Methodische und Praktische Grundlagen der Informatik 4

Übungsblatt 11

mpgi4@cg.tu-berlin.de

WiSe 2014/2015

1. Rekursive diskrete Fouriertransformation

Wir kennen bereits die diskrete Fouriertransformation. Ist $z \in \mathbb{C}^N$ ein Signal der Länge N und bezeichne $\omega_N = e^{-2\pi i/N}$, dann ist diese durch

$$\hat{z}(m) = \sum_{n=0}^{N-1} z(n)\, \omega_N^{mn} \tag{1}$$

gegeben. Ist $N = 2M$ gerade, so können wir die Summe aufteilen in eine Summe über die geraden und eine Summe über die ungeraden Indizes:

$$\hat{z}(m) = \sum_{n=0,2,4,\ldots}^{N-1} z(n)\, \omega_N^{mn} + \sum_{n=1,3,5,\ldots}^{N-1} z(n)\, \omega_N^{mn}. \tag{2a}$$

Führen wir nun eine neue Indizierung der zwei Summen ein, so dass die erste Summe von 0 bis $M = N/2$ läuft, mit nun $2n$ im Exponenten der komplexen Exponentialfunktion, und analog für die zweite Summe, so erhalten wir,

$$\hat{z}(m) = \sum_{n=0}^{M-1} z(2n)\, \omega_N^{2nm} + \sum_{n=0}^{M-1} z(2n+1)\, \omega_N^{(2n+1)m}. \tag{2b}$$

In der zweiten Summe können wir $\omega_N^{(2n+1)m}$ auch als $\omega_N^{(2n+1)m} = \omega_N^{2nm+m} = \omega_N^{2nm}\, \omega_N^m$ schreiben, so dass

$$\hat{z}(m) = \sum_{n=0}^{M-1} z(2n)\, \omega_N^{2nm} + \omega_N^m \sum_{n=0}^{M-1} z(2n+1)\, \omega_N^{2mn} \tag{2c}$$

$$= \sum_{n=0}^{M-1} z(2n)\, \omega_M^{mn} + \omega_N^m \sum_{n=0}^{M-1} z(2n+1)\, \omega_M^{mn}\,, \tag{2d}$$

wobei in der letzten Zeile ausgenutzt wurde, dass

$$\omega_N^{2mn} = e^{2\pi i 2mn/N} = e^{2\pi imn/M} = \omega_M^{mn} \tag{3}$$

gilt. Mit Gleichung (2d) haben wir eine diskrete Fouriertransformation der Länge N in zwei Transformationen der Länge $M = N/2$ zerlegt. Diese ist, wie wir in wenigen Momenten sehen werden, entscheidend für die bessere Laufzeit der schnellen Fouriertransformation. Allerdings hat es auch zur Folge, dass wir nur Frequenzen bis $M = N/2$ durch Gleichung (2d) erhalten. Für die hochfrequenten Signalanteile müssen wir auch $m > M$ betrachten.

Durch die Periodizität der komplexen Exponentialfunktion (oder, äquivalent, mittels der Eulerschen Formel und der Periodizität der Sinus- und Kosinusfunktion) gilt

$$\omega_M^{-Mn} = 1 = e^{2\pi iMn/M} = e^{2\pi in} = 1 \tag{4}$$

für alle n. Damit haben wir

$$\omega_M^{mn} = \omega_M^{mn}\, 1 = \omega_M^{mn}\, \omega_M^{-Mn} = \omega_M^{(m-M)n}. \tag{5}$$

Außerdem erhält man in analoger Weise

$$\omega_N^m = \omega_N^{m-M} = -\omega^{m-M}. \tag{6}$$

Für $m \geq M$ können wir (2d) somit schreiben als:

$$\hat{z}(m) = \sum_{n=0}^{M-1} z(2n)\,\omega_M^{(m-M)n} - \omega_N^{m-M} \sum_{n=0}^{M-1} z(2n+1)\,\omega_M^{(m-M)n} \qquad (m \geq M). \tag{7}$$

Bezeichnen wir die geraden bzw. ungeraden Koeffizienten von z mit

$$z_1(n) = z(2n) \quad \text{und} \quad z_2(n) = z(2n+1)\,, \tag{8}$$

so ergibt sich zusammengefasst:

$$\hat{z}(m) = \begin{cases} \hat{z}_1\big(m\big) + \omega_N^m\,\hat{z}_2\big(m\big) & m < M \\ \hat{z}_1\big(m-M\big) - \omega_N^{m-M}\,\hat{z}_2\big(m-M\big) & m \geq M \end{cases} \tag{9}$$

Gleichung (9) ist der Schlüssel zur schnellen Berechnung der diskreten Fouriertransformation: Anstatt eine Transformation der Länge N zu berechnen, müssen nur zwei Transformationen der halben Länge $M = N/2$ bestimmt werden, welche jeweils die geraden und ungeraden Elementen in die Fourierdarstellung abbilden. Jede der Transformationen der Länge $N/2$ kann dabei erneut in zwei Transformationen der halben Länge $N/4$ zerlegt werden. Dieser Prozess wird fortgesetzt, bis nur triviale Fouriertransformationen der Länge 2 bestimmt werden müssen, welche anschließend mit Gleichung (9) wieder zusammen geführt werden, um die Rekursion aufzulösen. Damit erhalten wir einen Teile und Herrsche Algorithmus zur Bestimmung der diskrete Fouriertransformation. Der Algorithmus erzeugt einen Binärbaum, dessen Wurzel die diskrete Fouriertransformation der ursprünglichen Eingangsdaten ist, und für welchen der linke Kindknoten jedes Knotens die Fouriertransformation der geraden Elemente und der rechte Kindknoten die Transformation der ungeraden Element darstellt.[1] Bei Konstruktion enthält der Baum $\log_2(N)$ Ebenen und wir berechnen auf Ebene l jeweils N/k mal Gleichung (9) mit Aufwand $O(k)$, wobei $k = 2^l$ ist. Damit erhalten wir für die schnelle Fouriertransformation eine Komplexität von

$$O\big(k(N/k)\log_2(N)\big) = O\big(N\log_2(N)\big)\,. \tag{10}$$

Für große N ist dies eine entscheidende Einsparung im Vergleich zum Aufwand von $O(N^2)$, welcher für die klassische diskrete Fouriertransformation notwendig ist.

Analog zur Vorwärtstransformation kann auch die inverse diskrete Fouriertransformation rekursiv mit einem Teile und Herrsche Algorithmus bestimmt werden. Dabei muss in Gleichung (1) der Term ω^{mn} durch $\omega^{-mn} = \bar{\omega}^{mn}$ ersetzt werden. Alternativ kann die inverse Fouriertransformation auch mittels der Vorwärtstransformation ausgedrückt werden:

$$\check{z}(n) = \frac{1}{N}\sum_{m=0}^{N-1} z(m)e^{2\pi imn/N} = \frac{1}{N}\overline{\sum_{m=0}^{N-1} \bar{z}(m)e^{-2\pi imn/N}} = \frac{1}{N}\overline{\hat{\bar{z}}(m)} \tag{11}$$

wobei $\bar{z}(m)$ die komplex-konjugierten Fourierkoeffizienten sind, und $\overline{\hat{\bar{z}}(m)}$ die komplexe Konjugation der Fouriertransformation von $\bar{z}(m)$.

2. Schnelle diskrete Fouriertransformation (FFT)

Die Rekursionsformel (9) führt bereits zu einem verbesserten Algorithmus für die diskrete Fouriertransformation. Für eine optimale Implementierung ist es wünschenswert, die in den meisten Programmiersprachen substantiellen Kosten für einen rekursiven Funktionsaufruf zu vermeiden. Im Folgenden wollen wir daher die Rekursion auflösen. Der sich dadurch ergebene Algorithmus wird als *schnelle diskrete Fouriertransformation* (*engl.* fast Fourier transform (FFT)) bezeichnet.

Um zu verstehen, welche Berechnungen durch die rekursive Zerlegung der Fouriertransformation entstehen, wollen wir den Ablauf für ein Signal der Länge $N = 8$ betrachten. Als erste Beobachtung können wir festhalten, dass die Bearbeitung in zwei Phasen aufgeteilt werden kann. In der ersten Phase (*Sortierphase*) wird das Signal nach geraden und ungeraden Indizes aufgeteilt:

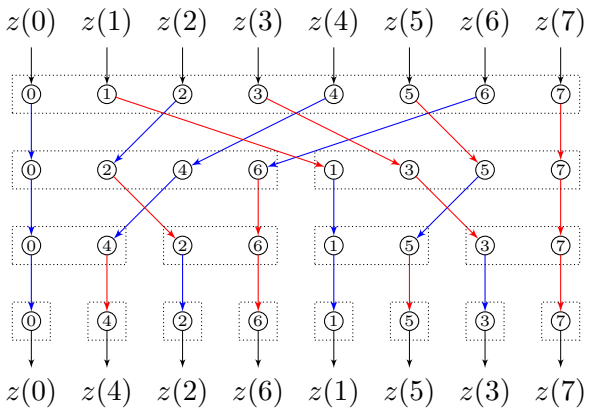
[1] Wir betrachten hier nur den Fall das $N = 2^a$ mit a eine ganze, positive Zahl. In der Literatur werden auch die Veränderungen diskutiert, welche für beliebige N notwendig sind.

```python
import numpy as np
from numpy import pi
import cmath

def dft_recursive(c):
    N = len(c)
    if N == 1:
        return c
    else:
        M = N // 2
        z1 = dft_recursive(c[0::2])
        z2 = dft_recursive(c[1::2])
        omega = cmath.exp(-2j * pi / N)
        z = np.empty((N,), dtype=complex)
        for m in range(M):
            z[m] = z1[m] + (omega ** m) * z2[m]
            z[m+M] = z1[m] - (omega ** m) * z2[m]
        return z

def idft_recursive(c):
    N = len(c)
    return 1.0/N * np.conjugate(dft_recursive(np.conjugate(c)))
```

Abbildung 1: *Rekursive Implementierung der diskreten Fouriertransformation für den Fall* $N = 2^n$.



Diese Phase entspricht einer Umsortierung des Signals. Stellt man die Indizes des Signals in Binärdarstellung in einer Tabelle gegenüber, so sieht man, dass sich die neuen Indizes durch Umkehrung der Bits ergeben:

$$\begin{array}{lllllll} 0 & = & 000 & \to & 000 & = & 0 \\ 1 & = & 001 & \to & 100 & = & 4 \\ 2 & = & 010 & \to & 010 & = & 2 \\ 3 & = & 011 & \to & 110 & = & 6 \\ 4 & = & 100 & \to & 001 & = & 1 \\ 5 & = & 101 & \to & 101 & = & 5 \\ 6 & = & 110 & \to & 011 & = & 3 \\ 7 & = & 111 & \to & 111 & = & 7 \end{array}$$

Dies entspricht einer Umwandlung der Bitwertigkeit von LSB-0 (*least significant bit*) in MSB-0 (*most significant bit*):

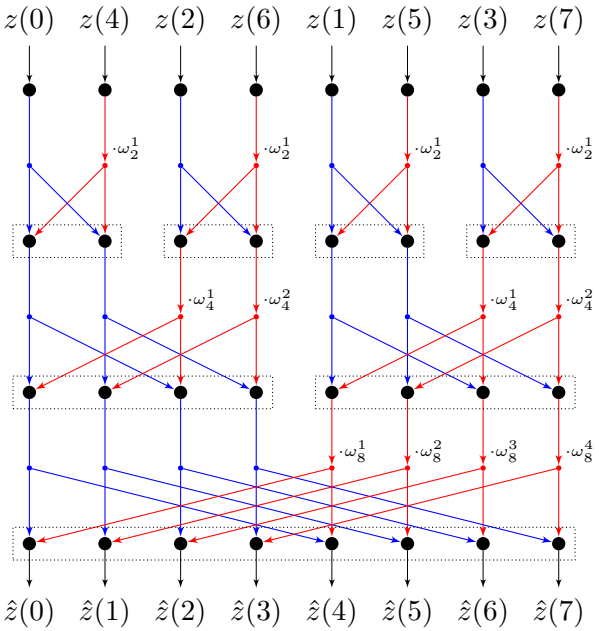
$$i = \sum_{k=0}^{p-1} a_k 2^k \mapsto \sum_{k=0}^{p-1} a_k 2^{p-1-k} = j \tag{12}$$

Mittels dieser Formel kann man für $i = 0, \ldots, 2^p - 1$ neue Indizes j bestimmen, und das Signal entsprechend umordnen.

Nebenbemerkung. *Ein effizientes Umordnen der Signalelemente nach Gleichung (12) ist wie folgt möglich. Schaut man sich obige Tabelle genauer an, so fällt zunächst auf, dass die Umordnung immer paarweise erfolgt. Es reicht somit aus, sich auf Vertauschungen mit* $i < j$ *zu beschränken. Bei der Berechnung von* i *und* j *kann außerdem induktiv vorgegangen werden. Begonnen wird mit dem Paar*

$(i, j) = (0, 0)$. *In jedem Schritt wird nun i inkrementiert. Der zugehörige Index j muss auch entsprechend erhöht werden, allerdings bzgl. MSB Bitwertigkeit. Dazu werden die Bits von j vom höchsten bis zum niedrigsten Bit abgearbeitet. Gesetzte Bits werden dabei gelöscht. Wird ein nicht gesetztes Bit erreicht, so wird dieses gesetzt und der Vorgang abgeschlossen.*

Nach der Umordnung des Signal folgt die zweite Phase des Algorithmus (*Kombinationsphase*), in welcher die Fourierkoeffizienten bestimmt werden:

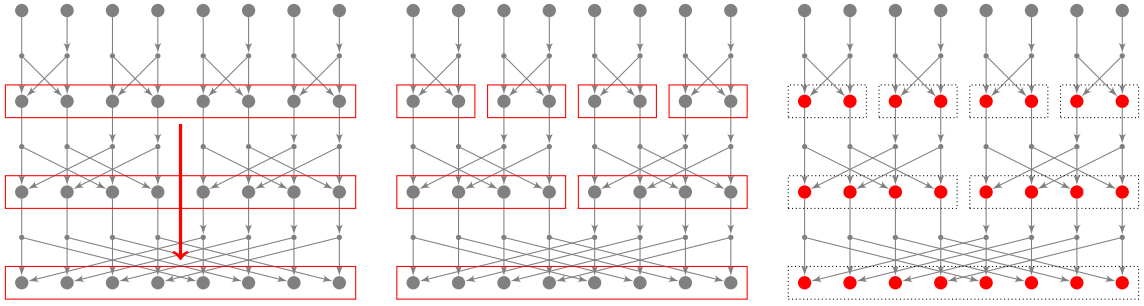


Dies entspricht einem Auflösen des Baumes von den Blättern hin zum Wurzelknoten. Zur Auflösung der Rekursion ist es vorteilhaft, die Berechnung schrittweise in der Breite auszuführen (siehe Abbildung 2). Dafür müssen jeweils immer zwei Fouriertransformationen der Länge M zu einer Fouriertransformation der Länge $2M$ kombiniert werden (siehe Abbildung 2(b)). Für die Kombination der Fouriertransformationen k und $k+1$ im M-ten Schritt ergibt sich aus der Rekursionsformel (9):

$$\begin{aligned} z\big(m+2kM\big) &= z\big(m+2kM\big) + \omega_{2M}^m\, z\big(m+(2k+1)M\big) \\ z\big(m+(2k+1)M\big) &= z\big(m+2kM\big) + \omega_{2M}^m\, z\big(m+(2k+1)M\big) \end{aligned} \qquad (m = 0, \ldots, M-1) \tag{13}$$

Obige Formel stellt den zentralen Rechenschritt der schnellen Fouriertransformation dar, welcher von den Blättern zur Wurzel und für jede Ebene für $k = 0, 1, \cdots$ ausgeführt wird. Diese Operation wird in der Literatur oft als *butterfly* bezeichnet, da die graphische Darstellung an einen Schmetterling erinnert.

Abbildung 3 zeigt die Implementierung der schnellen diskreten Fouriertransformation in Python. In der ersten Hälfte der Implementierung erfolgt die Sortierung und in der zweiten Hälfte dann die Kombination. Im Gegensatz zur Erläuterung oben wird der Index k immer um $2M$ inkrementiert. Dadurch entfällt das Multiplizieren mit $2M$ in der innersten Schleife. Außerdem wurden die Schleifen so angeordnet, dass ω_m aus der innersten Schleife herausgezogen werden kann.



(a) $M = 1, 2, 4, \ldots, N/2$ (b) $k = 0, 1, \ldots, N/2M - 1$ (c) $m = 0, 1, \ldots, M - 1$

Abbildung 2: *Ablauf der Kombinationsphase der schnellen diskreten Fouriertransformation für ein Signal der Länge N.*

```python
import numpy as np
from numpy import pi
import cmath

def dft_fast(c):
    z = c.copy()
    N = len(z)
    j = 0
    for i in range(1,N):
        b = N >> 1
        while b > 0:
            if j & b:
                j -= b
            else:
                j += b
                break;
            b >>= 1
        if i > j:
            z[i], z[j] = z[j], z[i]

    M = 1
    while M < N:
        for m in range(M):
            omega_m = cmath.exp(-2j * pi * m / (2*M))
            for k in range(0, N, 2*M):
                a = z[k+m]
                b = omega_m * z[k+m+M]
                z[k+m] = a + b
                z[k+m+M] = a - b
        M *= 2
    return z

def idft_fast(c):
    N = len(c)
    return 1.0/N * np.conjugate(dft_fast(np.conjugate(c)))
```

Abbildung 3: *Implementierung der schnellen diskreten Fouriertransformation für den Fall $N = 2^n$.*