.googleusercontent.com/search?q=
//cache:eqEUQzxUxDIJ:straub.as/java/awt/Graphik.html+
//createimage+java&cd=12&hl=de&ct=clnk&gl=at&lr=lang_de&client=
//firefox-a&source=www.google.at
//Die Funktion getGraphics gestattet Zugriff auf das Image.
regler = regl.getGraphics();
bildgraf = bild.getGraphics();

//Die Variable yOffset dient zur Ermittlung des
//Positionsstarts fuer den Reglerbereich.
yOffset = bereich.height/3+10;

//d1 speichert den Abstand zwischen den Reglern.
d1 = reglerBereich.width/koeffAnz;

//x1 ist die Position des ersten Reglers, d1 der Abstand
//x0 ist zu Beginn die Position des ersten Reglers minus dem
//Abstand zum linken Regler. Es geht hierbei um den Bereich,
//in dem die Maus sein muss, um zum Bereich des Reglers zu
//gehoeren.
//x0 ist quasi die Toleranz, die zwischen den einzelnen Regler
//sein darf um zum Regler zu gehoeren.
x0 = x1 - d1;
```

```
//ddh ist der Abstand von oben/unten bzw. Mitte des kompletten
//Reglerpanels nach oben/unten und den Startbereich der Regler.
//h ist die Hoehe der "Spur" der Schiebregler.
h=reglerBereich.height-2*ddh-10;//10 ist die Schriftgroesse.

//h2 definiert die Mitte eines Reglers.
h2=h/2;

//y2 speichert den Positionsstart fuer die zweite Reglerhaelfte.
//dOffs ist der Abstand zwischen oberer und unterer Reglerhaelfte.
y2 = reglerBereich.height+dOffs;

//ddh der Abstand von oben/unten bzw. Mitte des Reglerpanels
//nach oben/unten und den Start-/Endbereich der Regler.
//Somit ist ya der Positionsstart der oberen Schiebereglerhaelfte.
ya = ddh;

//groesse speichert die Groesse des "Knopfes" fuer den
//Schieberegler. yaaa dient somit zum Ermitteln der maximalen
//Hoehe, die bei Klick des Mauszeigers auf den Knopf zulaessig ist.
yaaa = ya - groesse;

//yb ist die Unterkante des oberen Schiebreglers .
yb = ya + h;

//Analog zu yaaa, aber Knopf am Minimum.
ybbb = yb + groesse;

//yac dient zur Ermittlung der Mitte des Schiebereglerknopfes.
yac = (ya+yb)/2;

//yc ist die Oberkante des unteren Schiebereglers.
yc = y2 + ddh;

//Analog zur ersten Schiebereglerhaelfe.
yccc = yc - groesse;
yd = yc + h;
yddd = yd + groesse;
ycd = (yc+yd)/2;

//maxx speichert Information ueber den maximalen x-Bereich,
//der fuer die Erfassung der Position des Mauszeigers
//zulaessig ist.
```

```
    maxx = reglerBereich.width+d1/2;
  }

  /** Die Funktion neustart wird ausgefuehrt, wenn fundamentale
   * Aenderungen z.B. in der Frequenz oder in der Koeffizientenanzahl
   * vorgenommen wurden. Es wird uberprueft, welche Funktion
   * ausgewaehlt ist, es werden die Funktionswerte errechnet und die
   * Reglersteuerung neugestartet.
   *
   * @param: -
   * @return: -
   * @author: David Stadler
   * @date: 14.3.2011
   */
  private void neustart() {
    //Die Variable temp nimmt die Werte von "SIN" oder "COS" an.
    //Wenn sich der Mauszeiger im Reglerbereich der ak befindet,
    //wird temp auf "COS" gesetzt, wenn der Mauszeiger im Bereich
    //der bk ist, dann wird diese Variable auf "SIN" gesetzt,
    //ansonsten auf -1.
    temp=-1;

    //Ermittlung, welche Kurve ausgewaehlt ist:
    letsPlay();
    //Berechnen der y-Werte der Kurve, welche Kurve dies ist,
    //wird durch letsPlay definiert
    yVal=f.yVal();

    //Neuzeichnen der Reglersteuerung
    ursprung();
    this.repaint();
  }

  /** Die Funktion letsPlay wird aufgerufen, um zu ermitteln, welche
   * Checkbox akutell ausgewaehlt ist. Entsprechend werden die
   * Fourierkoeffizienten im Objekt f
   * verschieden gesetzt.
   *
   * @param: -
   * @return: -
   * @author: David Stadler
   * @date: 14.3.2011
   */
```

```
private void letsPlay(){

  //blosz sinus:
  if(radioGroup.getSelectedCheckbox().getLabel() == "Sin") {

    //Alle Koeffizienten auf 0 setzen.
    for(int i=0;i<=koeffAnz;i++){
      f.set_bk(i,0.);
      f.set_ak(i,0.);
    }

    //Grundwelle (von sin) auf 1 setzen
    f.set_bk(1,1.0);
  }

  //blosz cosinus:
  if(radioGroup.getSelectedCheckbox().getLabel() == "Cos") {

    //Alle Koeffizienten auf 0 setzen
    for(int i=0;i<=koeffAnz;i++){
      f.set_bk(i,0.);
      f.set_ak(i,0.);
    }

    //Grundwelle (von cos) auf 1 setzen
    f.set_ak(1,1.0);
  }

  //angelehnt an (14.3.2011): http://www.ilp.physik.uni-essen.de/
  //vonderLinde/lehrveranstaltungen/PHYSIK_II/Material/
  //fourier_reihen.pdf
  if(radioGroup.getSelectedCheckbox().getLabel() == "Rechteck") {
    for(int i=0;i<=koeffAnz;i++){
      if(i%2==1) //für ungerade Koeffizienten
        f.set_bk(i,4.0*0.77/PI*1/i);
      else
        f.set_bk(i,0.);

      //Alle anderen Koeffizienten auf 0 setzen.
      f.set_ak(i,0.);
    }
  }
```

```
    //angelehnt an (14.3.2011): http://www.ilp.physik.uni-essen.de/
    //vonderLinde/lehrveranstaltungen/PHYSIK_II/Material/
    //fourier_reihen.pdf
    if(radioGroup.getSelectedCheckbox().getLabel() == "Dreieck") {
      for(int i=0;i<=koeffAnz;i++){
        if(i%2==1) //für ungerade Koeffizienten
          f.set_ak(i,4.0*0.77/PI*1/(i*i));
        else
          f.set_ak(i,0.);

        //Alle anderen Koeffizienten auf 0 setzen.
        f.set_bk(i,0.);
      }
    }

    //angelehnt an (14.3.2011): http://www.ilp.physik.uni-essen.de/
    //vonderLinde/lehrveranstaltungen/PHYSIK_II/Material/
    //fourier_reihen.pdf
    if(radioGroup.getSelectedCheckbox().getLabel() == "Saegezahn") {
      f.set_bk(0,0.);
      f.set_ak(0,0);
      for(int i=1;i<=koeffAnz;i++){
          f.set_bk(i,-2.0*1.54/PI*1/(i));

          //ak auf 0 setzen
          f.set_ak(i,0.);
      }
    }
  }


  /** Mittels der Funktion ursprung wird die Reglersteuerung
   * neugezeichnet. Dies wird z.B. benoetigt, wenn ein anderes als
   * aktuell ausgewaehltes Signal ausgewaehlt wird und dadurch die
   * Schieberegler angepasst werden muessen.
   *
   * @param: -
   * @return: -
   * @author: David Stadler
   * @date: 14.3.2011
   */
  private void ursprung() {
```

```
    //Zu Beginn wird die Farbe festgelegt
    regler.setColor(reglerhintergrund);
    //Dann wird der Reglerbereich mit dieser Farbe gefuellt (es
    //gelten die Dimensionen des Grafbereichs (drittel der Gesamtmasze))
    regler.fillRect(0, 0, grafBereich.width, grafBereich.height);

    //Setzen der Farbe auf schwarz.
    regler.setColor(Color.black);

    //Initialisierung des Reglerbereichs mit akutellen
    //Fourierkoeffizienten (siehe Funktion "regler()")
    for(int i=x1-d1/2,j=0;j<=koeffAnz;i+=d1,j++){
      regler(j,SIN,schiebereglerfarbe);
      regler(j,COS,schiebereglerfarbe);

      //Die Beschriftung der Regler (Nummerierung ueber bzw. unter
      //dem entsprechenden Reglers)
      regler.setColor(Color.black);

      //Alle werden beschriftet, bis auf den ersten (a0, b0)
      if(j!=0)
        regler.drawString(""+j,i-4,yb+25);
    }

    //Zuletzt werden noch am linken Rand "b" und "a" notiert um
    //zwischen den aks und bks zu unterscheiden.
    regler.drawString("b",8,yb+14);
    regler.drawString("a",8,yc-6);
  }

  /** Die Funktion regler dient zum Zeichnen der einzelne Regler der
   * Schiebereglersteuerung.
   *
   * @param: int l (aktueller Fouierkoeffizient), int stat (Sinus-
   *  oder Cosinus- Reglerbereich), Color c (mit welcher Farbe)
   * @return: -
   * @author: David Stadler
   * @date: 14.3.2011
   */
  private void regler(int l,int stat,Color c){

    //Farbe auf schwarz setzen.
    regler.setColor(Color.black);
```

```
//Positionsermittlung (x-Koordinate) des einzustellenden Reglers.
xRegler=x1-d1/2+l*d1;

//System.out.println("l= " + l);
//System.out.println("(int)(f.get_ak(l)= " + (f.get_ak(l)));
//System.out.println("(int)(f.get_bk(l)= " + (f.get_bk(l)));


//Wenn im Sinus- Bereich (bk) des Reglers
if(stat==SIN) {

  //Berechnung der oberen Kante (y- Koordinate) des neu zu-
  //zeichnenden Reglerknopfes:
  yyc=yac-groesse-(int)(f.get_bk(l)*h2);

  //Der Knopf des Schiebereglers soll nur gezeichnet werden,
  //wenn die neue Reglerposition im Range bleibt
  //(ycc zwischen 6 und 120):
  if(yyc<6)
    yyc=6;

  if(yyc>120)
    yyc=120;

  //Nun werden die "Spuren" des Schieberegler gezeichnet:
  //draw3DRect(x,y,width,height,erhoehter wert)
  regler.draw3DRect(xRegler-1,ya,2,h,false);

  //Die Farbe wird nun auf blau gesetzt.
  regler.setColor(Color.blue);

  //Um die Mitte des Reglers zu erkennen, wir in der Mitte
  //ein 4 Pixel breiter Strich gezeichnet.
  regler.drawLine(xRegler-2,yac,xRegler+2,yac);
}else if(stat==COS){//Wenn im Kosinus- Bereich (bk) des Reglers

  //Berechnung der oberen Kante (y- Koordinate) des neu zu-
  //zeichnenden Reglerknopfes:
  yyc=ycd-groesse-(int)(f.get_ak(l)*h2);

  //Hier nur zeichnen, wenn die neue Reglerposition im Range
  //bleibt (yyc zwischen 158 und 272):
```

```
      if(yyc<158)
        yyc=158;

      if(yyc>272)
        yyc=272;

      //Nun werden die "Spuren" des Schieberegler gezeichnet:
      //draw3DRect(x,y,width,height,erhoehter wert)
      regler.draw3DRect(xRegler-1,yc,2,h,false);

      //Die Farbe wird nun auf blau gesetzt.
      regler.setColor(Color.blue);

      //Um die Mitte des Reglers zu erkennen, wir in der Mitte
      //ein 4 Pixel breiter Strich gezeichnet.
      regler.drawLine(xRegler-2,ycd,xRegler+2,ycd);
      //}
    }

    //Die Farbe wird nun auf die uebergebene Farbe gesetzt.
    regler.setColor(c);

    //Da nun alle notwendigen Koordinaten bekannt sind, wird zuletzt
    //die Flaeche fuer die Schieberegler gezeichnet.
    //groesse2 ist die Gesamtbreite des Schiebereglerknopfes.
    //fillOval(x,y,width,height)
    regler.fillRect(xRegler-groesse,yyc,groesse2,groesse2);
  }

  /** Die Funktion prepareGBC dient zur Voreinstellung der danachfolgenden
   *  Positionierung der Panels bei der Initialisierung.
   *
   * @Quellen: 9.3.2011 http://webcache.googleusercontent.com/search?
   *  q=cache:wes9OEDCTpoJ:www.java-forum.org/awt-swing-swt/
   *  100429-buttons-positionieren-gridbaglayout.html+gridlayout+
   *  platzieren+java+add&cd=1&hl=de&ct=clnk&gl=at&client=firefox-
   *  a&source=www.google.at
   *  http://www.dpunkt.de/java/Programmieren_mit_Java/
   *  Oberflaechenprogrammierung/82.html
   *
   * @param: int x, int y, int width, int height
   * @return: -
   * @author: David Stadler
```

```
 * @date: 9.3.2011
 */
private void prepareGBC(int x, int y, int width, int height) {
   //Zur Positionierung gibt es die Funktionen gridx und gridy.
  //Sie legen die obere linke Gitterzelle des Anzeigebereichs fest.
        gbc.gridx = x;
        gbc.gridy = y;

        //Das Datenelement gridwidth definiert die Anzahl an
        //Gitterzellen, ueber die sich der Anzeigebereich horizontal
        //erstreckt.
        //Analog definiert die Anzahl an Gitterzellen, über die sich
        //der Anzeigebereich vertikal erstreckt, die Funktion gridheight.
        gbc.gridwidth = width;
        gbc.gridheight = height;
    }

/** Die Methode paint wird immer dann aufgerufen, wenn neugezeichnet w
 * erden muss. Z.B. wenn das Fenster in dem sie sich befindet verdeckt
 * war und wieder im Vordergrund angezeigt wird.
 *
 * @Quelle: http://www.ai.wu.ac.at/manuals/hahsler/
 *  java/javakurs/part10.html
 *
 * @param: Graphics g(Graphics ist der Bildschirmbereich, auf den
 * //gezeichnet werden kann. Die Klasse enthält Methoden zum
 * Schreiben von Text und zeichnen von geometrischen Figuren.)
 * @return: -
 * @author: David Stadler
 * @date: 9.3.2011
 */
public void paint(Graphics g){
  ursprung();
  update(g);
}


/** Die Methode update wird aufgerufen, wenn ein Befehl zum neu-
 * zeichnen aufgerufen wird, z.B. durch repaint.
 *
 * @param: Graphics g(Graphics ist der Bildschirmbereich, auf den g
 * ezeichnet werden kann. Die Klasse enthält Methoden zum Schreiben
 * von Text und zeichnen von geometrischen Figuren.)
```

```
     * @return: -
     * @author: David Stadler
     * @date: 13.3.2011
     */
    public void update(Graphics g){

      //Wenn (Graphics)bildgraf existiert, soll die Funktion neu durch
      //die Funktion zeichneGraf definiert werden.
      if(bildgraf!=null)
        zeichneGraf();

      //Die Funktion drawImage zeichnet das angegebene Image in seiner
      //urspruenglichen physikalischen Groesse an der angegebenen Position.
      g.drawImage(bild, 0, 0, this);
      g.drawImage(regl, 0, yOffset, this);

      //repaint();
      validate();
    }

    /** Die Funktion zeichneGraf definiert den Grafen neu. Dies inkludiert
     * die Farbwahl, Berechnung der Fourierkoeffizienten und zeichnen
     * dieser.
     *
     * @param: -
     * @return: -
     * @author: David Stadler
     * @date: 14.3.2011
     */
    private void zeichneGraf(){

      //Zunaechst wir die Hintergrundfarbe definiert.
      bildgraf.setColor(kurvenhintergrund);

      //Nun wird ein grossen Rechteck als Hintergrund gezeichnet.
      bildgraf.fillRect(0, 0, grafBereich.width, grafBereich.height);

      //Jetzt wird die Farbe fuer die Abszisse definiert...
      bildgraf.setColor(Color.black);

      //... und gezeichnet.
      bildgraf.drawLine(0,grafBereich.height/2,grafBereich.width,
          grafBereich.height/2);
```

```
    //Nun werden die Fourierkoeffizienten berechnet.
    f.berechne();

    //Die Amplitude ist von b0 abhaengig und wird hier so angepasst,
    //dass die auswaehlbaren Funktionen gut dargestellt werden.
    amplitude=50.*(f.get_bk(0)+1.0);

    //Fuer jeden Bildpunkt in der x-Koordinate (die Variable i nimmt
    //diese Werte an) wird der Funktionswert des Grafen ermittelt.
    //grafBereich.height/2 ist die Abszisse. yVal wird durch die
    //HilfsKlasse FourierHelp ermittelt und enthaelt die unskalierten
    //Funktionswerte.
    for(int i=1;i<grafBereich.width-1;i++){
      polY[i]=grafBereich.height/2-
      (int)(amplitude*yVal[i % f.periode]);
      //i%f.periode: nach Periode f.periode wird aequivalenter
      //Wert wieder ausgegeben.

    //System.out.println("polY= " + java.util.Arrays.toString(polY));
    }

    //Farbe wird auf schwarz gesetzt.
    bildgraf.setColor(Color.black);

    //Hier erfolgt die Zeichnung des Grafen durch einen Polygonzug.
    //polX beinhaltet die x-Werte, polY die y-Werte.
    //drawPolygon(int[] xPoints,int[] yPoints,int nPoints).
    bildgraf.drawPolygon(polX,polY,grafBereich.width);

    //Zum Darstellen der Periodendauer der Funktion wird an jeder
    //entsprechenden Position ein 10 Pixel breiter Strich gezeichnet.
    for(int i=f.periode;i<grafBereich.width;i+=f.periode)
      bildgraf.drawLine(i,grafBereich.height/2-5,i,
          grafBereich.height/2+5);
  }


  /** Die Funktion mouseMove wird aufgerufen, wenn der Mauszeiger
   * bewegt wird (keine Maustaste wurde gedrueckt).
   *
   * @Quelle: 10.3.2011: http://www.debacher.de/wiki/Java
   *
```

```
 * @param: Event e (hier kann ueberprueft werden ob beispielsweise
 * shift oder strg gedrueckt ist), int x (x-Koordinate des
 * Mausezeigers), int y (y-Koordinate des Mausezeigers)
 * @return: boolean (wird nicht benoetigt)
 * @author: David Stadler
 * @date: 10.3.2011
 */
public boolean mouseMove(Event e, int x, int y){

  //Von Interesse ist der Mauszeiger nur, wenn er im Range des
  //Schiebereglersteuerung ist, also im Bereich ab yOffset ist.
  //(y=y-yOffset)
  y-=yOffset;

  //Der Mauszeiger ist nun genau dann im Schiebereglerbereich,
  //wenn y>0 ist. Weiters soll nur der Fall behandelt werden, wenn
  //die Maustaste nicht gedrueckt ist.
  if(gedrueckt || y<0 )
    return false;

  //Tritt obige Bedingung ein, so interessiert nun die x- Position
  //des Mauszeigers. Hierbei ist nur wichtig, die
  //x-Koordinate<xmaxx ist
  if(x<maxx){

    //x wird nun auf den minimal- zulaessigen x- Wert des
    //jeweiligen Reglers gesetzt (x=x-x0)
    x-=x0;

    //System.out.println("x= " + x);

    //Nun wird abgefragt, ob sich die Maus aktuell im Sinus-
    //Bereich befindet.
    if(y>yaaa && y<ybbb){
      //Wenn ja, wird temp auf SIN gesetzt
      temp=SIN;

      //Nur wird ermittelt, um welchen Regler es sich handelt.
      //d1 speichert den Abstand zwischen den Reglern.
      //Entsprechend ist die Division der Position durch den
      //Abstand der gesucht Regler (Id*d1=x)
      Id=x/d1;
```

```
        //System.out.println("Id=" + Id);

        //Die Position des Mauszeigers ist nun ermittelt. Fuer
        //diese Position ist ein errechneter Koeffizient im
        //Objekt f gespeichert. Dieser Koeffiziente wird in
        //das Label koeff gespeichert.
        koeff.setText(String.valueOf(f.get_bk(Id)));

      }else if(y>yccc && y<yddd){
        //falls der Mauszeiger andererseits im Kosinus- Bereich
        //ist, passiert analoges.
        temp=COS;
        Id=x/d1;
        koeff.setText(String.valueOf(f.get_ak(Id)));
      }else temp=-1; //Falls der Mauszeiger weder im Sinus-,
      //noch im Kosinus-Bereich ist, so wird temp auf -1 gesetzt.
    }
    return true;
  }

  /** Die Funktion mouseDown wird aufgerufen, wenn die Mousetaste
   * gedrueckt wurde.
   *
   * @param: Event e (hier kann ueberprueft werden ob beispielsweise
   *  shift oder strg gedrueckt ist), int x (x-Koordinate des
   *  Mausezeigers), int y (y-Koordinate des Mausezeigers)
   * @return: boolean (wird nicht benoetigt)
   * @author: David Stadler
   * @date: 10.3.2011
   */
  public boolean mouseDown(Event e, int x, int y){

    //Von Interesse ist der Mauszeiger nur, wenn er im Range des
    //Schiebereglersteuerung ist, also im Bereich ab yOffset ist.
    //(y=y-yOffset)
    if((y-=yOffset)<0)
      return false;

    //Weiters ist der Mauszeiger nur von Interesse, wenn dieser
    //aktuell im ak-,bzw. bk- Bereich ist.
    if(temp==COS || temp==SIN) {
      //Wenn dies der Fall ist, so wird die Variable gedrueckt
      //auf true gesetzt...
```

```
      gedrueckt=true;

      //...und die Checkbox individuell gesetzt.
      individuell.setState(true);
    }

    return true;
  }

  /** Die Funktion mouseDrag wird aufgerufen, wenn der Mauszeiger
   * mit gedrueckter Taste bewegt wurde.
   *
   * @param: Event e (hier kann ueberprueft werden ob beispielsweise
   *  shift oder strg gedrueckt ist), int x (x-Koordinate des
   *  Mausezeigers), int y (y-Koordinate des Mausezeigers)
   * @return: boolean (wird nicht benoetigt)
   * @author: David Stadler
   * @date: 10.3.2011
   */
  public boolean mouseDrag(Event e, int x, int y){
    //Von Interesse ist der Mauszeiger nur, wenn er im Range des
    //Schiebereglersteuerung ist, also im Bereich ab yOffset ist.
    //(y=y-yOffset) und die Variable gedrueckt gesetzt ist, also
    //die Maustauste gedrueckt ist.
    if((y-=yOffset)<0 || !gedrueckt)
      return false;

    //Falls obige Bedingungen zutreffen, dann wird je nach aktueller
    //Position des Mauszeiger (definiert durch SIN und COS) der
    //Schiebereglerknopf bewegt und der entsprechende Koeffizient
    //gesetzt.
    switch(temp){
    case SIN:
      //ya ist die obere Kante des Schieberegler, yb die untere
      //Kante.
      if(y>ya && y<yb){
        //Zunaechst wird der zu aendernde Schiebereglerknopf
        //mit der Hintergrundfarbe des Schiebereglerbereichs
        //belegt. Somit ist zunaechst kein Schiebereglerknopf
        //kurzzeitig sichtbar.
        regler(Id,SIN,reglerhintergrund);

        //Hier werden nun entsprechend die Fourierkoeffizienten
```

```
        //(in Abhaengigkeit der Position des Schiebereglers)
        //neu gesetzt. Dabei wird ein kleiner Trick angewandt:
        //da yac, y und h2 Integer sind und der erlaubte Range
        //des Schiebereglers zwischen -1 und 1 liegt, ist die
        //releveante Dividion (yac-y)/h2 immer 0. Damit dies
        //unterbunden wird, wird der errechnet Werte mit 1000
        //multiplizert und danach wieder
        //dividiert.
        f.set_bk(Id,tmp=(int)(1000.*(yac-y)/h2)/1000.);

        //Der errechnete neue Koeffizient tmp wird als Text
        //im Textfield koeff dargestellt.
        koeff.setText(String.valueOf(tmp));

        //An der akutellen Position wird nun wieder der Knopf
        //fuer den Schieberegler gezeichnet.
        regler(Id,SIN,schiebereglerfarbe);

        //Neuzeichnen.
        repaint();
      }
      break;
    case COS:
      //yc ist die obere Kante des Schieberegler in der zweiten
      //Schiebereglersteuerungshaelfte, yd die untere Kante.
      if(y>yc && y<yd){
        //Analoges vorgehen zum Sinus-Bereich.
        regler(Id,COS,reglerhintergrund);
        f.set_ak(Id,tmp=(int)(1000.*(ycd-y)/h2)/1000.);
        koeff.setText(String.valueOf(tmp));

        regler(Id,COS,schiebereglerfarbe);
        repaint();
      }
      break;
    default:
      break;
    }
    return true;
  }

  /** Die Funktion mouseUp wird aufgerufen, wenn die Mousetaste
   * losgelassen wurde.
```

```
   *
   * @param: Event e (hier kann ueberprueft werden ob beispielsweise
   *  shift oder strg gedrueckt ist), int x (x-Koordinate des
   *  Mausezeigers), int y (y-Koordinate des Mausezeigers)
   * @return: boolean (wird nicht benoetigt)
   * @author: David Stadler
   * @date: 10.3.2011
   */
  public boolean mouseUp(Event e, int x, int y){
    //Von Interesse ist der Mauszeiger nur, wenn er im Range des
    //Schiebereglersteuerung ist, also im Bereich ab yOffset ist.
    //(y=y-yOffset)
    if((y-=yOffset)<0)
      return false;

    //Wenn die Maustaste losgelassen wird, wird die Variable gedrueckt
    //auf false gesetzt.
    gedrueckt=false;

    //Neuzeichnen.
    repaint();

    //temp auf -1 setzen
    temp=-1;
    return true;
  }

  /** Die Funktion action wird aufgerufen, wenn die enter- Taste
   * gedrueckt wurde. In diesem Fall wird die Frequenz geaendert
   * auf den Wert, der im Textfield eingegeben wurde.
   *
   * @param: Event ev (hier kann ueberprueft werden, ob andere Tasten
   *  zusaetzlich gedrueckt wurden), Object arg
   * @return: boolean (wird nicht benoetigt)
   * @author: David Stadler
   * @date: 10.3.2011
   */
  public boolean action(Event ev, Object arg) {
    //Zunaechst wird der Wert aus dem Textfeld ausgelesen.
    freq=Double.valueOf(frequenz.getText()).doubleValue();

    //Dann wird dieser Wert an das Objekt f uebergeben.
    f.grundFrequenz(freq);
```

```
    //Es wird die neue Periodendauer ermittelt...
    perDauer = 1/freq;

    //... und diese Periodendauer in das zugehoerige Textfeld
    //eingefuegt.
    periodenDauer.setText(String.valueOf(perDauer));

    //Da signifikante Aenderungen beim dargestellten Graf und der
    //Akustik gemacht wurden, ist ein Neustart uber die Funktion
    //neustart() notwendig.
    neustart();

    return true;
  }

  /** Die Funktion itemStateChanged wird aufgerufen, wenn eine
   * unterschiedliche als aktuelle ausgewaehlte Checkbox ausgewaehlt wird.
   *
   * @param:
   * @return:
   * @author: David Stadler
   * @date: 10.3.2011
   */
  public void itemStateChanged(ItemEvent arg0) {

    //Wenn eine Checkbox ausgewaehlt wird, die nicht der Checkbox
    //Individuell entspricht, so soll das Zusatzpanel p3 entfernt
    //werden.
    if(arg0.getItem() != "Individuell") {
      p0.remove(p3);

      repaint();
    }

    //Neustart, da signifikante Aenderungen vorgenommen sind (andere
    //Funktion, Akustik).
    neustart();

  }

  /** Die Funktion actionPerformed wird aufgerufen, wenn ein Button
   * gedrueckt wurde.
```

```
   *
   * @param: ActionEvent arg0 (fuer die Zuordnung des gedrueckten
   *  Buttons)
   * @return: -
   * @author: David Stadler
   * @date: 10.3.2011
   */
  public void actionPerformed(ActionEvent arg0) {

    //Der Fall, dass der loeschen- Button gedrueckt wurde.
    if(arg0.getActionCommand() == "loeschen"){
      //Die Variable temp wird auf -1 gesetzt.
      temp=-1;

      //Alle Fourierkoeffizienten werden auf 0 gesetzt.
      for(int i=0;i<=koeffAnz;i++){
        f.set_bk(i,0.);
        f.set_ak(i,0.);

      }
      //Berechnen der y-Werte der Kurve (alle 0, da durch
      //set_?k auf 0 gesetzt)
      yVal=f.yVal();

      //Neuzeichnen der Reglersteuerung
      ursprung();

      //Die Checkbox "individuell" soll ausgewaehlt sein
      individuell.setState(true);

      //Vorbereitende Einstellungen fuer das Positionieren des
      //Panels p3
          prepareGBC(0, 4, 6, 1);

          //Panel p3 wird in das Panel p0 eingefuegt.
      p0.add(p3, gbc);

      //Neuzeichnen.
      repaint();
    }

    //Der Fall, dass der Synthese- Button gedrueckt wurde.
    if(arg0.getActionCommand() == "SYNTHESE"){
```

```
//Hier wird ein try-catch Block herangezogen um fehlerhafte
//Eingaben des Users abzufangen.
try {
  //An dieser Stelle kommt JEP zum Einsatz. JEP ist externe
  //SW, welche regulaere Ausdruecke (Strings) auswerten
  //kann.
  //In dieser Arbeit wird Version 2.4.1 verwendet, da
  //diese frei zugaenglich ist.

  //Zunaechst wird der zu verarbeitende Gleichanteils-
  //Ausdruck an das JEP- Objekt myParser uebergeben.
  myParser.parseExpression(config(
      indiGleichanteil.getText()));

  //Nun wird das davor uebergebene Objekt ausgewertet.
  Object result = myParser.getValueAsObject();

  //Hier findet nun die eigentliche Funktionssynthese
  //statt.
  //a0 wird auf den Wert des Gleichanteils gesetzt.
  f.set_ak(0,((Double) result)/
      Double.valueOf(indiZoom.getText()));

  //Hier erfolgt nun die Auswertung der einzelnen
  //Fourierkoeffizienten.
  for(int i=1;i<=koeffAnz;i++){

    //Dem JEP- Ausdruck myParser wird der Wert von i
    //bekanntgegeben:
    myParser.addVariable("i", i);

    //Auswerten von ak fuer den Koeffizienten i.
    myParser.parseExpression(
        config(indiKoeff_ak.getText()));

    //Schliesslich wird er davor errechnet Koeffizient
    //gesetzt.
    f.set_ak(i,((Double) myParser.getValueAsObject())/
        Double.valueOf(indiZoom.getText()));

    //Auswerten von bk fuer den Koeffizienten i.
    myParser.parseExpression(config(
```

```
                indiKoeff_bk.getText()));

            //Nun wird der errechnet Koeffizient an das Objekt
            //f uebergeben.
            f.set_bk(i,((Double) myParser.getValueAsObject())/
                Double.valueOf(indiZoom.getText()));

        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    //Alle Werte sind nun im Objekt f bekannt. Es werden nun
    //Funktionswerte ausgelesen:
    yVal=f.yVal();

    //Neuzeichnen der Reglersteuerung
    ursprung();

    //Im Fall, dass neben der gewollten Synthese der Fourierreihe
    //auch die Frequenz geaendert wurde, wird diese hier neue
    //definiert.
    freq=Double.valueOf(frequenz.getText()).doubleValue();

    //Uebergabe der eingegebenen Frequenz an das Objekt f.
    f.grundFrequenz(freq);

    //Berechnung der Periodendauer
    perDauer = 1/freq;

    //Text des Textfeldes periodenDauer setzen.
    periodenDauer.setText(String.valueOf(perDauer));

    //Neustart.
    neustart();

    //Neuzeichnen.
    repaint();

}

//Der Fall, dass der "TON AN"- Button gedrueckt wurde.
if(arg0.getActionCommand() == "TON AN"){
```

```
    //In diesem Fall beginnt die Tonausgabe...
    f.start();

    //...und der Button wird in "TON AUS" umbenannt.
    soundButton.setLabel("TON AUS");

    //Neuzeichnen
    repaint();
  }

  //Der Fall, dass der "TON AUS"- Button gedrueckt wurde.
  if(arg0.getActionCommand() == "TON AUS"){

    //In diesem Fall endet die Tonausgabe...
    f.stop();

    //...und der Button wird in "TON AN" umbenannt.
    soundButton.setLabel("TON AN");

    //Neuzeichnen
    repaint();
  }

  //Der Fall, dass der ">"- Button gedrueckt wurde. Es werden
  //mehr Fourierkoeffizienten angefordert.
  if(arg0.getActionCommand() == ">"){

    //Die Variable koeffAnz (fuer die Anzahl der
    //Fourierkoeffizienten) wird erhoeht.
    koeffAnz++;

    //Da die Schiebereglersteuerung nun einen Regler mehr
    //inkludieren soll, sind die Dimension neu zu berechnen.
    berechneDim();

    //Neustart des Systems.
    neustart();

    //Neuzeichnen.
    this.repaint();
  }
```

```
    //Der Fall, dass der "<"- Button gedrueckt wurde. Es werden
    //weniger Fourierkoeffizienten angefordert.
    if(arg0.getActionCommand() == "<"){

      //Die Anzahl soll nur verringert werden, wenn die Anzahl >1
      //ist, da sonst ein Fehler auftritt.
      if(koeffAnz>1)
        koeffAnz--;

      //Wenn weniger Fourier-Koeffizienten verwendet werden sollen,
      //muss der letzte Koeffizient 0 gesetzt werden.
      f.set_ak(koeffAnz+1,0.);
      f.set_bk(koeffAnz+1,0.);


      //Da die Schiebereglersteuerung nun einen Regler mehr
      //inkludieren soll, sind die Dimension neu zu berechnen.
      berechneDim();

      //Neustart des Systems.
      neustart();

      //Neuzeichnen.
      this.repaint();
    }
  }



     /** Die Funktion config dient zum Parsen des uebergebenen Textes.
      * Es generiert aus dem String "i^j" den String  "pow(i,j)", wie
      * es in java verarbeitet werden kann.
     *
     * @param: String text (der zu parsende- Text)
     * @return: String (der verarbeitete Text)
     * @author: David Stadler
     * @date: 18.3.2011
     */
     private String config(String text) {

        //Zunaechst wird ueberprueft, ober in dem uebergebenen String
       //ueberhaupt das Zeichen "^" vorkommt. Wenn "+" oder "-"
       //im String vorkommt, dann wird dieser nicht veraendert,
```

```
//da in diesem Fall die externe Software JEP zum Einsatz
//kommt.
  if(text.contains("^") && text.contains("+")==false &&
      text.contains("-")==false) {

  String ersetze=text;
//Wenn ja, dann wird nun ermittelt, an welcher Position
    //sich das Zeichen "^" befindet.
int posH = text.indexOf("^");

//Ab dieser Position wird der String in zwei Hilfsstrings
//aufgeteilt (vor und nach "^")
String s1 = text.substring(0, posH); //0 ist die Position
//des ersten Zeichens.
String s2 = text.substring(posH+1, text.length());
//text.length ist die Position des letzten Zeichens.

//Die Hilfsvariable help1 wird -1 gesetzt.
int help1=-1;

//Im ersten String wird nun die Position des ersten
//"*" oder "/" ermittelt:
if(s1.lastIndexOf("*") >= 0)
    help1 = s1.lastIndexOf("*");

if(s1.lastIndexOf("/") >= 0 && s1.indexOf("/") < help1
   && help1!=-1)
    help1 = s1.lastIndexOf("*");


//Basis der zu-ausgefuehrenden Potenzierung:
String s3 = text.substring(help1+1,posH);

//Die Hilfsvariable help2 wird -1 gesetzt.
int help2=-1;

//Im zweiten String wird die Position des ersten "*"
//oder "/" ermittelt:
if(s2.indexOf("*") >= 0)
  help2 = s2.indexOf("*");

if(s2.indexOf("/") >= 0 && s2.indexOf("/") < help2 ||
    help2==-1)
```

```
            help2=s2.indexOf("/");
          else //nach "^" gibt es nur mehr eine Zahl, kein "*"
            //und "/"
            help2=s2.length();

          //s1 wird nun aus String vor dem "^" gebildet.
          s1 = s1.substring(0, s1.length()-s3.length());

          //s4 enthaelt den Exponenten
          String s4 = text.substring(posH+1, help2+posH+1);

          //s2 beinhaltet den Teil nach dem Exponenten.
          s2 = s2.substring(s4.length(), s2.length());

          //System.out.println("s4=" + s4);
          //System.out.println("s3=" + s3);
          //System.out.println("s2=" + s2);
          //System.out.println("s1=" + s1);

          //Zuletzt wird nun aus den ermittelten Strings der
          //zurückzugebende zusammengestellt.
          ersetze = s1 + "pow(" + s3 + "," + s4 + ")" + s2;

      return ersetze;
      }
      else
        return text;
    }

    /** Die Funktion destroy wird aufgerufen, wenn das Applet geschlossen
     * wird. Falls das Programm Sound ausgibt, soll diese Ausgabe hier
     * gestoppt werden.
     *
     * @param: -
     * @return: -
     * @author: David Stadler
     * @date: 19.3.2011
     */
    public void destroy() {
      f.stop();
    }

}
```

Source Code (FourierHelp.java)

```
import java.io.InputStream;
import sun.audio.AudioData;
import sun.audio.AudioPlayer;
import sun.audio.ContinuousAudioDataStream;

public class FourierHelp {
  //http://www.phy.ntnu.edu.tw/ntnujava/index.php?topic=1060.0

  /**
   * Grundlage fuer diese Klasse ist http://www.phy.ntnu.edu.tw/
   * ntnujava/index.php?topic=1060.0. Unter diesem Link sind fuer
   * diese Diplomarbeit nuetzliche Information erhaeltlich. Unter
   * anderem wird beschrieben, wie ein sinus- Oszillator in Java
   * programmiert werden kann, bei dem entsprechend die Frequenz
   * und die Amplitude geaendert werden koennen.
   */

  //Hier werden die notwendigen Variablen deklariert und teilweise
  //auch definiert.

  //Das Integer periode speichert die Periodendauer.
  public int periode;

  //Das byte- Array "wiedergabe" dient der Soundausgabe des aktuell-
  //angewaehlten Signals.
  private byte wiedergabe[]=new byte[periode];

  //Die Variable "amplitude" speichert die Amplitude der Funktion.
  private double amplitude;

  //Das double- Array "yWerte" speichert die y- Werte des akutellen
  //Signals.
  private double yWerte[]=new double[periode];

  //Das double "einheit" definiert die Aufloesung des Signals. Je
  //hoeher diese ist, desto groesser ist das aktuelle Signal
  //dargestellt.
  private double einheit=33000.;

  //Definition des Kreisfrequenz.
```

```
private double omega=2.*Math.PI/periode;

//wt ist eine Hilfsvariable fuer die Kreisfrequenz omega.
private double wt;

//Hier wird das obere Limit der Fourier- Koeffizienten definiert.
private int maxKoeff = 100;

//Deklaration der Arrays, welche die aks und bks speichern.
public double bk[]=new double[maxKoeff], ak[]=new double[maxKoeff];

//InputStream soundStr dient der Soundwiedergabe.
private InputStream soundStr=null;



/** Dies ist der Konstruktor der Klasse FourierHelp. Wenn das Objekt
 * in FourierApp erzeugt wird, wird diese Funktion aufgerufen.
 *
 * @param: double per (uebergebene Periodendauer)
 * @return: -
 * @author: David Stadler
 * @date: 9.3.2011
 */
FourierHelp(double per){
  //Der Wert der Periodendauer, welcher von FourierApp uebergeben
  //wird, wird hier ebenso angenommen.
  periode = (int)per;
}

/** Wenn in FourierApp die Ausgangsfrequenz geaendert wird, so wird
 * die Funktion grundFrequenz aufgerufen. Hierbei wird die
 * Periodendauer neu errechnet und die Arrays "yWerte" und
 * "wiedergabe" neu deklariert. Weiters werden diese durch die
 * Funktion "berechne" definiert und die Soundausgabe beginnt neu.
 *
 * @param: double grundf (uebergebene Grundfrequenz)
 * @return: -
 * @author: David Stadler
 * @date: 9.3.2011
 */
public void grundFrequenz(double grundf){
  //Berechnung der Periodendauer und des omegas. die Variable
```

```
    //"einheit" definiert die Aufloesung.
    periode=(int)(einheit/grundf);
    omega=2.*Math.PI/periode;

    //Es werden die Arrays "yWerte" und "wiedergabe" neu deklariert.
    yWerte=new double[periode];
    wiedergabe=new byte[periode];

    //Berechnung der neuen Fourierkoeffizienten.
    berechne();

    //Soundausgabe neustarten.
    if(soundStr!=null){
      stop();
      start();
    }
  }


  /** Die in dieser Klasser emittelten y- Werte werden an die Klasse
   * FourierApp als double- Array zurueckgegeben.
   *
   * @param: -
   * @return: double[]
   * @author: David Stadler
   * @date: 9.3.2011
   */
  public double[] yVal(){
    return yWerte;
  }


  /** In der Funtion set_bk werden die Fourierkoeffizienten (bks) in
   * Abhaengigkeit vom Index und vom Wert gespeichert.
   *
   * @param: int i (welcher Koeffizient gesetzt werden soll),double
   * val (entsprechender Wert)
   * @return: -
   * @author: David Stadler
   * @date: 9.3.2011
   */
  public void set_bk(int i,double val){
    //Der Wert wird nur gesetzt, wenn i nicht das Maximum der
```

```
    //Fourierkoeffizienten ueberschreitet.
    if(i<maxKoeff)
      bk[i]=val;
  }

  /** In der Funtion set_ak werden die Fourierkoeffizienten (aks)
   * in Abhaengigkeit vom Index und vom Wert gespeichert.
   *
   * @param: int i (welcher Koeffizient gesetzt werden soll),
   *  double val (entsprechender Wert)
   * @return: -
   * @author: David Stadler
   * @date: 9.3.2011
   */
  public void set_ak(int i,double val){
    //Der Wert wird nur gesetzt, wenn i nicht das Maximum der
    //Fourierkoeffizienten ueberschreitet.
    if(i<maxKoeff)
      ak[i]=val;
    //soundSetup=false;
  }

  /** In der Funtion get_bk werden die Fourierkoeffizienten (bks) in
   * Abhaengigkeit vom Index wiedergeholt.
   *
   * @param: int i (Index, welcher Koeffizient geholt werden soll)
   * @return: double (der Fourierkoeffizient als double- Wert)
   * @author: David Stadler
   * @date: 9.3.2011
   */
  public double get_bk(int i){
    //Koeffizient soll nur returned werden, wenn i nicht das Maximum
    //der Fourierkoeffizienten ueberschreitet.
    if(i<maxKoeff) return bk[i];
    else return 0.;//ansonsten wird 0 zurueckgegeben.
  }

  /** In der Funtion get_ak werden die Fourierkoeffizienten (aks) in
   * Abhaengigkeit vom Index wiedergeholt.
   *
   * @param: int i (Index, welcher Koeffizient geholt werden soll)
   * @return: double (der Fourierkoeffizient als double- Wert)
   * @author: David Stadler
```

```
 * @date: 9.3.2011
 */
public double get_ak(int i){
  //Koeffizient soll nur returned werden, wenn i nicht das Maximum
  //der Fourierkoeffizienten ueberschreitet.
  if(i<maxKoeff) return ak[i];
  else return 0.;//ansonsten wird 0 zurueckgegeben.
}


/** Wenn die Funktion start() aufgerufen wird, soll die Tonausgabe
 *  beginnen-> soundStr wird gesetzt.
* Damit hier keine Fehlermeldungen kommen, muss rt.jar in den Ordner
* lib kopiert werden und build path angepasst werden, also die jar-
* Datei veroeffentlicht werden.
*
* @param: -
* @return: -
* @author: David Stadler
* @date: 12.3.2011
*/
public void start(){
  //Zunaechst werden die Fourierkoeffizienten berechnet.
  berechne();

  //Der folgenden Block wird mit try-catch surrounded, damit das
  //Programm bei einem Fehler trotzdem stabil weiterlaeft.
  try{
    //Wenn soundStr definiert ist, wird dieser gestoppt.
    if(soundStr!=null)
      AudioPlayer.player.stop(soundStr);

    //Danach wird ein neuer Soundstream in Abhaengigkeit vom
    //Array "wiedergabe"erstellt...
    soundStr = new ContinuousAudioDataStream(new AudioData(wiedergabe));

    //...und gestartet.
    AudioPlayer.player.start(soundStr);
  }catch(SecurityException e){
  }
}

/** Wenn die Funktion stop() aufgerufen wird, soll die Tonausgabe
```

```
     * beendet werden-> soundStr wird auf null gesetzt.
    *
    * @param: -
    * @return: -
    * @author: David Stadler
    * @date: 12.3.2011
    */
    public void stop(){
      //Der Soundstream wird nur beendet, wenn dieser nicht null ist.
      if(soundStr!=null)
      try{//Der folgenden Block wird mit try-catch surrounded, damit
        //das Programm bei einem Fehler trotzdem stabil weiterlaeft.
        AudioPlayer.player.stop(soundStr);

        //soundStr auf null setzen.
        soundStr=null;
      }catch(SecurityException e){
      }
    }

    /** Die Funktion berechne() fuegt alle uebergebenen aks und bks
     * zusammen und erstellt ein Array yWerte mit den entsprechenden,
     * transformierten Werten von diesen Fourierkoeffizienten.
    *
    * @param: -
    * @return: -
    * @author: David Stadler
    * @date: 12.3.2011
    */
    public void berechne() {
      //Zunaechst wird die Amplitude fuer die akustische Wiedergabe
      //angepasst.
        amplitude=1000.*(bk[0]+1.0);

          //In dieser Schleife wird das Array yWerte ermittelt.
          for (int i = 0; i < periode; i++){
            //Die Hilfsvariable y wird auf 0 gesetzt.
            double y=0;

            //Fuer jedem Fourierkoeffizient wird eine eigene
            //Variable wt berechnet. Diese besteht aus dem
            //Produkt aus i und der Kreisfrequenz.
            wt=i*omega;
```

```
            //y nimmt zunaechst den Wert von a0/2 an. Siehe
            //Definition der Fourierreihe.
            y += ak[0]/2.0;

            //Zum zuvor berechneten Gleichanteil wird noch der
            //fehlende Fourierkoeffizient dazuaddiert.
            for (int j = 1; j < maxKoeff; j++)
              y += (ak[j]*Math.cos(wt*j)+bk[j]*
                  Math.sin(wt*j));

            //Das Array "wiedergabe" speichert als byte- array
            //die Werte fuer die Wiedergabe (externe SW zur
            //Transformation)
            wiedergabe[i] = int2ulaw((int)(y*amplitude));

            //Schliesslich wird das Array yWerte definiert.
            yWerte[i]=y;
          }
    }

    /** Die Funktion int2ulaw dient der Umwandlung der aktuellen
     * Information in ein sound- Objekt.
     *
     * @Quelle: 9.3.2011 http://www.phy.ntnu.edu.tw/ntnujava/index.php?topic=
     * @param: int ch
     * @return: byte
     * @date: 12.3.2011
     */
    public static byte int2ulaw(int ch) {
      int mask;
      if (ch < 0) {
        ch = -ch;
        mask = 0x7f;
      }else mask = 0xff;

      if (ch < 32) ch = 0xF0 | 15 - (ch/2);
        else if (ch < 96) ch = 0xE0 | 15 - (ch-32)/4;
        else if (ch < 224) ch = 0xD0 | 15 - (ch-96)/8;
        else if (ch < 480) ch = 0xC0 | 15 - (ch-224)/16;
        else if (ch < 992 ) ch = 0xB0 | 15 - (ch-480)/32;
        else if (ch < 2016) ch = 0xA0 | 15 - (ch-992)/64;
        else if (ch < 4064) ch = 0x90 | 15 - (ch-2016)/128;
```

```
      else if (ch < 8160) ch = 0x80 | 15 - (ch-4064)/256;
      else ch = 0x80;
    return (byte)(mask & ch);
  }
}
```

Eidesstaattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit „Dynamische Fourier - Synthese zur Beschreibung musikalischer und physikalischer Zusammenhänge in Java“ selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Wien, im Juni 2011

David Stadler