DFT und FFT

Anwendung, Herleitung und Implementierung in C/C++

Dr. Robert Heß

Dezember 2010

Inhaltsverzeichnis

1 Einführung

Die diskrete Fourier-Transformation (DFT) und deren spezielle Implementierung, die schnelle Fourier-Transformation (engl. *fast Fourier transform*, FFT) sind ein weit verbreitetes Instrument zur Analyse von Signalen. In diesem Schriftstück werden beispielhaft einige Anwendungen gezeigt, die elementaren Gleichungen hergeleitet und die Implementierung gezeigt.

Begleitend zu diesem Text gibt es ein paar Klangbeispiele (siehe Abschnitt 2.2) und ein Beispielprogramm zur Implementierung von DFT und FFT (siehe Abschnitt 4).

Für das Verständnis des Themas werden die Fourier-Analyse, sowie die Programmiersprache C/C++ als bekannt vorausgesetzt.

2 Anwendungen

2.1 Spektrum-Analyser

Jedes Signal kann als eine Summe von Sinus- und Kosinus-Wellen betrachtet werden. Dabei ist es häufig von Interesse, wie die Beiträge der einzelnen Frequenz-Bänder zu dem gesamten Signal sind. Die Analyse wird mit dem Spektrum-Analyser durchgeführt. Der Windows Media Player (eingetragenes Warenzeichen) verwendet spielerisch einen Spektrum-Analyser zur Visualisierung des Audio-Signals, siehe Abbildung 1. Es werden beim laufenden Audio-Strom Stücke herausgeschnitten, transformiert, und als Frequenzspektrum dargestellt. Da die Transformation in engen Zeitabständen erfolgt, erscheint ein dynamisches Spektrum auf dem Bildschirm.

Abbildung 1: Der Spektrum-Analyser im Windows Media Player (eingetr. Warenzeichen).

2.2 Equalizer

Durch geeignete Transformation wird ein Audio-Signal in seine Frequenzanteile zerlegt. Werden vor der Rück-Transformation einige Frequenzen verstärkt, gedämpft oder ganz auf null gesetzt, so beeinflusst das den Klang des Audio-Signals.

Bei den zwei beigefügten Klangbeispielen wurde wie folgt vorgegangen:

1. Transformation des Audio-Signals in den Frequenzbereich.
2. Duplizieren des Frequenzspektrums: Bei der einen Kopie wurden die hohen Frequenzen, bei der anderen die tiefen Frequenzen auf null gesetzt.
3. Rück-Transformation beider Kopien in den Zeitbereich.

Ergebnis sind zwei Klangbeispiele, das eine ausschließlich mit hohen, das andere ausschließlich mit niedrigen Frequenzen, siehe (höre) beigefügte WAV-Dateien.

Als Klangregelung ist dieses Verfahren allerdings nur bedingt geeignet: Entweder muss das Audio-Signal als ganzes bearbeitet werden (hoher Rechenaufwand), oder es entstehen bei stückweiser Bearbeitung an den Übergängen Probleme. Eine bessere Alternative ist die Faltung des Audio-Signals mit einem geeigneten Kern, der die gewünschte Eigenschaft mit sich bringt. Die DFT/FFT kann beim Design des Filters eingesetzt werden.

2.3 Entfernung von Rasterlinien

Bei der Erstellung von Röntgenbildern entstehen ungewollte Streustrahlen, die sich negativ auf die Bildqualität auswirken. Um die Streustrahlen am Bildempfänger zu reduzieren, werden zwischen dem zu untersuchenden Objekt und dem Bildempfänger dichte Bleilamellen platziert, die auf die Röntgenquelle ausgerichtet sind, das so genannte *Streustrahlenraster*. Röntgenstrahlen, die das Objekt direkt passieren, werden nur mit einer Wahrscheinlichkeit von 10 bis 20% von den Bleilamellen absorbiert, während Streustrahlen, die innerhalb des Objektes entstehen, mit einer deutlich höheren Wahrscheinlichkeit abgefangen werden. Damit sinkt der Anteil der Streustrahlen im Bild.

Nachteil dieses Verfahrens ist, das die Lamellen dazu neigen, sich im Röntgenbild abzubilden. Es werden Rasterlinien im Bild sichtbar, die nichts mit der Anatomie der durchleuchteten Objekte zu tun haben. Neben der Vermeidung dieser Linien durch sich bewegende Streustrahlenraster, können die Linien auch nachträglich durch geeignete Bildverarbeitung entfernt werden. In Abbildung 2 ist ein solches Bild mit Rasterlinien zusammen mit der zweidimensionalen FFT dargestellt.

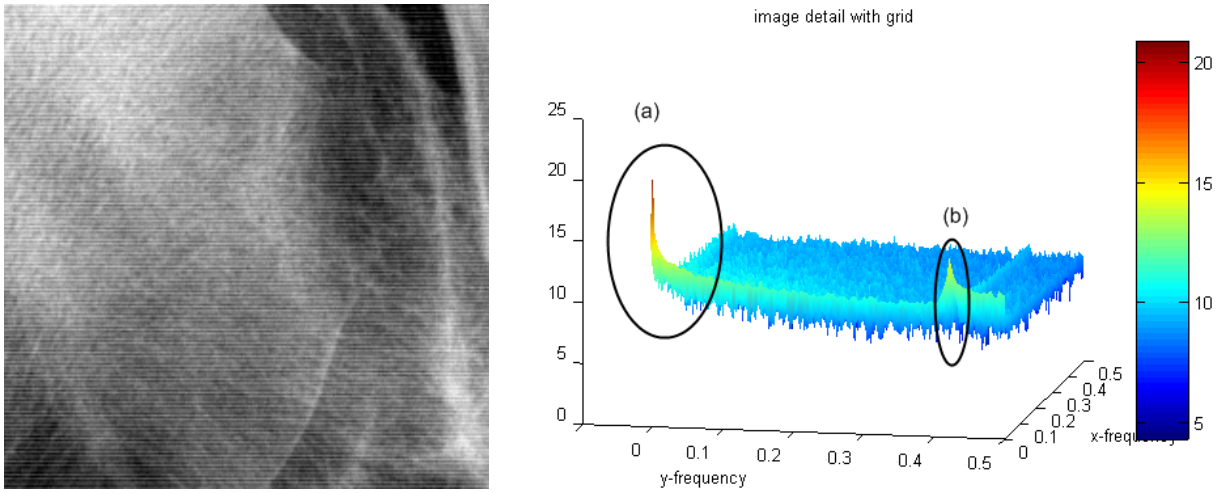

Abbildung 2: 2d FFT Analyse zum Finden der Streuraster-Linien.

Im FFT-Spektrum in Abbildung 2 sind deutlich zwei Spitzen zu erkennen, von denen eine durch die Rasterlinien erzeugt wird. Werden diese Spitzen entfernt, bzw. auf das gleiche Niveau wie die Frequenzen um die Spitzen herum reduziert, so sind die Rasterlinien im zurück-transformierten Bild nicht mehr sichtbar, siehe Abbildung 3.

2.4 Weitere Anwendungen der DFT/FFT

Die Anwendungen der DFT und FFT sind vielfältig. Einige weitere Anwendungen:

- Messkurven glätten
- Brummfilter
- Gemischte Signale separieren
- Strukturen in Bildern erkennen

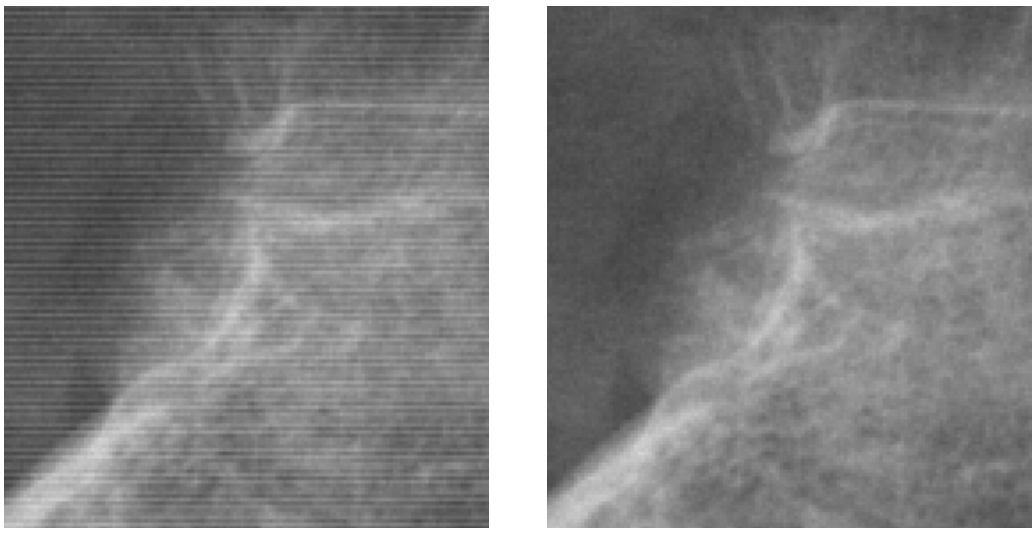

Abbildung 3: Röntgenteilbild vor und nach dem Entfernen von Rasterlinien.

- Rauschfilter in Bildern
- ...

3 Herleitung

3.1 Herleitung der DFT

Wir setzen die Gleichungen für die Fourier-Analyse als bekannt voraus: Betrachtet wird eine reale oder komplexe Funktion mit einem reellen Argument: $x(t) : \mathbb{R} \to \mathbb{C}$. Die Funktion sei integrierbar und periodisch mit Periodendauer $T \in \mathbb{R}_{>0}$. D.h. $x(t + kT) = x(t)$, $k \in \mathbb{Z}$. Die Fourier-Analyse für komplexe Koeffizienten c_n sei definiert mit:

$$c_n = \frac{1}{T} \int_{t_0}^{t_0+T} x(t) e^{-\frac{2\pi \mathrm{j}}{T} nt} dt \tag{1}$$

$$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{2\pi \mathrm{j}}{T} nt} \tag{2}$$

Die Konstante $t_0 \in \mathbb{R}$ kann beliebig gewählt werden. Es seien $x_k = x(k\frac{T}{N})$, $k \in \mathbb{Z}$ die Stützstellen der betrachteten Funktion mit $N \in \mathbb{N}$ als die Anzahl der Stützstellen pro Periode. Die abgetastete Funktion $x'(t)$ werde durch eine Serie von Dirac-Impulsen dargestellt:

$$x'(t) = \frac{T}{N} \sum_{k=-\infty}^{\infty} x_k \delta(t - kT/N) \tag{3}$$

Da wir bei der Fourier-Analyse nur eine Periode betrachten, reicht es die Dirac-Funktionen für $k = \{0 \dots N-1\}$ zu betrachten. Es folgt für die Koeffizienten c'_n der abgetasteten Funktion $x'(t)$, bei der wir mit der Fourier-Analyse (1) beginnen:

$$
\begin{aligned}
c'_n &= \frac{1}{T}\int_0^T x'(t)e^{-\frac{2\pi \mathrm{j}}{T}nt}dt \\
&= \frac{1}{T}\int_0^T \frac{T}{N}\sum_{k=0}^{N-1} x_k\delta(t-kT/N)e^{-\frac{2\pi \mathrm{j}}{T}nt}dt \\
&= \frac{1}{N}\sum_{k=0}^{N-1} x_k\int_0^T \delta(t-kT/N)e^{-\frac{2\pi \mathrm{j}}{T}nt}dt \\
&= \frac{1}{N}\sum_{k=0}^{N-1} x_k e^{-\frac{2\pi \mathrm{j}}{T}n\frac{kT}{N}} \\
c'_n &= \frac{1}{N}\sum_{k=0}^{N-1} x_k e^{-\frac{2\pi \mathrm{j}}{N}nk} \qquad (4)
\end{aligned}
$$

Die Koeffizienten c'_n sind periodisch über N, was durch die auf der Zeitachse verschobenen Dirac-Funktionen herrührt.

3.2 Herleitung der inversen DFT

Die inverse DFT wird mit der inversen Fourier-Analyse (2) hergeleitet.

$$x'(t) = \sum_{n=-\infty}^{\infty} c'_n e^{\frac{2\pi \mathrm{j}}{T}nt}$$

Da c'_n über N periodisch ist, kann die Gleichung umgeschrieben werden zu:

$$
\begin{aligned}
x'(t) &= \sum_{k=-\infty}^{\infty}\sum_{n=0}^{N-1} c'_n e^{\frac{2\pi \mathrm{j}}{T}(n+kN)t} \\
x'(t) &= \sum_{n=0}^{N-1} c'_n e^{\frac{2\pi \mathrm{j}}{T}nt}\sum_{k=-\infty}^{\infty} e^{\frac{2\pi \mathrm{j}}{T}kNt}
\end{aligned}
$$

Die zweite Summe entspricht einer inversen Fourier-Analyse mit Periode $\frac{T}{N}$ und Koeffizienten 1, was einer periodischen Folge von Dirac-Impulsen mit Periodendauer $\frac{T}{N}$ und Gewichtung $\frac{T}{N}$ entspricht:

$$
\begin{aligned}
x'(t) &= \sum_{n=0}^{N-1} c'_n e^{\frac{2\pi \mathrm{j}}{T}nt}\frac{T}{N}\sum_{k=-\infty}^{\infty}\delta(t-kT/N) \\
x'(t) &= \frac{T}{N}\sum_{k=-\infty}^{\infty}\sum_{n=0}^{N-1} c'_n e^{\frac{2\pi \mathrm{j}}{T}nt}\delta(t-kT/N)
\end{aligned}
$$

Durch Vergleich mit der Gleichung für die Abtastung (3) folgt:

$$x_k = \sum_{n=0}^{N-1} c'_n e^{\frac{2\pi \mathrm{j}}{T} nt} \tag{5}$$

Für den Faktor $\frac{1}{N}$ gibt es bei der DFT unterschiedliche Konventionen: Entweder wird er bei der Hin- oder bei der Rück-Transformation eingefügt. Eine weitere Variation der Mathematiker verteilt die Wurzel des Faktors auf beide Richtungen. Wir verwenden die in der Signalverarbeitung übliche Notation und fügen den Faktor $\frac{1}{N}$ bei der Rück-Transformation ein.

Bei allen Fourier-Transformationen, so auch bei der DFT, ist der Exponent bei der Hin- und Rück-Transformation einmal positiv und einmal negativ. Bei der Signalverarbeitung ist es üblich den negativen Exponenten bei der Hin-Transformation zu verwenden, so dass wir für die Vorzeichen bei der bisherigen Notation bleiben. Es ergeben sich folgende Gleichungen für die DFT:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} \tag{6}$$

$$x_k = \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{\frac{2\pi \mathrm{j}}{N} kn} \tag{7}$$

3.3 Herleitung der FFT

Der Ansatz der FFT ist, die Transformation für N Elemente auf zwei Transformationen mit jeweils $\frac{N}{2}$ Elementen aufzuteilen. Wird diese Aufteilung geschickt vollzogen, so können diverse Rechenoperationen eingespart werden.

Zunächst teilen wir die Summenbildung der DFT (6) in zwei Hälften:

$$\begin{aligned}
X_n &= \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} \\
&= \sum_{k=0}^{N/2-1} x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} + \sum_{k=N/2}^{N-1} x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} \\
&= \sum_{k=0}^{N/2-1} x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} + \sum_{k=0}^{N/2-1} x_{k+\frac{N}{2}} e^{-\frac{2\pi \mathrm{j}}{N} n(k+\frac{N}{2})} \\
&= \sum_{k=0}^{N/2-1} \left\{ x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} + x_{k+\frac{N}{2}} e^{-\frac{2\pi \mathrm{j}}{N} n(k+\frac{N}{2})} \right\} \\
&= \sum_{k=0}^{N/2-1} \left\{ x_k e^{-\frac{2\pi \mathrm{j}}{N} nk} + x_{k+\frac{N}{2}} e^{-\frac{2\pi \mathrm{j}}{N} nk} e^{-\frac{2\pi \mathrm{j}}{2} n} \right\}
\end{aligned}$$

Jetzt unterscheiden wir zwischen geraden und ungeraden Elementen:

$$X_{n_{even}} = X_{2n'} = \sum_{k=0}^{N/2-1} \left\{ x_k e^{-\frac{2\pi \mathrm{j}}{N} 2n'k} + x_{k+\frac{N}{2}} e^{-\frac{2\pi \mathrm{j}}{N} 2n'k} \underbrace{e^{-\frac{2\pi \mathrm{j}}{2} 2n'}}_{=1} \right\}$$
$$= \sum_{k=0}^{N/2-1} (x_k + x_{k+\frac{N}{2}}) e^{-\frac{2\pi \mathrm{j}}{N/2} n'k} \tag{8}$$

$$X_{n_{odd}} = X_{2n'+1} = \sum_{k=0}^{N/2-1} \left\{ x_k e^{-\frac{2\pi \mathrm{j}}{N}(2n'+1)k} + x_{k+\frac{N}{2}} e^{-\frac{2\pi \mathrm{j}}{N}(2n'+1)k} \underbrace{e^{-\frac{2\pi \mathrm{j}}{2}(2n'+1)}}_{=-1} \right\}$$
$$= \sum_{k=0}^{N/2-1} (x_k - x_{k+\frac{N}{2}}) e^{-\frac{2\pi \mathrm{j}}{N}(2n'+1)k}$$
$$= \sum_{k=0}^{N/2-1} (x_k - x_{k+\frac{N}{2}}) e^{-\frac{2\pi \mathrm{j}}{N/2} n'k} e^{-\frac{2\pi \mathrm{j}}{N} k} \tag{9}$$

In beiden Fällen, für die geraden und die ungeraden Elemente von X_n, steht am Ende ein Ausdruck, der einer DFT mit $\frac{N}{2}$ Elementen entspricht. Für die geraden Elemente werden die Koeffizienten durch eine Summe von zwei der ursprünglichen Koeffizienten ermittelt. Für die ungeraden Elemente ergeben sich die Koeffizienten durch die Differenz von zwei der ursprünglichen Koeffizienten multipliziert mit einem komplexen Faktor. Die Abbildungen 4 und 5 zeigen diesen Vorgang für eine FFT mit acht und vier Elementen.

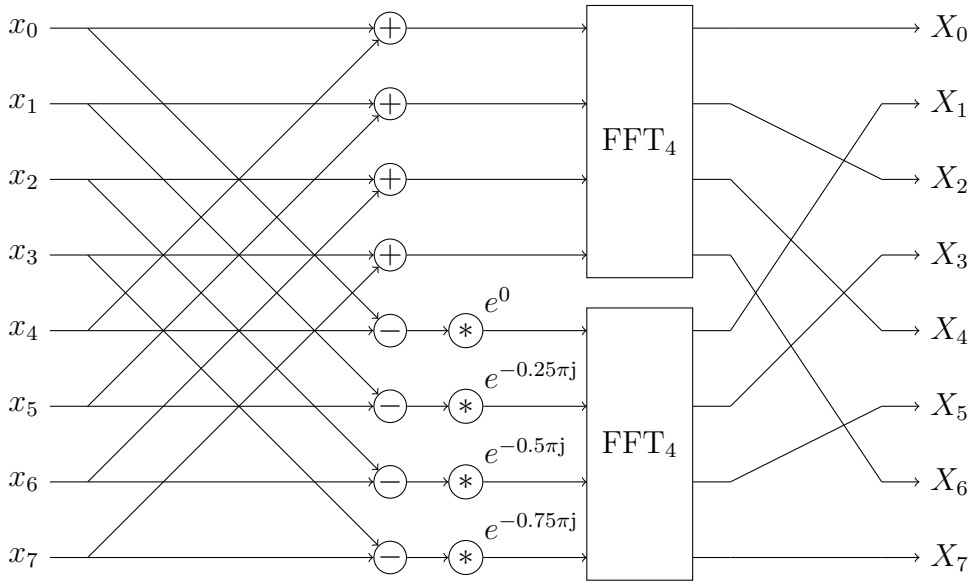

Abbildung 4: FFT für acht Elemente.

Das Prinzip der FFT ist nun, dass die Anzahl der Elemente für die Transformation immer wieder halbiert wird, bis am Ende nur noch ein Element übrig bleibt. Die DFT einer einzigen Zahl ist die Zahl selbst und kann daher ohne Rechnung übernommen werden.

In diesem Abschnitt wird nur auf die *Hin*-Transformation eingegangen. Die *Rück*-Transformation erfolgt nach dem gleichen Schema, nur das einmal ein Vorzeichen umgekippt, und der Faktor $\frac{1}{N}$ eingefügt werden muss.

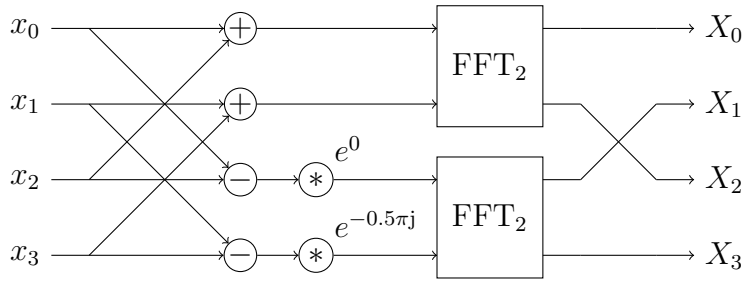

Abbildung 5: FFT für vier Elemente.

3.4 Sortieren der Elemente

Die oben beschriebene FFT hat noch ein Problem: Die Elemente sind falsch sortiert. Aus Abbildung 4 und 5 wird deutlich, wie nach der Transformation die geraden und ungeraden Elemente wieder verzahnt werden müssen. Um unnötig häufiges Sortieren zu vermeiden, bietet es sich an, erst am Ende alle Elemente an ihren Platz zu verschieben.

Zum Sortieren wird das sogenannte *Bit-Umkehrverfahren* benötigt: Um herauszufinden an welche Stelle ein Element mit Index n gehört, müssen die binären Ziffern des Index in der Reihenfolge gespiegelt werden. Für acht Elemente heißt das, es müssen die Elemente 1 und 4, sowie die Elemente 3 und 6 vertauscht werden, siehe Tabelle 1.

alter Index		→	neuer Index	
dezimal	binär		binär	dezimal
0	000	→	000	0
1	001	→	100	4
2	010	→	010	2
3	011	→	110	6
4	100	→	001	1
5	101	→	101	5
6	110	→	011	3
7	111	→	111	7

Tabelle 1: Bitumkehrverfahren für acht Elemente.

Was bei acht Elementen noch recht einfach erscheint, wird bei mehr Elementen zunehmend kompliziert, siehe Tabelle 2. Bei einer großen Anzahl von Elementen müssen fast alle Elemente paarweise vertauscht werden. Für den Mechanismus des Vertauschens gibt es unterschiedliche Algorithmen, auf die hier nicht weiter eingegangen wird.

Anzahl der Elemente	4	8	16	256	2^{16}	2^{32}
Anzahl der Vertauschungen	1	2	6	120	32.640	2.147.450.880

Tabelle 2: Anzahl der Vertauschungen für verschiedene Anzahlen an Elementen.

3.5 Rechenaufwand für DFT und FFT

Gemäß Gleichung (6) für die DFT werden für N Elemente der Bildfunktion jeweils N Multiplikationen und $N-1$ Additionen benötigt. Es ergibt sich ein Aufwand, der mit N^2 wächst (...die -1 haben wir unter den Tisch fallen lassen):

$$\text{Aufwand der DFT} \in \mathcal{O}\left(N^2\right)$$

Der Aufwand für die FFT sind $\frac{N}{2}$ Additionen, $\frac{N}{2}$ Subtraktionen, $\frac{N}{2}$ Multiplikationen der Differenzen, $\frac{N}{2}$ Multiplikationen beim Erstellen der Faktoren und zweimal der Aufwand der FFT mit $\frac{N}{2}$ Elementen. Am Ende sind nochmal einige Operation für das Sortieren der Elemente erforderlich. Nach Auflösen der Rekursion ergibt sich ein Aufwand:

$$\text{Aufwand der FFT} \in \mathcal{O}\left(N \log N\right)$$

Je größer die Anzahl der Elemente, desto signifikanter der zeitliche Vorteil der FFT gegenüber der DFT. Angenommen, die Rechenzeiten für DFT und FFT sind bei 16 Elementen gleich, so ist die FFT bei 256 Elementen bereits um den Faktor 8 schneller, bei 2^{16} Elementen ist die FFT 1000 mal so schnell und bei 2^{32} Elementen braucht die FFT nur ein 33-Millionstel der Zeit einer DFT.

4 Implementierung in C/C++

In diesem Abschnitt sollen die DFT und FFT unter C/C++ implementiert werden. Es wurde die Programmierumgebung wxDev-C++ verwendet, siehe wxdsgn.sourceforge.net. Der beigefügte Quelltext verwendet komplexe Zahlen, die durch die Header-Datei *complex* eingebunden werden (bei anderen Compilern ggf. mit Endung .h: *complex.h*). Ansonsten wurde auf die Objekt orientierte Programmierung (OOP) verzichtet und die strukturierte Programmierung angewandt.

Die gezeigten Programmstücke sind lauffähig, aber nicht auf minimale Laufzeit optimiert. Es wurde vielmehr der Fokus auf Verständlichkeit gelegt. In der Literatur finden sich diverse Variationen die je nach Anwendung spezielle Vorteile bieten.

Es wird jeweils nur die *Hin*-Transformation gezeigt, dann aber auf die Besonderheiten der *Rück*-Transformation hingewiesen.

Für die folgenden Quelltexte wurde die Headerdatei `<complex>` eingebunden, der Namensraum `std` gewählt und die Konstante `pi` definiert:

```
#include <complex>
using namespace std;
const double pi=3.14159265358979;
```

4.1 Implementierung der DFT

Soll für N Elemente eine DFT erstellt werden, so müssen für N Frequenzen die Koeffizienten X_n bestimmt werden, was zugleich die äußere Schleife ergibt. Für jeden Koeffizient X_n müssen alle Stützstellen x_k der Original-Funktion mit einem Faktor multipliziert und dann addiert werden, was die zweite, innere Schleife ergibt.

Die folgende Funktion erwartet einen Zeiger auf einen Vektor komplexer Zahlen mit den Stützstellen der Original-Funktion, sowie die Anzahl der Stützstellen. Die Funktion reserviert zunächst Speicher für die transformierten Koeffizienten und initialisiert sie mit null. In der dann folgenden äußeren Schleife werden dann alle transformierten Koeffizienten der reihe nach bestimmt.

Innerhalb der äußeren Schleife wird der Faktor `wActual` auf eins gesetzt und die Schrittweite `wStep` für die Exponentialfunktion ermittelt. In der inneren Schleife wird jede Stützstelle x_k mit dem Faktor `wActual` multipliziert und im Koeffizienten X_n aufaddiert. Danach wird in der inneren Schleife der veränderte Faktor `wActual` bestimmt.

Nach dem Durchlauf der Schleifen werden die berechneten Koeffizienten X_n in das übergebene Datenfeld kopiert, und der Speicher wird wieder freigegeben.

```
void DFT(complex<double> *data, unsigned N)
{
    bool error=false;           // error flag
    complex<double> *tmp=NULL;  // pointer to temporary data
    int k, n;                   // local index variables
    complex<double> wActual;    // actual rotation factor
    complex<double> wStep;      // step between rotation factors

    // get memory for temporary data
    error = error || (tmp=new complex<double>[N])==NULL;

    // initialise temporary data
    for(n=0; !error && n<N; n++) tmp[n] = complex<double>(0, 0);

    // loop over all target values
    for(n=0; !error && n<N; n++) {

        // actual angle and angle step
        wActual = complex<double>(1, 0);
        wStep = complex<double>(cos(-2*pi*n/N), sin(-2*pi*n/N));

        // loop over all source values
        for(k=0; k<N; k++) {
            tmp[n] += data[k]*wActual;
            wActual *= wStep;
        }
    }

    // copy data
    for(n=0; !error && n<N; n++) data[n] = tmp[n];

    // free memory
    if(tmp) delete[]tmp;
}
```

Die inverse DFT läuft nach dem gleichen Schema mit nur zwei Unterschieden:

1. Das Vorzeichen in der Sinus- und Kosinus-Funktion muss positiv sein.

2. Nach der Berechnung müssen alle berechneten Koeffizienten durch N dividiert werden.

4.2 Implementierung der FFT

Bei der FFT findet immer wieder eine Rechnung statt, die aus zwei komplexen Zahlen zwei neue komplexe Zahlen erzeugt. Für die erste neue Zahl wird die Summe der alten Zahlen berechnet. Für die zweite neue Zahl wird die Differenz der alten Zahlen ermittelt, und die Differenz mit einem Faktor multipliziert. Es wird von der *Schmetterlingsrechnung* (engl. *butterfly*) gesprochen. Der Schmetterling ist immer halb so groß wie die aktuelle FFT, siehe auch Abbildung 4 und 5.

Die FFT hat gegenüber der DFT den weiteren Vorteil, dass sie ohne temporären Speicher für die Koeffizienten auskommt. Daher fällt gegenüber der DFT die Reservierung und Freigabe von Speicher weg.

Die FFT wurde mit drei verschachtelten Schleifen implementiert:

- Die erste, äußere Schleife durchläuft die Rekursionsebenen der FFT, bzw. die Größe der *Schmetterlinge*.

- Die zweite, mittlere Schleife durchläuft alle Schmetterlingsrechnungen der aktuellen FFT.

- Die dritte, innere Schleife durchläuft alle FFT einer Rekursionsebene.

Intuitiv wäre man geneigt, die zweite und dritte Schleife zu vertauschen. Durch die hier gezeigte Reihenfolge können aber die Faktoren für die Differenzen mehrfach verwendet werden, was erheblich Rechenzeit einspart.

Innerhalb der dritten, inneren Schleife findet die Schmetterlingsrechnung statt, für die eine temporäre Variable benötigt wird.

Nach den dreifach verschachtelten Schleife sind die transformierten Koeffizienten noch nicht an der richtigen Stelle und müssen paarweise vertauscht werden. Dies geschieht im letzten Block der Funktion, auf den hier nicht weiter eingegangen wird.

```
void FFT(complex<double> *data, unsigned N)
{
    unsigned butterflySize;   // size for actual butterfly calculation
    int i, j, k;              // local index variables
    complex<double> wActual;  // actual rotation factor
    complex<double> wStep;    // step rotation factors
    complex<double> tmp;      // temp. value for butterfly calculation

    // loop over all level of FFT
    for(butterflySize=N/2; butterflySize>0; butterflySize/=2) {
        // evaluate angle step and set first angle
        wStep = complex<double>(cos(-pi/butterflySize),
                                sin(-pi/butterflySize));
```

```
            wActual = complex<double>(1, 0);
            // loop over number of butterflys
            for(j=0; j<butterflySize; j++) {
                // loop over number of FFTs
                for(i=j; i<N; i+=2*butterflySize) {
                    // get index of second element
                    k = i+butterflySize;
                    // perform butterfly calculation
                    tmp = data[i];          // store one element
                    data[i] += data[k];     // take sum
                    data[k] = tmp-data[k];  // take difference
                    data[k] *= wActual;     // multiply with rotation factor
                }
                // evaluate next rotation factor
                wActual *= wStep;
            }
        }

        // perform bit reversal
        j = 0;
        for(i=0; i<N; i++) {
            if(j>i) {
                // swap numbers
                tmp = data[i];
                data[i] = data[j];
                data[j] = tmp;
            }
            k = N/2;
            while(k>=2 && j>=k) {
                j -= k;
                k /= 2;
            }
            j += k;
        }
    }
```

Die inverse FFT läuft wieder nach dem gleichen Schema mit zwei Unterschieden:

1. Das Vorzeichen in der Sinus- und Kosinus-Funktion muss positiv sein.
2. Nach der Berechnung müssen alle berechneten Koeffizienten durch N dividiert werden.