Schnelle Fourier-Transformation

Jan Metzner Alexander Schaal

1. November 2002

Mathematische Ausarbeitung für das Fach "Seminar Numerische Mathematik"
von Prof. Dr. Krautwald und Prof. Dr. Gleich
an der Fachhochschule München

Jan Metzner

Alexander Schaal

Inhaltsverzeichnis

1 Historie **3**
1.1 Historie der Fourier-Transformation . . . 3
1.2 Historie der Schnellen Fourier-Transformation . . . 3

2 Fourier-Transformation **4**
2.1 Transformationsanalysis . . . 4
2.2 Anwendungsgebiete . . . 4
2.3 Allgemeine Erklärung der Fourier-Transformation . . . 4

3 Schnelle Fourier-Transformation **9**
3.1 Matrixdarstellung . . . 9
3.2 Signalflußgraph . . . 10
3.3 Bitumkehr . . . 10
3.4 Vollständiges Beispiel mit $N = 8$ . . . 11

4 Rechenzeitvergleich **14**

5 Grenzen der Fourier-Transformation **15**

6 Umsetzung in ein C++ Programm **16**
6.1 Programmbeschreibung . . . 16
6.2 Beispiele aus der Aufgabenstellung . . . 17
6.2.1 Beispiel 1: Synthese . . . 17
6.2.2 Beispiel 2: Analyse . . . 20

7 Quellen und Anhang **24**

1 Historie

1.1 Historie der Fourier-Transformation

Der Ursprung der Fourierreihen geht auf das Problem der schwingenden Saite zurück. 1747 war von d'Alembert die Wellengleichung zur Bestimmung der Schwingung einer Saite hergeleitet worden. Wenn die Saite ursprünglich in Ruhestellung ist, dann wird ihre künftige Bewegung vollständig durch die anfängliche Auslenkung aus dem Gleichgewichtszustand bestimmt. Euler wies darauf hin, dass es keinen physikalischen Grund für die Forderung gibt, dass die Ausgangsposition der Saite durch eine einzelne Funktion gegeben wird. Verschiedene Abschnitte der Saite könnten sehr wohl durch verschiedene Formeln beschrieben sein, solange sie sich nur glatt aneinanderfügen.

Bernoulli

1755 fand Bernoulli eine andere Lösungsform für die schwingende Saite, indem er stehende Wellen verwendete. In jedem Augenblick hat die Saite die Form einer Sinuskurve, und an jedem Punkt auf der Saite wird die Bewegung in der Zeit durch eine Kosinusfunktion der Zeit gegeben.
Bernoullis Methode, das allgemeine Problem zu lösen, bestand darin, dass er eine unendliche Anzahl stehender Wellen summierte. Dies erforderte, dass die anfängliche Auslenkung die Summe einer unendlichen Anzahl von Sinusfunktionen sein musste.

$$y = \alpha \sin(\pi x/a) + \beta \sin(2\pi x/a) + ...$$

Bernoullis stehende Welle ist eine Funktion von zwei Variablen (Zeit und Ort), welche die Besonderheit hat, in eine Funktion des Ortes multipliziert mit einer Funktion der Zeit zu zerfallen. Damit ein solches Produkt die Gleichung für eine schwingende Saite erfüllt, müssen beide Faktoren entweder Sinus- oder Kosinusfunktionen sein. Die Randbedingungen (die Enden fixiert, Anfangsgeschwindigkeit null) und die Länge der Saite bestimmen dann, dass sie von der Form $sin(nx)$ und $cos(mt)$ sein müssen.
Als Fourier seine Gleichung der Wärmeleitung herleitete, entdeckte er, dass sie ebenfalls spezielle Lösungen hat, die sich in eine Funktion des Orts multipliziert mit einer Funktion der Zeit zerlegen lassen. In diesem Fall ist die Zeitfunktion exponential statt trigonometrisch, doch wenn der Körper, dessen Wärmeleitung wir betrachten, rechteckig ist, erhalten wir wieder trigonometrische Ortsfunktionen. Wärend Bernoulli nur an jene Funktionen dachte, die durch einen einzelnen analytischen Ausdruck gebildet werden, bezog Fourier ausdrücklich auch jene Funktionen mit ein, die stückweise durch mehrere verschiedene Formeln gegeben sind. Er sah, dass die endgültige Koeffizientenformel und die Herleitung mittels Orthogonalität der Sinusfunktion für jeden Graphen sinnvoll bleibt, der ein bestimmtes Gebiet begrenzt. Er verkündete, dass jede Temperaturverteilung, also jeder Graph, unabhängig davon, aus wie vielen Einzelstücken er besteht, durch eine Reihe von Sinus- und Kosinusfunktionen dargestellt wird.

Fourier

1.2 Historie der Schnellen Fourier-Transformation

Während einer Sitzung des wissenschaftlichen Beratungskomitees des US-Präsidenten stellte Richard Garwin fest, dass John Tukey sich mit der Erstellung von Programmen für die Fourier Transformation beschäftigte. Garwin war für seine Forschungsarbeit auf der Suche nach einer schnellen Methode zur Berechnung der Fourier-Transformation. Tukey erklärte Garwin im Wesentlichen das, was später zu dem berühmten Cooley-Tukey-Algorithmus führte.
Garwin ließ das Verfahren im IBM Forschungszentrum programmieren. Dort wurde James Cooley mit der Bearbeitung der Aufgabe beauftragt. Nachdem er das Programm für Garwin fertiggestellt hatte, häuften sich die Nachfragen für Kopien für das Programm und Cooley wurde zu einer Veröffentlichung darüber gebeten.
1965 publizierte Cooley und Tukey den jetzt weltberühmten Aufsatz "An Algorithm for the Mashine Calculation of Complex Fourier Series" in "Mathematics of Computation". Nach der Publikation wurde bekannt, dass sich auch andere Personen ähnlicher Verfahren bedienten.

2 Fourier-Transformation

2.1 Transformationsanalysis

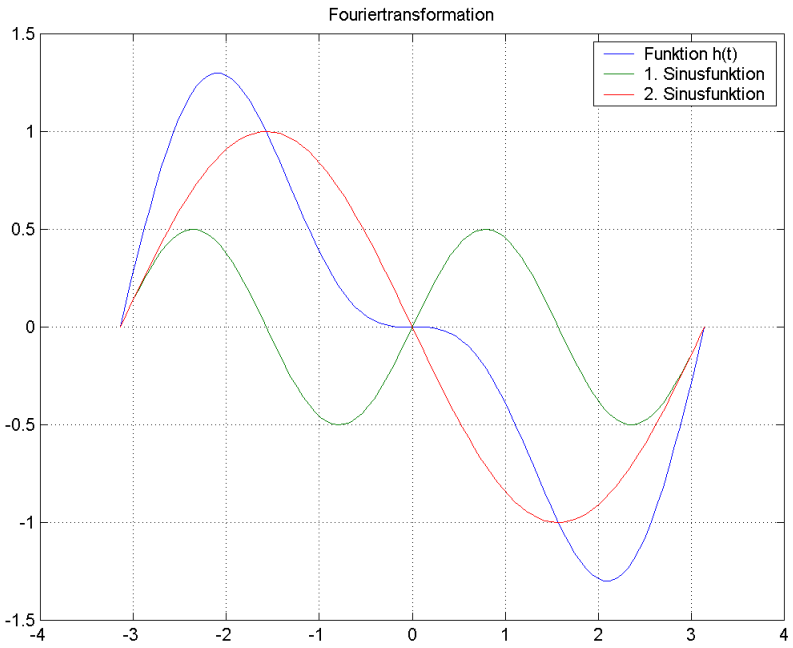
Im allgemeinen führen Transformationen zu einer Vereinfachung der Problemlösung. In der Praxis hat man eine Problemstellung, die durch direkte Analysis nur sehr aufwendig zu lösen ist. Führt man aber eine Transformation aus, kann man relativ einfach von der transformierten Problemstellung auf eine transformierte Lösung kommen. Als letzten Schritt muss man nur noch diese rücktransformieren.
Ein einfaches Beispiel ist die Division von Y=X/Z. Solch eine Division kann sehr aufwendig sein. Wenn man die Werte X und Z transformiert nach log(X) und log(Z), so muss man sie nur subtrahieren: log(Y)=log(X)-log(Z). Nun fehlt noch die Rücktransformation von log(Y) nach Y.
Genau dies ist die Idee der Fourier-Transformation.

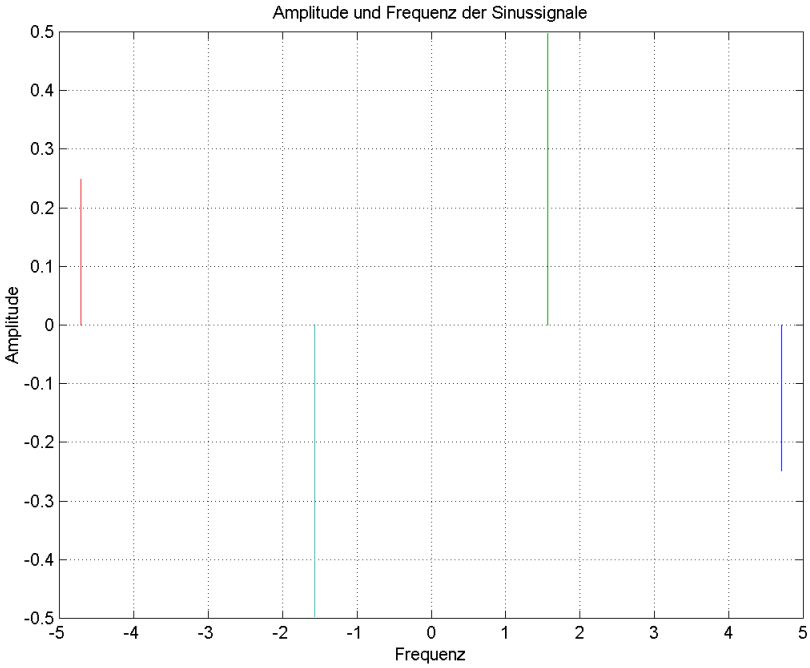
2.2 Anwendungsgebiete

Die Fourier-Transformation hat sich zur Problemvereinfachung in vielen wissenschaftlichen Bereichen als sehr effektiv erwiesen. Eine sehr einfache Anwendung ist die Frequenzfilterung. Da man eine Funktion, bzw. ein Signal vom Zeitbereich in den Frequenzbereich transformiert, kann man nun leicht Frequenzen verändern, löschen oder hinzufügen, bevor man das Signal wieder rücktransformiert. Weitere Anwendungsgebiete sind Lineare Systeme, Antennen, Optik, Stochastische Prozesse, Wahrscheinlichkeitstheorie, Quantenphysik oder Randwert-Probleme.

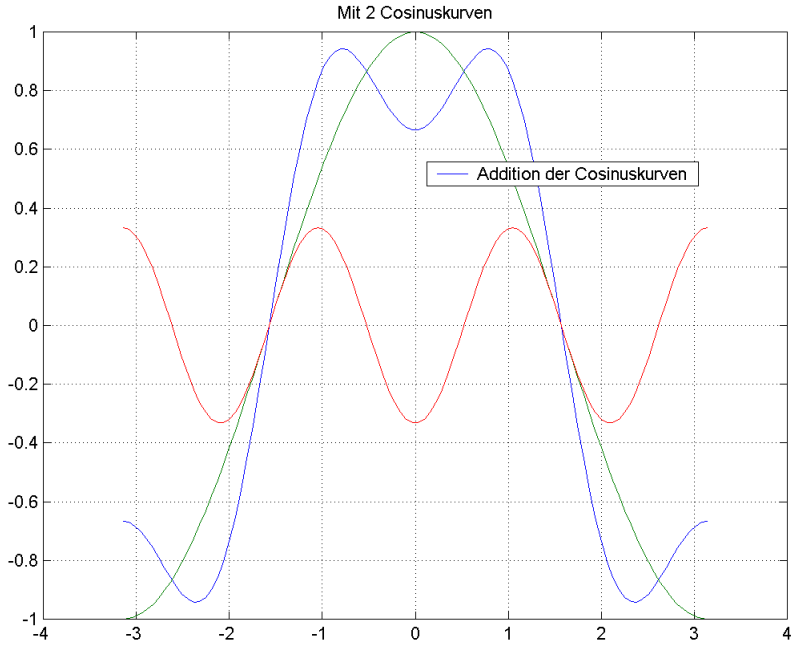
2.3 Allgemeine Erklärung der Fourier-Transformation

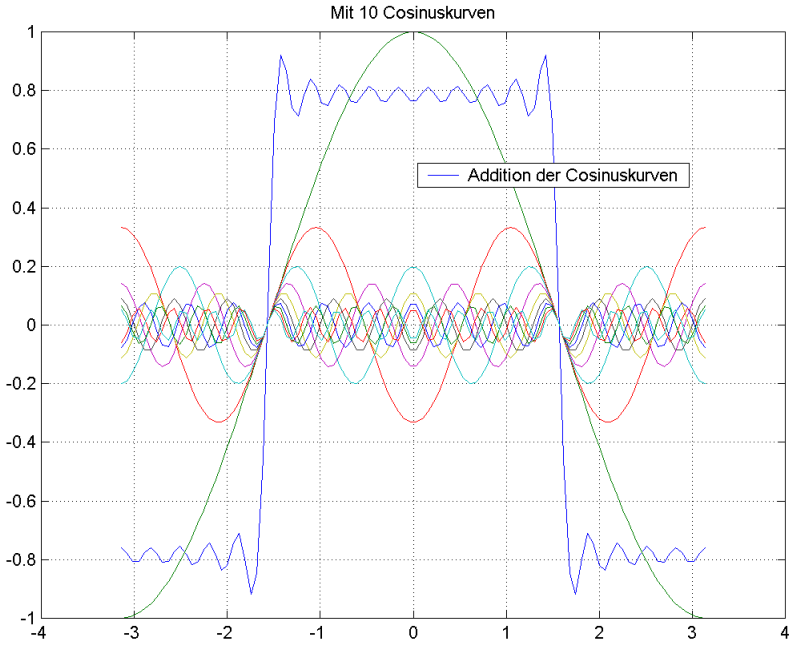
Im Gegensatz zur eindimensionalen Logarithmus-Transformation, die einen einzelnen Wert X in einen einzelnen Wert log(X) transformiert, transformiert die Fourier-Transformation eine Funktion einer Variable in eine Funktion einer anderen Variable, jeweils definiert von $-\infty$ bis $+\infty$.
Die Hauptaufgabe der Fourier-Transformation besteht darin, ein beliebiges Signal in eine Summe von Sinusfunktionen unterschiedlicher Frequenz, Amplitude und Phase zu zerlegen. Folgende zwei Plotts sollen dies verdeutlichen:

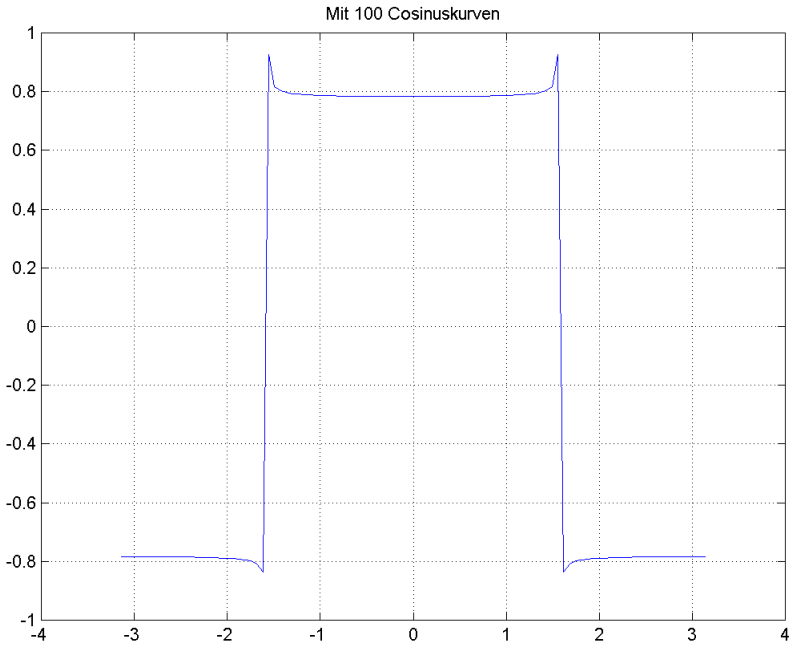


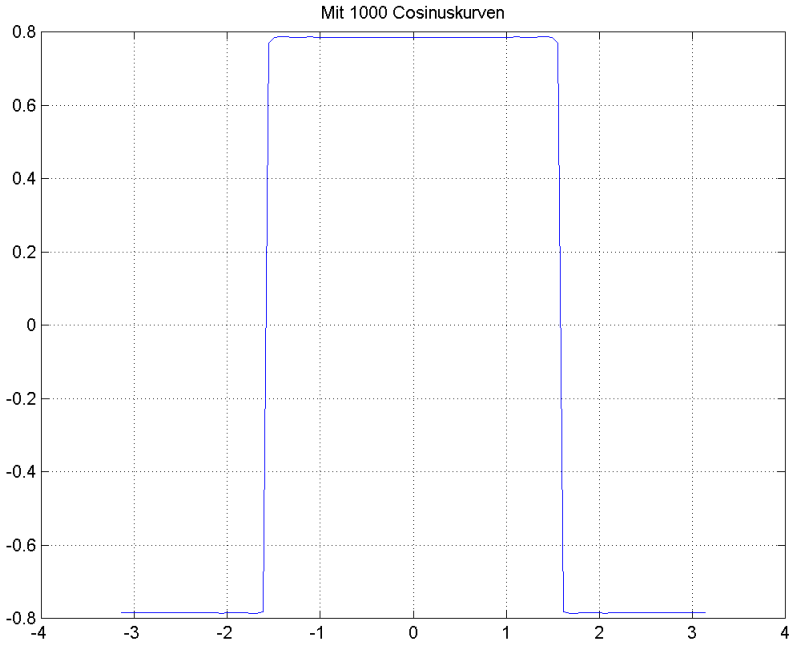


Auch ein Rechtecksignal kann man in Sinussignale, bzw. Cosinussignale zerlegen. Oder eben auch erzeugen:

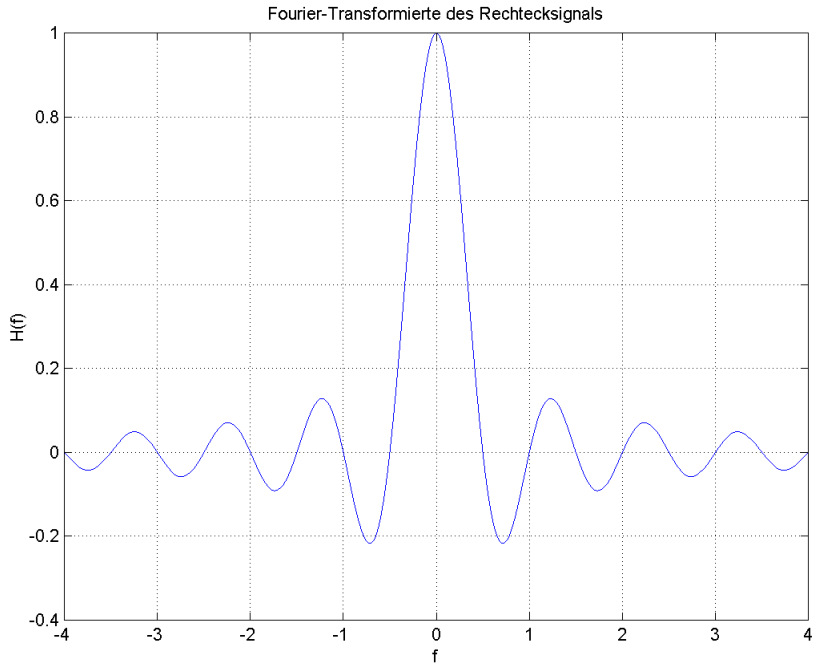



Mit 10 Cosinuskurven
Addition der Cosinuskurven
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-4
-3
-2
-1
0
1
2
3
4


Mit 100 Cosinuskurven
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-4
-3
-2
-1
0
1
2
3
4



Würde man nun wieder alle Frequenzen mit ihren Amplituden wie oben in ein Diagramm übertragen und deren Spitzen miteinander verbinden, bzw. interpolieren, so erhält man die Fourier-Transformierte der Rechteckfunktion:



Das Fourierintegral ist bestimmt durch

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-i2\pi ft} dt.$$

H(f) ist die Fourier-Transformierte von h(t). h(t) wird normalerweise als Funktion der Variablen Zeit und H(f) als Funktion der Variablen Frequenz betrachtet. Im Allgemeinen ist die Fourier-Transformierte eine komplexe Größe:

$$H(f) = R(f) + iI(f) = |H(f)|e^{i\Theta(f)},$$

wobei R(f) der Realteil und I(f) der Imaginärteil der Fourier-Transformierten sind. $|H(f)|$ ist das Amplitudenspektrum von h(t) und ist gegeben durch

$$\sqrt{R^2(f) + I^2(f)},$$

$\Theta(f)$ ist das Phasenspektrum der Fourier-Transformierten und ist gegeben durch

$$arctan\left[\frac{I(f)}{R(f)}\right].$$

Das inverse Fourierintegral ist bestimmt durch

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{i2\pi ft} dt.$$

Für einen math. Lösungsansatz ändert sich also nur ein Minuszeichen im Exponenten der e-Funktion. Transformiert man nun eine Funktion und wendet dann die inverse Transformation an, so wird man feststellen, dass man um einen konstanten Faktor daneben liegt. Dieser Faktor ist $\frac{1}{2\pi}$. Folglich muss die exakte Rücktransformation folgendermaßen aussehen:

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(f) e^{i2\pi ft} dt.$$

Anmerkung: Berechnet man von 0 bis 2π die inverse Fourier-Transformation mit N äquidistanten Stützstellen, so ist der Faktor 1/N.

3 Schnelle Fourier-Transformation

Die schnelle Fourier-Transformation ist ein Algorithmus, der die Berechnung der diskreten Fourier-Transformation schneller als alle anderen bekannten Algorithmen ermöglicht. Allerdings gilt dies nur für Zweierpotenzen.

3.1 Matrixdarstellung

Aus der diskreten Fourier-Transformation folgt:

$$c_k = \sum_{j=0}^{N-1} y_j e^{-i2\pi kj/N},\ k = 0, 1, ..., N-1$$

$$\text{mit } \omega = e^{-i\frac{2\pi}{N}} \text{ folgt: } c_k = \sum_{j=0}^{N-1} y_j \omega^{kj}$$

Aus dieser Summe erhält man N Gleichungen. Als Beispiel betrachten wir den Fall N = 4.

$$\begin{aligned} c_0 &= y_0\omega^0 + y_1\omega^0 + y_2\omega^0 + y_3\omega^0 \\ c_1 &= y_0\omega^0 + y_1\omega^1 + y_2\omega^2 + y_3\omega^3 \\ c_2 &= y_0\omega^0 + y_1\omega^2 + y_2\omega^4 + y_3\omega^6 \\ c_3 &= y_0\omega^0 + y_1\omega^3 + y_2\omega^6 + y_3\omega^9 \end{aligned}$$

Die vier Gleichungen lassen sich als Marix schreiben. Unter Berücksichtigung der Beziehung $\omega^{nk} = \omega^{nk\ mod(N)}$ und $\omega^0 = 1$ erhält man:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}, c = W\ y$$

Indem man zuerst die x-Werte mit geraden Indizes in aufsteigender Reihenfolge schreibt, danach diejenigen mit ungeraden Indizes und die Reihenfolge der y-Werte unverändert läßt, erhält man:

$$\left[\begin{array}{c} c_0 \\ c_2 \\ \hline c_1 \\ c_3 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ \hline 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega \end{array}\right] \left[\begin{array}{c} y_0 \\ y_1 \\ \hline y_2 \\ y_3 \end{array}\right],\ \hat{c} = \hat{W}\ y$$

Die zweireihigen Untermatrizen haben die Beziehungen $\hat{W}_{11} = \hat{W}_{12},\ \hat{W}_{22} = \omega^2\hat{W}_{21}$. Diese Beziehung legt nahe, die Matrix $\hat{W}$ als Produkt zweier Blockmatrizen zu schreiben:

$$\left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ \hline 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & \omega^2 \end{array}\right] \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 0 & \omega^2 & 0 \\ 0 & \omega & 0 & \omega^3 \end{array}\right]$$

Durch diese Schreibweise kann der Vektor $\hat{c}$ in zwei Teilschritten berechnet werden:
1. Teilschritt: Multiplikation von y mit der zweiten Untermatrix ergibt den Hilfsvektor z. Dabei wurde die Tatsache $\omega^2 = -1 = -\omega^0$ und $\omega^3 = -\omega^1$ ausgenutzt.

$$z = \left[\begin{array}{c} y_0 + y_1 \\ y_1 + y_3 \\ \hline y_0 + y_2\omega^2 \\ y_1\omega + y_3\omega^3 \end{array}\right] = \left[\begin{array}{c} y_0 + y_1 \\ y_1 + y_3 \\ \hline y_0 - y_2 \\ (y_1 - y_3)\omega \end{array}\right]$$

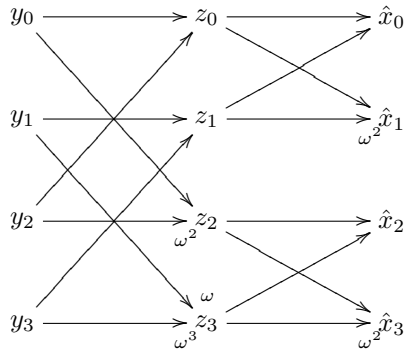
2. Teilschritt: Multiplikation von z mit der ersten Untermatrix.

$$\hat{c} = \left[\begin{array}{c} z_0 + z_1 \\ z_0 + z_1\omega^2 \\ \hline z_2 + z_3 \\ z_2 + z_3\omega^2 \end{array}\right] = \left[\begin{array}{c} z_0 + z_1 \\ z_0 - z_1 \\ \hline z_2 + z_3 \\ z_2 - z_3 \end{array}\right]$$

Diese Multiplikationen entsprechen einer Fourier-Transformationen mit N=2 (ω^2 wurde mitgeführt um die Veralgemeinerung zu veranschaulichen). Diese systematische Reduktion, die aber nur funktioniert, falls N eine Potenz von 2 ist, ist die zentrale Idee der schnellen Fourier-Transformation.

3.2 Signalflußgraph

Man kann die Matrixfaktorisierung graphisch interpretieren.
Pfeil: Die Werte werden mit einer Potenz von ω multipliziert und dann übertragen. Der Faktor wird an die Pfeilspitze geschrieben. Falls dieser fehlt, bedeutet das, dass er $\omega^\alpha = 1$ ist.
Knoten: Die ankommenden Werte werden aufsummiert.



3.3 Bitumkehr

Durch die Umsortierung der Matrix nach geraden und ungeraden Elementen erhält man nach der Matrixfaktorisierung $\hat{x}$ statt x. Nun muß man $\hat{x}$ rücktransformieren. Wenn man sich die Indizes nicht in Dezimal-, sondern in Binärdarstellung ansieht, kann man die nötigen Vertauschungen leicht nachvollziehen.

$$\overbrace{\left[\begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array}\right]}^{c} \Rightarrow \overbrace{\left[\begin{array}{c} 00 \\ 10 \\ 01 \\ 11 \end{array}\right]}^{\hat{c}}$$

Zuerst erscheinen die Indizes mit geraden Zahlen (letzte Binärstelle ist 0), danach die ungeraden (letzte Binärstelle 1). Die Rückvertauschung kann einfach bewerkstelligt werden, indem man den binären Index spiegelt (10 wird zu 01).
Dies funktioniert auch für größere N.

Beispiel $N = 16$:

$$\overbrace{\begin{bmatrix}0000\\0001\\0010\\0011\\0100\\0101\\0110\\0111\\1000\\1001\\1010\\1011\\1100\\1101\\1110\\1111\end{bmatrix}}^{\text{c}} \Rightarrow \overbrace{\begin{bmatrix}0000\\0010\\0100\\0110\\1000\\1010\\1100\\1110\\0001\\0011\\0101\\0111\\1001\\1011\\1101\\1111\end{bmatrix}}^{\text{1.Schritt}} \Rightarrow \overbrace{\begin{bmatrix}0000\\0100\\1000\\1100\\0010\\0110\\1010\\1110\\0001\\0101\\1001\\1101\\0011\\0111\\1011\\1111\end{bmatrix}}^{\text{2.Schritt}} \Rightarrow \overbrace{\begin{bmatrix}0000\\1000\\0100\\1100\\0010\\1010\\0110\\1110\\0001\\1001\\0101\\1101\\0011\\1011\\0111\\1111\end{bmatrix}}^{\text{3.Schritt}} \Rightarrow \overbrace{\begin{bmatrix}0000\\1000\\0100\\1100\\0010\\1010\\0110\\1110\\0001\\1001\\0101\\1101\\0011\\1011\\0111\\1111\end{bmatrix}}^{\text{4.Schritt}} \Rightarrow \overbrace{\begin{bmatrix}0000\\0001\\0010\\0011\\0100\\0101\\0110\\0111\\1000\\1001\\1010\\1011\\1100\\1101\\1110\\1111\end{bmatrix}}^{\text{Umkehr=c}}$$

3.4 Vollständiges Beispiel mit $N = 8$

Bei $N = 8$ ist $\omega = e^{-2\pi i/8}$. Die Ausgangssituation ist folgende:

$$\left[\begin{array}{c}c_0\\c_1\\c_2\\c_3\\\hline c_4\\c_5\\c_6\\c_7\end{array}\right] = \left[\begin{array}{cccc|cccc}1&1&1&1&1&1&1&1\\1&\omega&\omega^2&\omega^3&\omega^4&\omega^5&\omega^6&\omega^7\\1&\omega^2&\omega^4&\omega^6&1&\omega^2&\omega^4&\omega^6\\1&\omega^3&\omega^6&\omega&\omega^4&\omega^7&\omega^2&\omega^5\\\hline 1&\omega^4&1&\omega^4&1&\omega^4&1&\omega^4\\1&\omega^5&\omega^2&\omega^7&\omega^4&\omega&\omega^6&\omega^3\\1&\omega^6&\omega^4&\omega^2&1&\omega^6&\omega^4&\omega^2\\1&\omega^7&\omega^6&\omega^5&\omega^4&\omega^3&\omega^2&\omega\end{array}\right] \left[\begin{array}{c}y_0\\y_1\\y_2\\y_3\\\hline y_4\\y_5\\y_6\\y_7\end{array}\right], \; c = W\, y$$

Nach der Zeilenpermutation und der Reduktion der Exponenten von ω um modulo 8 sieht die Matrix $\hat{W}$ so aus:

$$\left[\begin{array}{c}c_0\\c_2\\c_4\\c_6\\\hline c_1\\c_3\\c_5\\c_7\end{array}\right] = \left[\begin{array}{cccc|cccc}1&1&1&1&1&1&1&1\\1&\omega^2&\omega^4&\omega^6&1&\omega^2&\omega^4&\omega^6\\1&\omega^4&1&\omega^4&1&\omega^4&1&\omega^4\\1&\omega^6&\omega^4&\omega^2&1&\omega^6&\omega^4&\omega^2\\\hline 1&\omega&\omega^2&\omega^3&\omega^4&\omega^5&\omega^6&\omega^7\\1&\omega^3&\omega^6&\omega&\omega^4&\omega^7&\omega^2&\omega^5\\1&\omega^5&\omega^2&\omega^7&\omega^4&\omega&\omega^6&\omega^3\\1&\omega^7&\omega^6&\omega^5&\omega^4&\omega^3&\omega^2&\omega\end{array}\right] \left[\begin{array}{c}y_0\\y_1\\y_2\\y_3\\\hline y_4\\y_5\\y_6\\y_7\end{array}\right], \; \hat{c} = \hat{W}\, y$$

Die Matrix $\hat{W}$ weist eine ähnliche Blockstruktur auf, wie im Fall $N = 4$. Wie schon vorher kann man auch diese Matrix als Produkt zweier Matrizen darstellen:

$$\hat{W} = \left[\begin{array}{cccc|cccc}1&1&1&1&0&0&0&0\\1&\omega^2&\omega^4&\omega^6&0&0&0&0\\1&\omega^4&1&\omega^4&0&0&0&0\\1&\omega^6&\omega^4&\omega^2&0&0&0&0\\\hline 0&0&0&0&1&1&1&1\\0&0&0&0&1&\omega^2&\omega^4&\omega^6\\0&0&0&0&1&\omega^4&1&\omega^4\\0&0&0&0&1&\omega^6&\omega^4&\omega^2\end{array}\right] \left[\begin{array}{cccc|cccc}1&0&0&0&1&0&0&0\\0&1&0&0&0&1&0&0\\0&0&1&0&0&0&1&0\\0&0&0&1&0&0&0&1\\\hline 1&0&0&0&\omega^4&0&0&0\\0&\omega&0&0&0&\omega^5&0&0\\0&0&\omega^2&0&0&0&\omega^6&0\\0&0&0&\omega^3&0&0&0&\omega^7\end{array}\right]$$

Die erste Untermatrix kann man mit demselben Schema nochmal unterteilen, so dass:

$$\begin{bmatrix} c_0 \\ c_4 \\ c_2 \\ c_6 \\ \hline c_1 \\ c_5 \\ c_3 \\ c_7 \end{bmatrix} = \hat{\hat{W}} \left[\begin{array}{c} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \hline y_4 \\ y_5 \\ y_6 \\ y_7 \end{array}\right], \; \hat{\hat{c}} = \hat{\hat{W}}\, y$$

mit

$$\hat{\hat{W}} = \left[\begin{array}{cccc|cccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & \omega^4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \omega^4 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \omega^4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \omega^4 \end{array}\right] \left[\begin{array}{cccc|cccc} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & \omega^4 & 0 & 0 & 0 & 0 & 0 \\ 0 & \omega^2 & 0 & \omega^6 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & \omega^4 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega^2 & 0 & \omega^6 \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & \omega^4 & 0 & 0 & 0 \\ 0 & \omega & 0 & 0 & 0 & \omega^5 & 0 & 0 \\ 0 & 0 & \omega^2 & 0 & 0 & 0 & \omega^6 & 0 \\ 0 & 0 & 0 & \omega^3 & 0 & 0 & 0 & \omega^7 \end{array}\right]$$

Der Vektor $\hat{\hat{c}}$ kann somit in drei Teilschritten berechnet werden:
1. Teilschritt: Multiplikation von y mit der dritten Untermatrix ergibt den Hilfsvektor z. Dabei wurde die Tatsache $\omega^{4+\mu} = -\omega^\mu$ für $\mu = 0, 1, 2, 3$ ausgenutzt.

$$z = \left[\begin{array}{c} y_0 + y_4 \\ y_1 + y_5 \\ y_2 + y_6 \\ y_3 + y_7 \\ \hline (y_0 - y_4)\omega^0 \\ (y_1 - y_5)\omega^1 \\ (y_2 - y_6)\omega^2 \\ (y_3 - y_7)\omega^3 \end{array}\right]$$

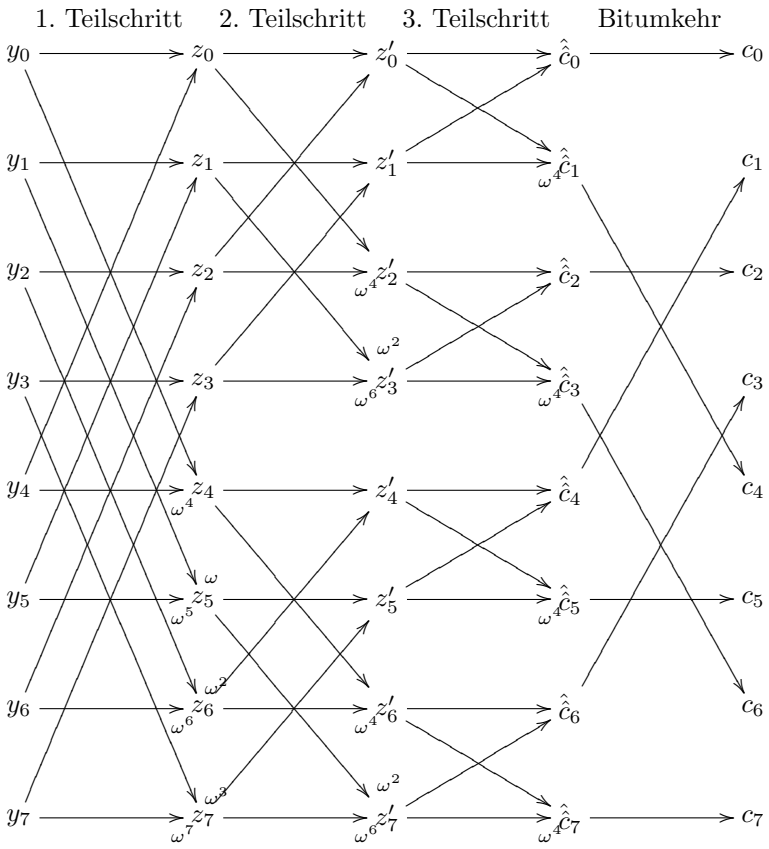
2. Teilschritt: Multiplikation von z mit der zweiten Untermatrix.

$$z' = \left[\begin{array}{c} z_0 + z_2 \\ z_1 + z_3 \\ z_0 - z_2 \\ (z_1 - z_3)\omega^2 \\ \hline z_4 + z_6 \\ z_5 + z_7 \\ z_4 - z_6 \\ (z_5 - z_7)\omega^2 \end{array}\right]$$

3. Teilschritt: Multiplikation von z' mit der ersten Untermatrix.

$$\hat{\hat{c}} = \left[\begin{array}{c} z'_0 + z'_1 \\ z'_0 - z'_1 \\ z'_2 + z'_3 \\ z'_2 - z'_3 \\ \hline z'_4 + z'_5 \\ z'_4 - z'_5 \\ z'_4 + z'_6 \\ z'_4 - z'_5 \end{array}\right]$$

Signalflußgraph mit Bitumkehr:

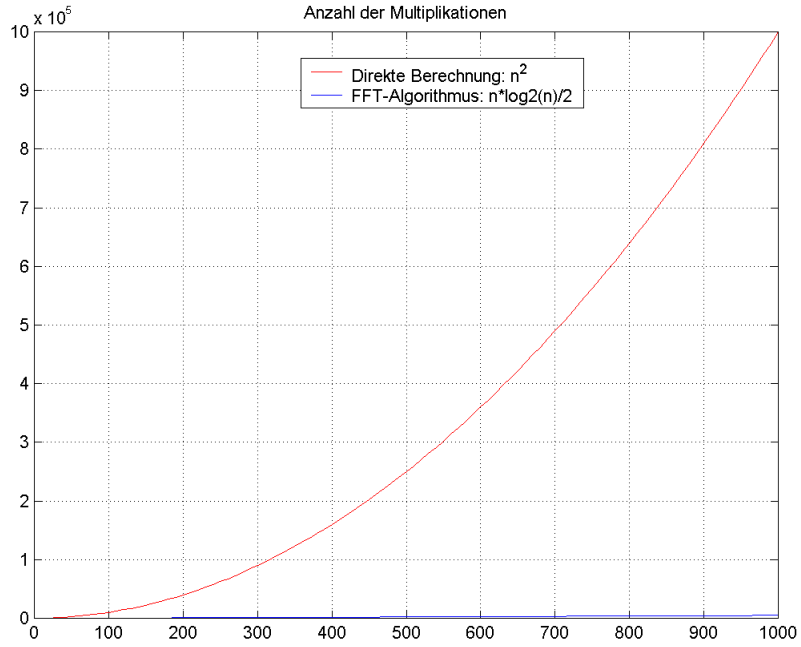


4 Rechenzeitvergleich

Interessant ist der Rechenzeitvergleich zwischen der direkten Berechnungsmethode und der schnellen Fourier-Transformation.
Für $N = 2^{\alpha}$ besteht der FFT-Algorithmus einfach aus der Faktorisierung einer N x N Matrix in p Matrizen der Größe N x N, und zwar derart, dass die Anzahl der komplexen Multiplikationen und Additionen einer jeden Teilmatrix minimal ist und zwar $\frac{Np}{2}$ komplexe Multiplikationen und Np komplexe Additionen. Geht man nun davon aus, dass die Multiplikationen das Entscheidende am Rechenaufwand sind, so ist dieser $\frac{NlogN}{2}$. Bei der direkten Berechnung sind logischer Weise N^2 komplexe Multiplikationen nötig. Und das ist doch erheblich mehr. Folgendes Zahlenbeispiel soll dies untermauern:

$$N = 1024 \Rightarrow \frac{N^2}{\frac{NlogN}{2}} = \frac{2N}{logN} = \frac{2048}{10} = 204.8$$

Der FFT-Algorithmus ist also bei N=1024 über 200mal schneller.

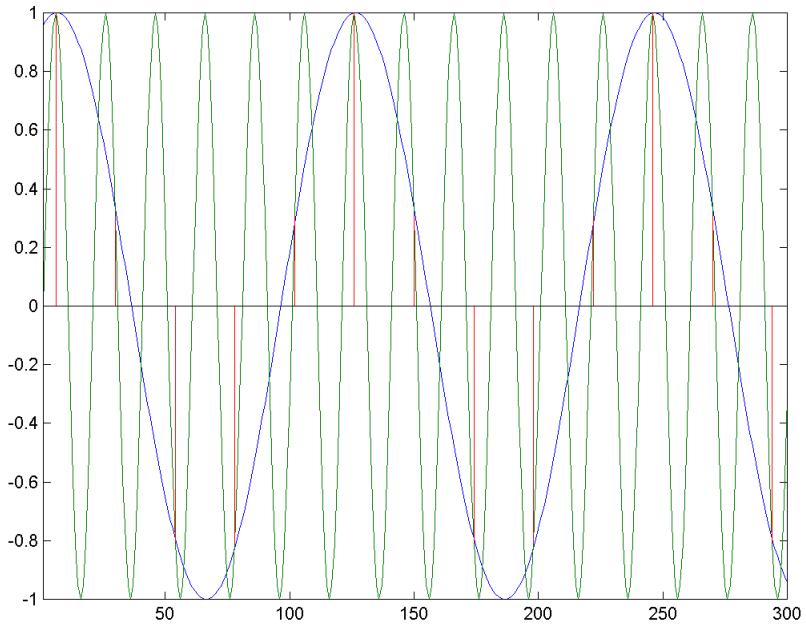


5 Grenzen der Fourier-Transformation

Im Idealfall müsste sich aus der Zeitdarstellung nach Fourier-Analyse und Synthese der ursprüngliche Datensatz vollständig wiedergewinnen lassen. In der Realität jedoch ist die Anzahl der eingehenden Datenpunkte begrenzt, und damit kann die diskrete Fourieranalyse nur eine endliche Anzahl von Frequenzen errechnen. Weiterhin gibt es eine prinzpielle Grenze, die durch das Abtast-Theorem gebildet wird: Um aus der Zeitdarstellung die Frequenzen zu errechnen, werden pro Frequenz mindestens zwei Meßwerte benötigt. Damit ist die höchste darstellbare Frequenz auf die sog. Nyquist-Frequenz f_c beschränkt:

$$f_c = \frac{1}{2t_a}$$

t_a ist dabei die numerische Differenz zwischen zwei erhobenen Zeitwerten. Enthält der Ursprungsdatensatz höhere Frequenzen als f_c, wird die Fourier-Analyse falsche Werte errechnen (*aliasing* genannt), da die außerhalb der Nyquist-Frequenz liegenden Frequenzen in den Nyquist-Bereich gespiegelt werden! Nur wenn man sicher ist, dass die Ursprungsdaten keine höheren Frequenzen als f_c enthalten, kann man also dem Ergebnis der Analyse trauen.
Folgendes Diagramm soll dies verdeutlichen:



Die grüne Frequenz wird an den roten Stellen abgetastet, aber eben nicht oft genug. Es wird die blaue Frequenz fälschlicher Weise gelesen.

Eine CD wird beispielsweise bei 44kHz gesampled, was bedeutet, dass die höchste vorkommende Frequenz bei 22kHz liegt.

6 Umsetzung in ein C++ Programm

6.1 Programmbeschreibung

Das Programm implementiert den oben erklärten FFT-Algorithmus.
Vorweg sei erwähnt, die Multiplikation mit dem Faktor ω^{α} wird immer ausgeführt, auch für $\alpha = 0$. Da bereits die erste Stufe aus solchen Elementaroperationen besteht, ergibt sich ein zusätzlicher Aufwand von N/2 Multiplikationen. Dies wurde bei der obigen Rechenzeitbetrachtung schon beachtet. Würde man dies noch abfangen, hätte man eine Komplexität von $\frac{Nlog\frac{N}{2}}{2}$ anstatt von $\frac{NlogN}{2}$.
In der Main-Routine werden die beiden Beispiele gefordert von der Aufgabenstellung nacheinander bearbeitet. Für die allgemeine Programmbetrachtung wird der Übersicht halber hier nur das erste Beispiel näher betrachtet. Das zweite Beispiel ist äquivalent. Näheres zu den Beispielen findet sich dann im nächsten Abschnitt.

Zunächst werden die komplexen Arrays a und b mit ihren Vorgabewerten gefüllt und dann daraus die komplexen Koeffizienten c berechnet:

```
for (k= 0; k<=32; k++) {
    a[k]=e*((cos(h*k)-pow(E,h))/(cosh(h)-cos(h*k)));
    b[k]=e*((-sin(h*k))/(cosh(h)-cos(h*k)));
}
// Umwandlung von a und b in c
for (k= 0; k<=32; k++) {
    c[k]=complex(a[k]*0.5,-b[k]*0.5);
}
for (k= 0; k<=30; k++) {
    c[N-(k+1)]=complex(a[k+1]*0.5,+b[k+1]*0.5);
}
```

Nach der Ausgabe ausgewählter Werte wird die FFT-Unterroutine aufgerufen:

```
fft(c,6,true);
```

Der Paramter true drückt aus, dass es sich hier um eine inverse FFT handelt.

Zum Schluss werden wieder die Werte ausgegeben.

Was passiert nun in der FFT-Unterroutine? Also als erstes werden die Bits umgekehrt. Dies kann nach, aber auch wie hier vor der eigentlichen Transformation stattfinden (c ist hier das Array mit den komplexen Koeffizienten):

```
int j=1;
...
for(i=1; i<=N-1; i++) {
    if (i<j) {
        v=c[j-1];
        c[j-1]=c[i-1];
        c[i-1]=v;
    }
    k=N/2;
    while (k<j) {
        j-=k;
        k/=2;
    }
    j+=k;
}
```

Als nächstes wird jede Stufe durchlaufen und jeweils die Fourier-Transformation ausgeführt:

```
// Stufenzaehler
for(l=1; l<=p; l++) {
    le=(int)pow(2.0,double(l));
    le1=le/2;
    wr.real=1.0;
    wr.img=0.0;
    di=PI/le1;
    // Fuer die inverse Fourier-Transformation
    // gehts natuerlich in die andere "Richtung"
    if (inv)
        di=-di;
    w.real=cos(di);
    w.img=-sin(di);
    // Diskrete Fourier Transformation
    for(r=1; r<=le1; r++) {
        for(iq=r; iq<=N; iq+=le){
            ip = iq + le1;
            v = c[ip-1]*wr;
            c[ip-1] = c[iq-1] - v;
            c[iq-1] = c[iq-1] + v;
        }
        wr=wr*w;
    }
}
```

Schließlich wird noch für die inverse Transformation der Faktor 1/N dazu multipliziert:

```
if (inv) {
    for(i=0; i<N; i++) {
        c[i]=c[i]*(1.0/N);
    }
}
```

Als Rückgabearray wird dasselbe Array verwendet wie das Übergabearray, d.h. die alten Werte sind nun überschrieben.
Die Zusatzklasse *complex* dient nur zur leichteren Handhabung der komplexen Zahlen, so lassen sich komplexe Zahlen definieren und man kann damit wie gewohnt rechnen, als wären es integer oder float Zahlen.

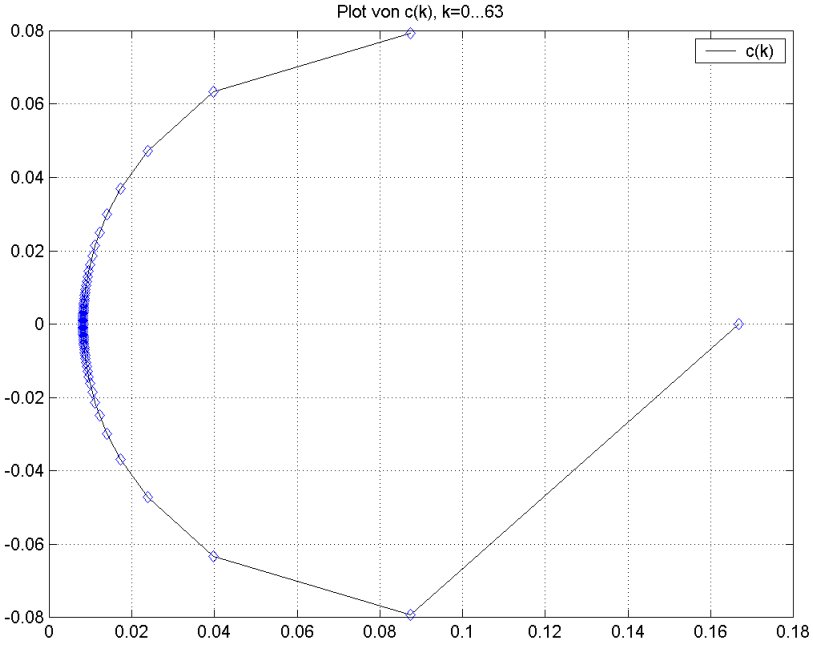
6.2 Beispiele aus der Aufgabenstellung

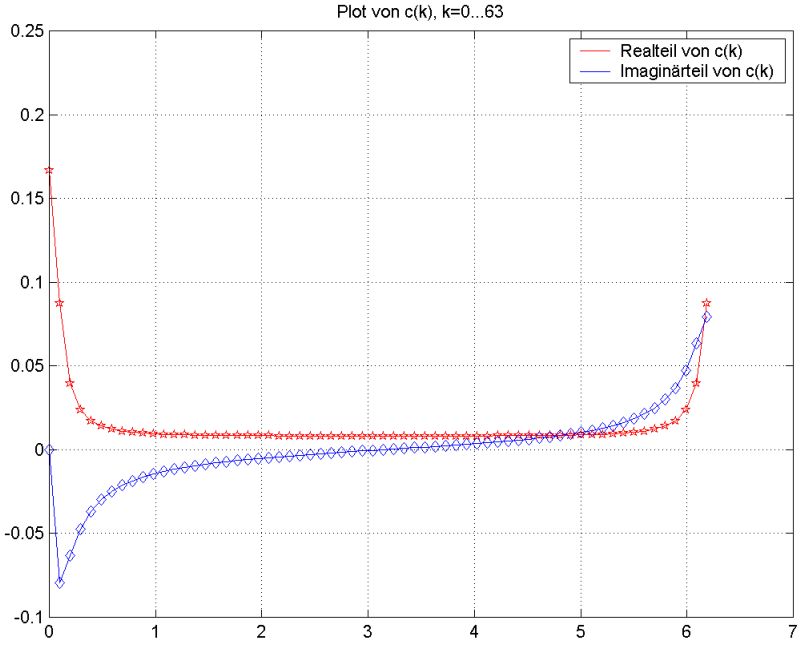
6.2.1 Beispiel 1: Synthese

$\alpha_k = \frac{e^{-2\pi}-1}{N} \frac{cos(hk)-e^h}{cosh(h)-cos(hk)}$ mit $k = 0, ..., \frac{N}{2}$
$\beta_k = \frac{e^{-2\pi}-1}{N} \frac{-sin(hk)}{cosh(h)-cos(hk)}$ mit $k = 1, ..., \frac{N}{2} - 1,$
$h = \frac{2\pi}{N}$ und $N = 64$

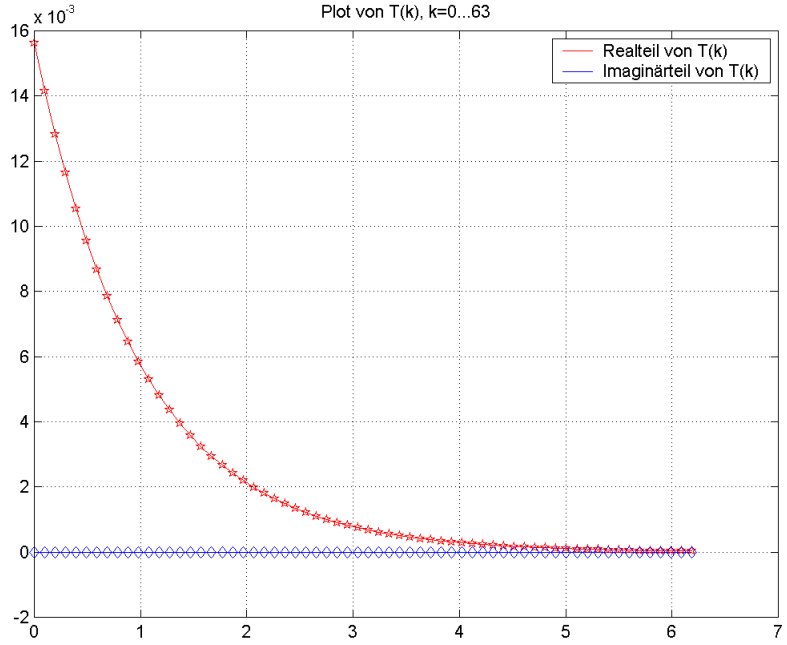
Daraus berechnet sich:
$c_k = \frac{1}{2}(\alpha_k - i\beta_k)$ mit $k = 0, ..., \frac{N}{2}$
$c_k = \frac{1}{2}(\alpha_k + i\beta_k)$ mit $k = 1, ..., \frac{N}{2} - 1$

Folgende Diagramme ergeben sich nun für c_k:





Nach der Synthese ergibt sich für T_k:



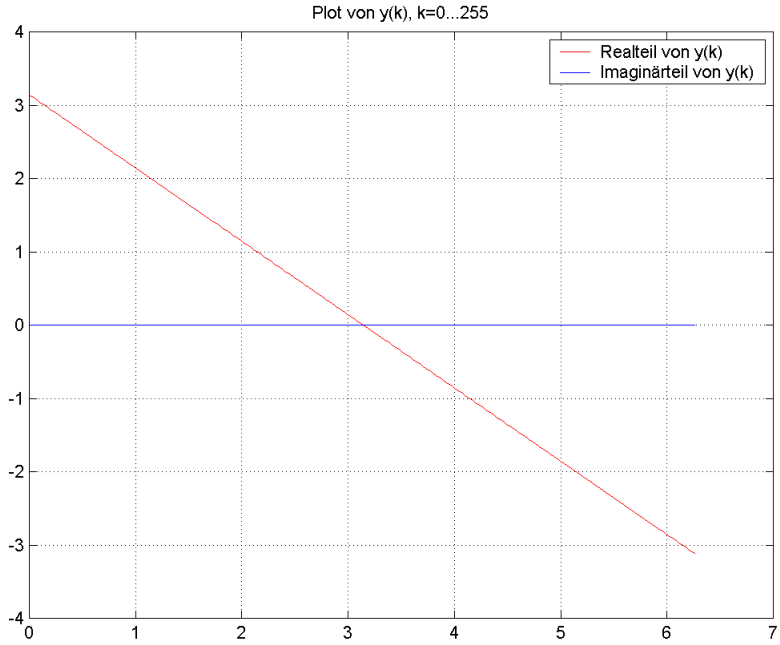
Das C++ Programm gibt folgende Werte aus:

```
Beispiel 1:
-----------
Vorher:
c(0) : 0.166783 + 0i
c(1) : 0.0873544 - 0.0793013i
c(30): 0.00818408 - 0.000766162i
c(31): 0.0081813 - 0.000382163i
c(32): 0.00818038 - 4.7632e-019i
c(62): 0.0396973 + 0.0632879i
c(63): 0.0873544 + 0.0793013i
Nacher:
T(0) : 0.015625 + 0i
T(1) : 0.0141639 + 2.24483e-019i
T(30): 0.000821707 - 6.81012e-018i
T(31): 0.00074487 - 7.45202e-018i
T(32): 0.000675217 + 0i
T(62): 3.55092e-005 + 1.22015e-018i
T(63): 3.21887e-005 + 2.35929e-018i
```

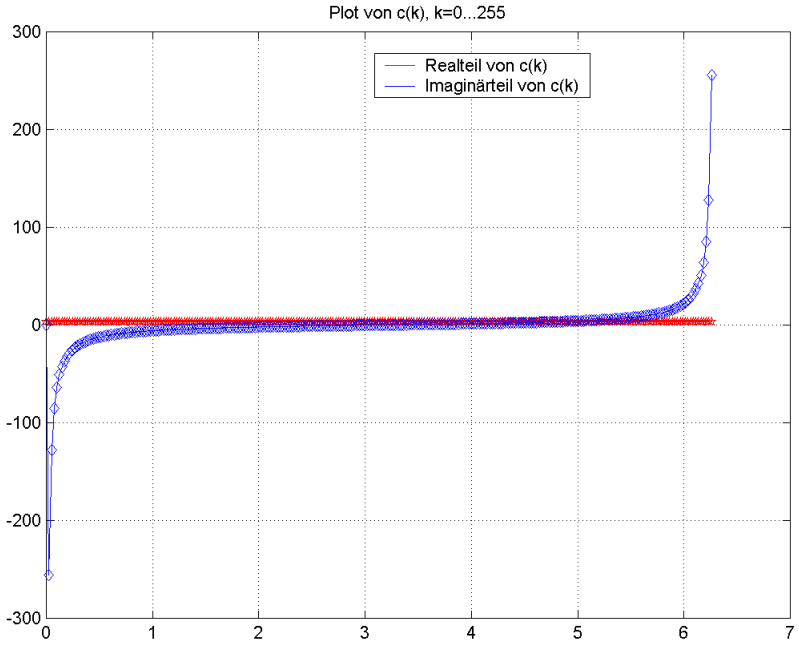
6.2.2 Beispiel 2: Analyse

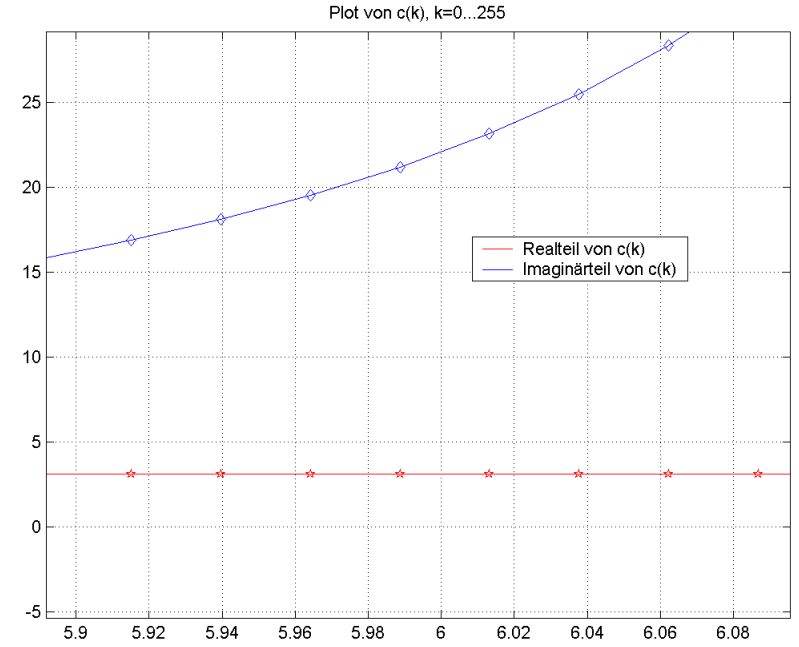
$y_j = \pi(1 - \frac{2j}{N})$
mit $j = 0, ..., N-1$ und $N = 256$

Folgendes Diagramm ergibt sich nun für y_j:

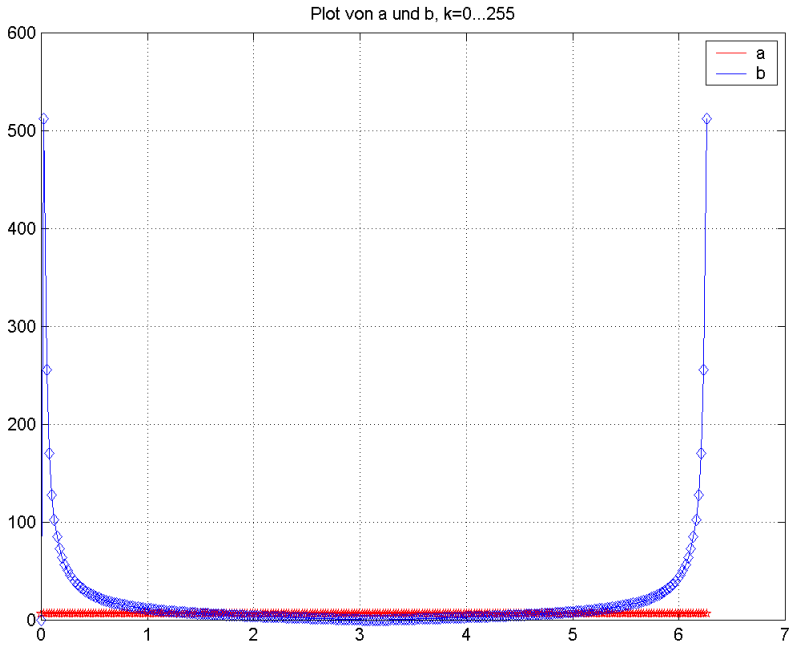


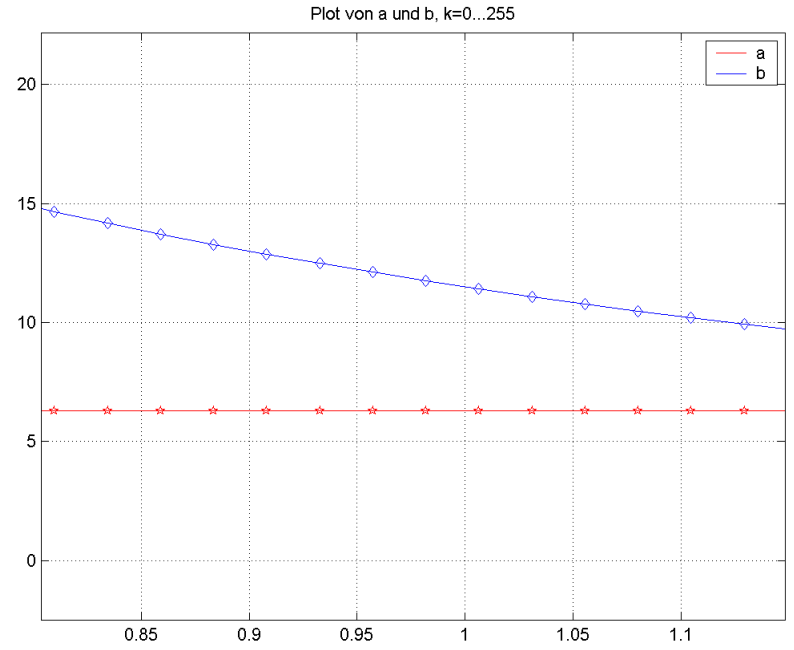
Nach der Synthese ergibt sich für c_k (das zweite Diagramm ist ein Ausschnitt des ersten):





Nach Umrechnung in α_k und β_k ergibt sich folgendes, wobei das zweite Diagramm wieder ein Ausschnitt des ersten ist:





```
Beispiel 2:
-----------
Vorher: y(0)  : 3.14159 + 0i
y(1)  : 3.11705 + 0i
y(30) : 2.40528 + 0i
y(31) : 2.38074 + 0i
y(100): 0.687223 + 0i
y(254): -3.09251 + 0i
y(255): -3.11705 + 0i
Nachher:
c(0)  : 3.14159 + 2.31133e-015i
c(1)  : 3.14159 - 255.987i
c(30) : 3.14159 - 8.14427i
c(31) : 3.14159 - 7.85578i
c(100): 3.14159 - 1.12408i
c(254): 3.14159 + 127.974i
c(255): 3.14159 + 255.987i
Daraus folgt:
a(0)  : 6.28319
a(1)  : 6.28319
a(30) : 6.28319
a(31) : 6.28319
a(100): 6.28319
a(254): 6.28319
a(255): 6.28319
b(0)  : -4.62267e-015
b(1)  : 511.974
b(30) : 16.2885
b(31) : 15.7116
b(100): 2.24816
b(254): 255.949
b(255): 511.974
```

7 Quellen und Anhang

Literatur

[1] E.O. Brigham: "FFT - Schnelle Fourier-Transformation",
Oldenburg Verlag, 1992

[2] D. Achilles: "Die Fourier-Transformation in der Signalverarbeitung",
Springer Verlag, 1985

[3] Philip J. Davis, Reuben Hersh: "Erfahrung Mathematik"
Birkhäuser, 1985

[4] http://www.scilab.de/Freque.htm: "XACT Frequenzanalysen"

Im Anhang befindet sich der komplette Source-Code des C++ Programms. D.h. das Hauptprogramm *fft.cpp* und seine Unterklasse *complex.cpp* mit ihrer Headerdatei *complex.h* die dazu dient, komfortabel mit komplexen Zahlen zu rechnen.