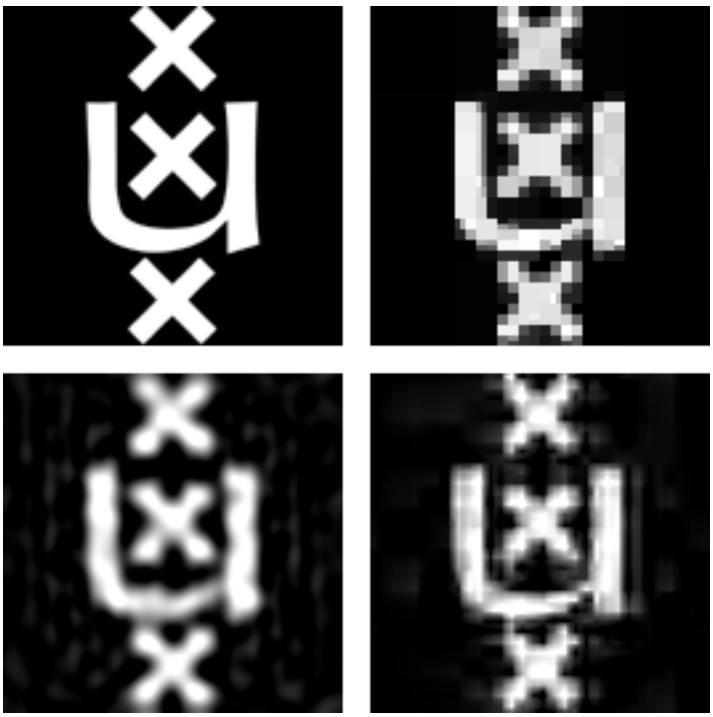
JPEG-2000: de wondere wereld van Wavelets

Jan Westerdiep, Ogier van Garderen

15 juli 2013

Projectverslag jaar 2

Begeleiding: Rob Stevenson



Korteweg-De Vries Instituut voor Wiskunde
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam

Samenvatting

In dit projectverslag zullen wij twee verschillende algoritmes bekijken die in de signaalverwerking van groot belang zijn en zijn geweest. De eerste van deze twee is de Discrete Fouriertransformatie welke ten grondslag ligt aan de JPEG-beeldcompressie-algoritme. Daarna zullen we een overzicht geven van de zogenaamde orthogonale discrete Wavelettransformatie, een versimpelde vorm van het algoritme achter JPEG-2000.

Beide algoritmes zijn *lossy*, wat betekent dat we niet geïnteresseerd zijn in *perfecte reconstructie* maar in een benadering. We zullen concluderen dat voor bepaalde klassen functies de Fouriertransformatie goed werkt maar dat zij praktische nadelen heeft bij signalen met grote sprongen als gevolg van haar globale *dragers*. Verder werpen we in ons verslag een licht op de zogenaamde Fast Fourier Transform, die in $\mathcal{O}(n \log n)$ de transformatie kan vinden.

De Wavelettransformatie is geconstrueerd met de gedachte dat een lokale drager in het geval van signalen met grote sprongen, beter werkt. We zullen beredeneren dat op tweedimensionale signalen er twee verschillende decomposities mogelijk zijn. Beide decomposities zullen we uitvoerig behandelen en analyseren. Naast de eerder genoemde eigenschap heeft de Wavelettransformatie ook een ander praktisch nut: de algoritme loopt in $\mathcal{O}(n)$. Dit feit stelt ons in staat óók bewegend beeld te comprimeren.

Naast een wiskundige analyse heeft dit project ook een groot praktisch deel gekend, namelijk het implementeren van beeldcompressie, waar we ook aandacht aan zullen besteden.

Op het voorblad is te zien, vanaf linksboven en dan met de klok mee:

- Een afbeelding van het UvA-logo;
- Compressie met een Haarwavelet waarbij 1% van de data opgeslagen is;
- Compressie met een Daubechies-2 wavelet met 1% van de data;
- Compressie met een Fouriertransformatie met 1% van de data.

Titel: JPEG-2000: de wondere wereld van Wavelets
Auteurs:
Jan Westerdiep, 10219242, janner@gmail.com
Ogier van Garderen, 10191429, ogiervangarderen@gmail.com
Begeleiding: Rob Stevenson
Einddatum: 15 juli 2013

Korteweg-De Vries Instituut voor Wiskunde
Universiteit van Amsterdam
Science Park 904, 1098 XH Amsterdam
`http://www.science.uva.nl/math`

Inhoudsopgave

Inleiding

Wanneer we foto's naar elkaar versturen, wanneer we grote bestanden downloaden, bij het opslaan van muziek: we hebben allemaal te maken met compressie. Compressie is het encoderen van informatie met minder bits (informatie-eenheden) dan de originele representatie. In het bijzonder heeft beeldcompressie als doel om irrelevante en redundante informatie in het beeld te verminderen zodat zij efficiënter opgeslagen of verstuurd kan worden.

Beeldcompressie kan *lossy* of *lossless* zijn. Lossless compressie wordt veelal gebruikt bij medische doeleinden. De naam zegt het al: lossless compressie gooit geen informatie weg. De reconstructie blijft perfect.

Lossy compressie wordt veel meer gebruikt bij vloeiende afbeeldingen zoals foto's, waar een (klein) verlies van weergave niet erg is wanneer de compressie een substantiëel verschil in grootte oplevert. Wij zullen naar deze vorm van compressie kijken. Er is een aantal manieren van compressie te onderscheiden, te kennen:

- Het verkleinen van het kleurenpallet. Hiermee worden alleen de meestgebruikte kleuren gekozen om zo het aantal bits per pixel te verkleinen;
- Het verlagen van de resolutie. Het gevolg is dat er minder pixels opgeslagen hoeven worden maar de details verdwijnen snel;
- Fractale compressie, wat gebruik maakt van de notie dat er vaak herhalingen in afbeeldingen zitten;
- Compressie op basis van een basistransformatie. Het beeld wordt als functie beschouwd die daarna getransformeerd kan worden naar een andere basis.

Vooral het laatste punt wordt in de praktijk veel gebruikt en ons project zal hier dan ook volledig om draaien. We zullen hiervoor twee algoritmes aan de lezer voorleggen.

JPEG

In 1974 hebben een drietal onderzoekers een transformatie bedacht op basis van de Fouriertransformatie: de *Discrete Cosine Transform*. Deze algoritme werkt – in één dimensie – heel

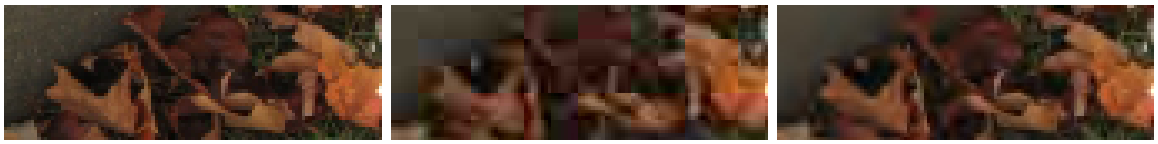
rechttoe-rechtaan. Laat $x[n]$ het originele signaal van lengte N zijn. Dan wordt

$$X[k] = \sum_{n=1}^{N} x[n] \cos\left(\frac{\pi}{N}\left[n + \frac{1}{2}\right] k\right)$$

de getransformeerde. In twee dimensies beschouwen we een matrix en wordt de DCT eerst op elke rij toegepast en daarna op elke kolom.

De algoritme die wij bekeken hebben – genaamd de *Discrete Fourier Transform* – ligt dicht bij de DCT en zullen we grondig behandelen in hoofdstuk 2.

In beide algoritmes wordt het hele originele signaal $x[n]$ gebruikt voor het vinden van de transformatie op positie k. Oneffenheden (zoals randen in de afbeelding) spelen zo in elke coëfficiënt een rol. De uiteindelijke JPEG-algoritme is om deze reden nog iets slimmer. Zij 'hakt' het beeld op in blokken van 8 bij 8 pixels en transformeert elk blok apart. Zo zijn de randen slechts *lokaal* zichtbaar. Op hoge compressieniveaus is dit al snel duidelijk, zoals te zien is in de figuur.



Links: (detail) van een afbeelding. Midden: dit detail onder hoge JPEG-compressie. Rechts: hetzelfde detail onder hoge JPEG-2000-compressie.

JPEG-2000

Wat uit de figuur ook direct duidelijk wordt, is dat de keuze om blokjes van constant 8 bij 8 pixels te nemen nogal arbitrair lijkt. Het zou veel natuurlijker zijn om de grootte van de blokjes te bepalen *op een slimme manier*. Met deze gedachte is in de jaren '90 van de vorige eeuw een andere algoritme bedacht op basis van de zogenaamde Wavelettransformatie, welke in hoofdstuk 3 aan bod zal komen.

Door dilaties wordt steeds het zichtsveld van de basisfunctie verkleind met als gevolg dat er blokken van verschillende grootte komen. We zullen bovendien zien dat deze blokken op twee fundamenteel verschillende manieren gemaakt kunnen worden: zowel met vierkanten (de Mallatdecompositie) als met rechthoeken (het Tensorproduct).

Compressie en praktijk

De vraag die misschien opkomt is, "hoe wordt compressie bereikt?". Het antwoord is tweeledig. In theorie is het het makkelijkst om (bijvoorbeeld) de eerste K basisfuncties te gebruiken

voor de reconstructie. De fout die hiermee bereikt kan worden staat beschreven in hoofdstukken 2 en 3. Dit is wel onder sterke aannames over het beginsignaal die over het algemeen wat genuanceerder liggen.

In de praktijk zijn we slimmer te werk gegaan. We laten namelijk de kleinste coëfficiënten weg. Met de Parsevalgelijkheid (zie §1.2) kunnen we dan concluderen dat de fout ook "klein" zal zijn. We zullen dit onderbouwen in sectie 6.1. Het praktische deel van ons project is te vinden in hoofdstukken 5 en 6. Hierin zullen we kijken naar de algoritmes door reconstructies met verschillende algoritmes te vergelijken.

Onze implementatie

Een belangrijk deel van ons project is de implementatie, waarover is te lezen is in hoofdstuk 4. Voor de implementatie van de algoritmes uit dit verslag is de taal Python gebruikt. Enkele redenen hiervoor waren:

- Voorzieningen om beeld en geluid te laden m.b.v. de *scipy*[14] en *Python Image Library*[15] pakketten. Deze pakketten lieten ons bijvoorbeeld toe om beelden te laden als matrix van kleurwaarden en deze weer op te slaan nadat we deze gereconstrueerd hadden. Zo hoefden we ons niet te bekommeren om het inladen van de verschillende beeldformaten.

- Robuuste pakketten voor wiskunde zoals *numpy*[13] en *scipy*. Dit stelde ons in staat gemakkelijk om te gaan met matrices en lijsten van arbitraire dimensie.

- Duidelijke relatie van wiskunde naar code. Python heeft een syntax die een wiskundige manier van denken en werken ondersteunt, in tegenstelling tot meer declaratieve talen zoals *C* of object-georienteerde talen zoals *Java*.

- Goede abstractie van onderliggende processen. Python is een geïnterpreteerde taal, wat in concreto betekent dat onze programma's draaien op een platform dat de allocatie van geheugen en rekenkracht regelt. Dit versimpelt de implementatie en zorgt dat er makkelijk veranderingen zijn aan te brengen in de code.

Het enige nadeel dat deze keuze teweeg heeft gebracht is dat de snelheid van onze algoritmes tegenvalt omdat we ons niet bezighouden met kleine optimalisaties en vertrouwen op de Python*interpreter*.

1. Basisbegrippen

In dit verslag zullen een aantal termen worden geïntroduceerd die wellicht niet bekend zijn. De eerste en meest belangrijke notie is dat wij *signalen* analyseren. Een signaal is door [17] gedefinieerd:

> Een signaal is een functie die informatie over de kenmerken of het gedrag van een of ander fenomeen overdraagt.

Enkele voorbeelden van een signaal zijn:

Geluid is de vibratie van een medium (zoals lucht) en een geluidssignaal associeert een bepaalde druk met elk moment in de tijd (en mogelijk op elk punt in de ruimte). Geluid wordt door een microfoon omgezet naar een elektrisch signaal welke weer ge*sampled* kan worden tot een discrete lijst waardes in de tijd. Geluid is dus een eendimensionaal signaal.

Afbeeldingen bestaan uit een helderheidssignaal als functie van twee dimensies. Een afbeelding kan bestaan uit een continu domein, zoals bij een schilderij of een analoge foto, of een discreet raster zoals op de computer. Een kleurenafbeelding bestaat meestal uit vier kanalen, één voor de helderheid van elke primaire kleur en een vierde voor het α- of doorzichtigheidskanaal.

Video is een lijst van afbeeldingen. Een punt in een video wordt op deze manier gekarakteriseerd door een punt in de tijd samen met een punt in het vlak. Bewegend beeld is hiermee een driedimensionaal signaal met een al dan niet discreet domein.

In het vervolg zullen we de termen 'signaal' en 'functie' door elkaar gebruiken, tenzij anders aangegeven. Tijdens het project hebben we elk van deze drie voorbeelden bekeken, hoewel de twee- en driedimensionale signalen in ons verslag onze focus zullen krijgen.

1.1. Implementatie

Signaaluitbreiding

Beide algoritmes die we zullen behandelen kunnen enkel omgaan met signalen die een tweemacht lang zijn. Om te zorgen dat een willekeurig signaal ook getransformeerd kan worden,

moet het dus uitgebreid worden voorbij zijn definitiegebied. De meeste bronnen onderscheiden de volgende manieren om het signaal $x_1, x_2, \ldots x_n$ uit te breiden. [6; 20]

Er zijn twee types die grote sprongen in het signaal kunnen veroorzaken (de discrete variant van een *discontinuïteit*).

Zero-padding $x' = 0, \ldots, 0 | x_1, x_2, \ldots, x_n | 0, \ldots, 0$;

Periodic padding $x' = x_1, \ldots, x_n | x_1, x_2, \ldots, x_n | x_1, \ldots, x_n$.

Daarnaast zijn er nog twee types die een niet-vloeiende overgang kunnen veroorzaken (de discrete variant van een *discontinue afgeleide*).

Constant padding $x' = x_1, \ldots, x_1 | x_1, x_2, \ldots, x_n | x_n, \ldots, x_n$;

Symmetric padding $x' = x_n, \ldots, x_1 | x_1, x_2, \ldots, x_n | x_n, \ldots, x_1$.

De keuze van de *signal extension mode* kan gevolgen hebben voor de mogelijkheid tot compressie op de rand. Verder in het verslag (zie §2.4 en §3.9) wordt duidelijk dat de functie f moet voldoen aan bepaalde continuïteitseisen. Wanneer hier niet aan wordt voldaan, is het gevolg meestal dat compressie slecht mogelijk is.

Python syntax

We zullen in de sectie over onze implementatie wat stukken Python code gebruiken, hoewel de taal voornamelijk leest alsof het pseudocode is dienen we wat syntactische elementen uit te leggen.

Blokhaaknotatie De elementen van een lijst `x` worden in python met `x[i]` aangegeven, waarbij `i` van 0 oploopt tot de lengte van de lijst.

List-comprehensions De notatie `[x for y in z]` is kortere schrijwijze voor een lijst die geconstrueerd word door alle elementen `y` in `z` af te gaan en steeds x, een statement dat van `y` afhangt, toe te voegen aan de lijst.

Function-mapping De functie `map(f,x)` is een mapping van een functie `f` over een lijst `x`, met gebruik van list-comprehensions word dit `[f(y) for y in x]`.

Reductie De functie `reduce(f,x)` reduceert een lijst tot één waarde, de functie `f` pakt hiervoor de elementen `x[0],x[1]` en geeft `a = f(x[0],x[1])` terug, vervolgens wordt de waarde `f(a,x[2])` berekend en weer in `a` opgeslagen, zo gaat de algoritme verder tot de lijst afgelopen is.

List-Concatenation De notatie `a + b`, met `a` en `b` lijsten, wordt gebruikt om twee lijsten te *concateneren* (samenvoegen). Bijvoorbeeld `[1,2,3] + [3,4,5]` geeft `[1,2,3,3,4,5]`.

1.2. Wiskunde

Notatie

Analoog aan de notatie $f(x)$ die we in het algemeen voor een functie gebruiken, voeren we de blokhaaknotatie $f[x]$ in die duidt op een discrete functie

$$f : A \subset \mathbb{Z} \to \mathbb{R} \quad \text{of} \quad f : A \subset \mathbb{Z} \to \mathbb{C}.$$

Vaak zullen we voor A het interval $\{1, \ldots, n\}$ nemen, dan is f te zien als een vector in $\mathbb{R}^n$ of $\mathbb{C}^n$. In het bijzonder duiden we met deze notatie dus de cellen van een vector aan.

We zullen de notatie ook uitbreiden naar meer dimensies door de notatie uit te breiden volgens

$$\begin{aligned} f : A_1 \times \cdots \times A_m \subset \mathbb{Z}^m &\to K \\ (x_1, \ldots, x_m) &\mapsto f[x_1, \ldots, x_m], \end{aligned}$$

Uit zo'n m-dimensionale functie f kunnen we vervolgens op een natuurlijke manier een $(m-k)$-dimensionale functie construeren door k coördinaten vast te nemen:

$$\begin{aligned} f \,\big|_{\, x_{i_1}=a_{i_1}, \ldots, x_{i_k}=a_{i_k}} : A_{i_{k+1}} \times \cdots \times A_{i_m} &\to K \\ (a_{i_{k+1}}, \ldots, a_{i_m}) &\mapsto f[a_1, \ldots, a_m] \end{aligned}$$

Hier zijn i_1'de tot en met de i_k'de coördinaat vastgenomen. Voor een matrix kan deze functie bijvoorbeeld gezien worden als een rij of een kolom. Dan geeft i_1 aan of de rij of kolom vast staat en geeft a_{i_1} aan welke rij respectievelijk kolom bekeken wordt.

We willen ook de convolutie-notatie introduceren. We schrijven voor m-dimensionale functies f, g de convolutie $f \star g$ als:

$$(f \star g)[a_1, \ldots, a_m] = \sum_{k_1=-\infty}^{\infty} \cdots \sum_{k_m=-\infty}^{\infty} f[k_1, \ldots, k_m] \cdot g[a_1 - k_1, \ldots, a_m - k_m]$$

We zullen deze notatie toepassen om onze algoritmen inzichtelijker op te schrijven.

Complexiteit

Aangezien we algoritmes behandelen in het verslag willen we hiervan de tijdscomplexiteit bepalen. Deze eigenschap bepaalt namelijk hoe de tijd die het een machine kost oploopt met de grootte van de input. We voeren daarom o, $\mathcal{O}$ en θ als notatie in.

$$\begin{array}{rcrcrcrcl} f \in \mathcal{O}(g) & \Leftrightarrow & \exists c : & \exists x_0 : & \forall x > x_0 & & & |f(x)| & \leq c|g(x)| \\ f \in o(g) & \Leftrightarrow & \forall c : & \exists x_0 : & \forall x > x_0 & & & |f(x)| & \leq c|g(x)| \\ f \in \theta(g) & \Leftrightarrow & \exists c_1, c_2 : & \exists x_0 : & \forall x > x_0 & c_1|g(x)| \leq & |f(x)| & \leq c_2|g(x)| \\ f \simeq g & \Leftrightarrow & \exists c_1, c_2 : & & \forall x & c_1|g(x)| \leq & |f(x)| & \leq c_2|g(x)| \end{array}$$

Merk op dat o een *sterker* begrip is dan $\mathcal{O}$: als $f \in o(g)$ dan $f \in \mathcal{O}(g)$. Ook geldt dat $f \simeq g \implies f \in \theta(g) \implies f \in \mathcal{O}(g)$. We zouden dus ook alles over één kam kunnen strijken en enkel $\mathcal{O}$ gebruiken (maar voor de volledigheid zullen we dit niet doen). Als laatste is er ook een eenzijdige variant van $\simeq$, namelijk $\lesssim$: $f \lesssim g$ wat betekent dat er een C is zó dat $f \leq C \cdot g$. Er geldt dus zeker dat $f \simeq g$ implicieert dat $f \lesssim g$.

Ruimtes

Een familie van ruimtes die wij gedurende het hele verslag zullen gebruiken is die van de Lebesgue-ruimtes. De L_p-ruimte is de ruimte over functies van $\mathbb{R}$ naar $\mathbb{C}$ die p'e-machts integreerbaar zijn:

$$\|f\|_{L_p} := \left(\int_{-\infty}^{\infty} |f(t)|^p dt\right)^{1/p} < \infty. \tag{1.1}$$

Wij zullen vooral de L_2-ruimte bekijken. Deze heeft als extra eigenschap dat zij een Hilbertruimte is, met als inproduct

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(t)g^*(t)dt, \tag{1.2}$$

waarbij $g^*(t)$ de complex geconjugeerde van g aangeeft. Het is duidelijk dat $||f||_{L_2} := \sqrt{\langle f, f\rangle}$ nu overeenkomt met (1.1), zodat we inderdaad de L_2-ruimte bekijken. Merk op dat L_2 ook de *enige* Hilbertruimte is van deze vorm.

Naast deze L_2-ruimte over $\mathbb{R}$ zullen we ook haar tegenhanger over $\mathbb{R}^n$ bekijken. Integraaltekens uit (1.1) en (1.2) worden nu gewoon integralen over $\mathbb{R}^n$. Wanneer we spreken over $L_2(\Omega)$ voor $\Omega \subset \mathbb{R}^n$, wordt een integraal over Ω bedoeld.

Een belangrijk lemma dat we voor deze ruimtes zullen gebruiken is het lemma van Parseval.

Lemma 1.1 (Parsevalgelijkheid [9])**.** *Laat f een functie zijn in $L_2(\Omega)$ met $\Omega \subset \mathbb{C}^n$ of $\Omega \subset \mathbb{R}^n$ die geschreven kan worden in een aftelbare orthonormale basis $\mathcal{B} = \{g_m\}$. Dan geldt*

$$\|f\|^2 = \sum_{g_m \in \mathcal{B}} |\langle f, g_m\rangle|^2.$$

Verder zullen we ook een discrete tegenhanger van L_2 bekijken, namelijk de ℓ_2-ruimte:

$$\ell_2 := \Big\{ f : \mathbb{Z} \to \mathbb{C} \,\Big|\, \|f\|^2 = \sum_{m=-\infty}^{\infty} |f[m]|^2 < \infty \Big\}.$$

We definiëeren op deze ruimte ook een inproduct, analoog aan dat op L_2:

$$\langle f, g\rangle = \sum_{m=-\infty}^{\infty} f[m] \cdot g^\star[m].$$

Vaak zullen we functies f bekijken die een eindige drager hebben, deze zijn in het bijzonder altijd een element van ℓ_2. Ook dienen we op te merken dat de Parsevalgelijkheid ook geldt voor ℓ_2.

2. Fourier

Bij het analyseren van periodieke functies is de Fouriertransformatie het gereedschap bij uitstek. Het is een manier om een signaal te karakteriseren aan de hand van een bereik aan frequenties en dit maakt het een overtuigende manier om 'nette' periodieke functies te beschrijven. We kunnen de eis van periodiciteit ook loslaten als we naar functies op een interval kijken door dit op te vatten als één fase van een periodieke functie.

Iets concreter kijken we dus naar functies $f \in L_2([a,b])$. De basisfuncties waarmee we vervolgens de analyse uitvoeren worden gegeven door de complexe e-machten.

Definitie 2.1 (Fourierbasis)**.** Bekijk de functieruimte $L_2([a,b])$. We definiëren de verzameling $F_{a,b}$ door:

$$F_{a,b} := \left\{ \phi_k(x) = \tfrac{1}{\sqrt{b-a}} e^{2\pi i \cdot k \frac{x-a}{b-a}} \;\middle|\; k \in \mathbb{Z} \right\}$$

We noemen $F_{a,b}$ de Fourierbasis van $L_2([a,b])$.

Het woord basis is hier met recht gebruikt. Het is immers bekend dat de complexe e-machten loodrecht staan onder het inproduct op L_2. Om nu een willekeurige functie te schrijven in deze basis, introduceren we de Fouriergetransformeerde.

Definitie 2.2 (Fouriertransformatie)**.** Zij gegeven een functie $f \in L_2([a,b])$. Schrijf dan $\hat{f} : \mathbb{Z} \to \mathbb{C}$ met entries gedefinieerd volgens:

$$\hat{f}[k] = \frac{1}{\sqrt{b-a}} \cdot \langle f, \phi_k \rangle = \frac{1}{b-a} \int_a^b f(x) \cdot e^{-2\pi i \cdot k \frac{x-a}{b-a}} \mathrm{d}x.$$

We noemen $\hat{f}$ de Fouriergetransformeerde van f.

Definitie 2.3 (Inverse Fouriertransformatie)**.** Gegeven een Fouriergetransformeerde $\hat{f}$ van een functie $f \in L_2([a,b])$, is de reconstructie f° van f gegeven door:

$$f^\circ(x) = \sum_{k=-\infty}^{\infty} \hat{f}[k] \phi_k(x).$$

In [16] is aangetoond dat f° een reconstructie geeft van f die voldoet aan:

$$||f - f^\circ||_{L_2([a,b])} = 0 \qquad \text{wanneer} \qquad f \in L_2([a,b]) \tag{2.1}$$

$$f(x) = f^\circ(x) \qquad \text{wanneer} \qquad f \in C^1. \tag{2.2}$$

Merk op dat $||f - f^\circ||_{L_2([a,b])} = 0$ niet betekent dat $f - f^\circ = 0$.

2.1. De *Discrete Fourier Transform*

Zoals we gezien hebben in het voorafgaande, kan de Fouriertransformatie gebruikt worden om continue signalen te karakteriseren door verschillende frequenties. Een groot gebied binnen de signaalanalyse is echter van discrete aard aangezien hier veelal digitale instrumenten worden gebruikt.

Om discrete signalen te analyseren lijkt het voor de hand te liggen om een deze als stapfuncties te zien; een stapfunctie zit immers in L_2. De discontinuïteiten van de stapfunctie leiden echter tot ongewenste resultaten. Elke reconstructie in termen van eindig veel continue basisfuncties is namelijk weer continu, zodat voor een perfecte reconstructie van een stapfunctie met deze methode altijd een oneindige rij coëfficiënten nodig is. Bovendien is het moeilijk om uit deze coëfficiënten relevante informatie te destilleren over de aard van het signaal.

In plaats van de discrete signalen in te bedden in L_2 zullen we ze beschouwen als rijen in ℓ_2 (zie sectie 1.2). Hiervoor zullen we de Fourier-basis discretiseren en ons richten op de ruimte $\mathbb{R}^n$. We veranderen daarvoor de coördinaten naar een discrete j volgens

$$\frac{x-a}{b-a} \leftrightarrow \frac{j}{n}$$

Op deze manier zal de discretisatie in de limiet naar het continue geval overgaan.

Definitie 2.4 (Discrete Fourierbasis)**.** Gegeven zij de ruimte $\mathbb{C}^n$. Definiër dan de verzameling

$$S_n := \left\{ s_k[j] = e^{2\pi i \cdot kj/n} \,\middle|\, k, j \in \{1, \ldots, n\} \right\},$$

als de *discrete Fourierbasis* op deze ruimte met basisvectoren s_k.

We weten dat de basisvectoren loodrecht staan vanwege de eigenschap:

$$\langle s_k, s_j \rangle = \sum_{m=1}^{n} s_k[m] \cdot \overline{s_j[m]} = \sum_{m=1}^{n} e^{2\pi i \cdot m(k-j)/n} = \begin{cases} 0 & \text{als } k \neq j \\ n & \text{als } k = j. \end{cases}$$

Hierdoor kunnen we een discreet signaal x schrijven in termen van S_n een vector van inproducten te definiëren. We noemen X de discrete Fouriertransformatie (DFT) met entries:

$$X[k] = \tfrac{1}{n}\langle x, s_k \rangle \quad k \in \{1, \ldots, n\}.$$

Vervolgens hebben we een inverse voor deze operatie die X afbeeldt op de reconstructie x° volgens:

$$x^\circ[j] = \langle X, s_j^{-1} \rangle \quad j \in \{1, \ldots, n\}.$$

Waarbij we de notatie $s_j^{-1} = (s_j[1]^{-1}, \ldots, s_j[n]^{-1})$ gebruiken. Vanwege de orthogonaliteit van de basis S_n is dit een perfecte reconstructie ($x[k] = x^\circ[k]$).

Om de claim te ondersteunen dat de DFT-methode een echte discretisatie is van de continue Fouriertransformatie, willen we bewijzen dat deze algoritme voor een steeds fijnere selectie

van waardes van een functie in de limiet hetzelfde resultaat geeft als de continue Fouriertransformatie. Zoals gebruikelijk bij het overschakelen van een discrete naar een continue setting kunnen we dit doen door de definitie van de Riemannintegraal toe te passen op de sommatie die voor handen ligt.

Stelling 2.5 (Limiet van discrete Fouriertransformatie)**.** *Gegeven zij een functie $f \in L_2([a,b])$. Bekijk een discretisatie van f in n gelijke intervallen zodat de discrete f precies de randwaarde van elk interval inneemt. Dan geldt dat de discrete Fouriergetransformeerde limiteert naar de algemene Fouriergetransformeerde wanneer $n \to \infty$.*

Bewijs. Gegeven een interval $[a,b]$ kunnen we een partitie P maken in n gelijke delen: laat $P = \{a = t_0, t_1, .., t_n = b\}$ met $t_j = a + \frac{j(b-a)}{n}$. We discretiseren onze functie f door uit elk interval $[t_{j-1}, t_j]$ van de partitie de randwaarde in $x_j = t_j$ te selecteren, dus

$$f[j] = f(x_j) = f(\frac{b-a}{n} j + a).$$

De DFT van de discrete functie $f[\cdot]$ wordt dan gegeven door:

$$F[k] = \frac{1}{n} \sum_{j=1}^{n} f[j] \cdot s_k^{-1}[j].$$

We schrijven dit om in termen van onze non-discrete functie door $x_j \in [a,b]$ te vervangen door zijn discrete tegenhanger en krijgen zo:

$$\begin{aligned} F[k] = & \quad \tfrac{1}{n} \textstyle\sum_{j=1}^{n} f[j] \cdot s_k^{-1}[n \cdot \frac{x_j - a}{b-a}] \\ = & \quad \tfrac{1}{n} \textstyle\sum_{j=1}^{n} f(x_j) \cdot e^{-2\pi i \cdot k \frac{x_j - a}{b-a}} \\ = & \quad \tfrac{1}{\sqrt{b-a}} \textstyle\sum_{j=1}^{n} f(x_j) \cdot \phi_k^*(x_j) \cdot \frac{b-a}{n} \end{aligned}$$

We merken op dat de term $\frac{b-a}{n}$ precies de grootte is van de intervallen van de partitie en dat we het geheel omgeschreven hebben in termen van onze continue functies f en ϕ_k. Omdat het product $f \cdot \phi_k$ integreerbaar is moet voor elke partitie P met waarden in punten x_j uit elk interval deze sommatie convergeren naar de integraal

$$\frac{1}{\sqrt{b-a}} \int_{[a,b]} f(x) \phi_k^*(x) \mathrm{d}x$$

wanneer we de maaswijdte ($\frac{b-a}{n}$) naar 0 laten gaan. [18] Dit is duidelijk het geval wanneer we de limiet $n \to \infty$ nemen. Dus is de DFT een goede discretisatie van de Fouriergetransformeerde. □

2.2. De Fast Fourier Transform

De snelheid van de DFT-algoritme valt in de praktijk nogal tegen. Het nemen van n inproducten over vectoren van lengte n heeft namelijk een tijdscomplexiteit van $\mathcal{O}(n^2)$. Dit staat de directe implementatie van de DFT voor praktische toepassingen in de weg. Daarom is er een alternatieve algoritme, de *Fast Fourier Transform.*

Algoritme (Fast Fourier Transform)**.** *Gegeven een inputsignaal x van lengte $n = 2^m$, geeft het algoritme FFT een lijst terug van waardes X van lengte $n = 2^m$ als volgt:*

Als $m = 0$ dan geeft de FFT de lijst (van één element) direct terug:

$$X = x.$$

Wanneer $m \neq 0$ splitsen we de lijst x op in lijsten ϵ, o van zijn even en oneven indices:

$$\begin{aligned} \epsilon[k] = &\quad x[2k] &&\text{voor } k < n/2, \\ o[k] = &\quad x[2k+1] &&\text{voor } k < n/2. \end{aligned}$$

Vervolgens voeren we hierop het FFT algoritme uit om de volgende lijsten te verkrijgen:

$$\begin{aligned} E = &\quad FFT(\epsilon), \\ O = &\quad FFT(o). \end{aligned}$$

Hiermee wordt de output van de algoritme geconstrueerd volgens:

$$X[k] = \begin{cases} E[k] & + \quad e^{-2\pi ik/n} \cdot O[k] & k < n/2, \\ E[k-n/2] & - \quad e^{-2\pi i(k-n/2)/n} \cdot O[k-n/2] & k \geq n/2. \end{cases}$$

Dit is dus een recursief gedefinieerde algoritme dat een signaal meermaals halveert. Het is gegarandeerd dat deze algoritme afloopt vanwege de conditie op $m = 0$ samen met de halvering van de input bij elke stap. Een belangrijke voorwaarde voor de relevantie van de FFT is nu dat de algoritme hetzelfde resultaat geeft als de DFT-algoritme en dit zullen.

Stelling 2.6. *Het uitvoeren van de Fast Fourier Transform algoritme op een datasetgeeft dezelfde getransformeerde als de discrete Fouriertranformatie.*

Bewijs. We bewijzen met inductie naar n. Onze aanname is dat de FFT-algoritme voor x van lengte $n = 2^m$ gelijk is aan de DFT van x, ofwel

$$X[k] = \sum_{k=1}^{n} x[j] \cdot e^{-2\pi i \cdot jk/n}.$$

Dit geldt duidelijkerwijs wanneer $m = 0$, onze basisstap. Hiervoor geldt namelijk:

$$X[k] = x[k] = x[1] = \sum_{k=1}^{2^0} x[j] \cdot e^{-2\pi i \cdot 1/2^0}.$$

Vervolgens passen we inductie toe naar m door onze aanname voor $m-1$ te gebruiken. We vullen hier $E[k]$ en $O[k]$ in de vergelijking voor $X[k]$ in, deze hebben immers lengte $n = 2^{m-1}$.

$$X[k] = \begin{cases} \sum_{j=1}^{n/2} \epsilon[j] \cdot e^{-2\pi i \cdot kj \cdot 2/n} & + \quad e^{-2\pi i \cdot k/n} \sum_{j=1}^{n/2} o[j] \cdot e^{-2\pi i \cdot kj \cdot 2/n} & k < n/2 \\ \sum_{j=1}^{n/2} \epsilon[j] \cdot e^{-2\pi i \cdot (k-n/2)j \cdot 2/n} & - \quad e^{-2\pi i \cdot (k-n/2)/n} \sum_{j=1}^{n/2} o[j] \cdot e^{-2\pi i \cdot (k-n/2)j \cdot 2/n} & k \geq n/2 \end{cases}$$

Merk op dat we de e-machten in het tweede geval kunnen vereenvoudigen volgens

$$e^{-2\pi i\cdot(k-n/2)j\cdot 2/n} = e^{-2\pi i\cdot kj\cdot 2/n} \quad , \quad e^{-2\pi i\cdot(k-n/2)/n} = -e^{-2\pi i\cdot k/n},$$

waardoor het gevalsonderscheid wegvalt, aangezien beide vergelijkingen nu identiek zijn. Het resultaat is $X[k]$ als sommatie over de lijsten ϵ en o. Vul de relatie voor ϵ en o met x in en neem de factor voor de oneven indices mee in de sommatie om te krijgen

$$X[k] = \sum_{j=1}^{n/2} x[2j]\cdot e^{-2\pi i\cdot k(2j)/n} + \sum_{j=1}^{n/2} x[2j+1]\cdot e^{-2\pi i\cdot k(2j+1)/n} = \sum_{j=1}^{n} x[j]\cdot e^{-2\pi i\cdot kj/n}.$$

Dit bewijst dat de FFT hetzelfde resultaat levert als de DFT-algoritme. Het bewijs voor de gelijkheid van de iDFT en de inverse FFT is hetzelfde wanneer men de substitutie $-2\pi i \to 2\pi i$ uitvoert. □

Opmerking. We hebben hier telkens aangenomen – en zullen deze aanname ook doorzetten – dat de lengte van het ingangssignaal een macht van 2 is. Dit is een belangrijke eigenschap waar de variant van de FFT-algoritme dat hier gebruikt wordt, door werkt. Deze versie van FFT wordt *Radix-2 Decimation In Time* van Cooley-Tukey genoemd. Algemenere vormen van deze algoritme worden ook toegepast in geoptimaliseerde algoritmes maar om de implementatie te versimpelen is voor Radix-2 gekozen. Eventuele verschillen in afmetingen tussen een signaal en een 2-macht zijn opgelost met signaalextensie, zoals eerder beschreven.

Complexiteit van de Fast Fourier Transform

We zullen de claim bewijzen dat de complexiteit van de FFT werkelijk beter lager is dan die van de DFT. Hiervoor hebben we de volgende stelling uit de Complexiteitstheorie nodig.

Stelling 2.7 (Akra-Bazzi [1])**.** *Zij $T : \mathbb{N} \to \mathbb{R}$ een recurrente betrekking van de vorm*

$$T(n) = \begin{cases} c_0 & \text{als } n \le d, \\ aT(n/b) + f(n) & \text{anders,} \end{cases}$$

waarbij $a, b, d \in \mathbb{N}$, $c_0 \in \mathbb{R}$ en f een functie $f : \mathbb{N} \to \mathbb{R}$ die voldoet aan

$$\exists k \in \mathbb{N} \,:\, f(n) \in \theta(n^{\log a/\log b}\log^k n),$$

dan wordt de orde van $T(n)$ gegeven door:

$$T(n) \in \theta(n^{\log a/\log b}\log^{k+1} n).$$

De recurrente betrekking $T(n)$ kan gezien worden als het aantal stappen dat een machine nodig heeft om de algoritme uit te voeren. Deze stelling is nu voldoende om een uitspraak te kunnen doen over de complexiteit.

Stelling 2.8 (Complexiteit van de FFT)**.** *De Fast Fourier Transform algoritme heeft een tijdscomplexiteit van $\theta(n\log n)$ voor input van lengte $n = 2^m$.*

Bewijs. We schrijven de FFT in pseudocode.

```
function FFT(x)
    n ← lengte(x)                                  ▷ Assumptie: n = 2^m voor een m
    if n == 1 then
        X ← x
    else
        E ← FFT(x[0 :: 2])                         ▷ FFT op even indices
        O ← FFT(x[1 :: 2])                         ▷ FFT oneven indices
        for i = 0 to n − 1 do
            if i < n/2 then
                X[i] ← E[i] + e^{−2iπk/n} · O[i]
            else
                X[i] ← E[i] − e^{−2iπk/n} · O[i]
            end if
        end for
    end if
    return X
end function
```

Deze algoritme is recursief zodat we de complexiteit kunnen schrijven door middel van een recurrente betrekking. Laat hiervoor $T(n)$ het aantal berekeningen zijn dat de algoritme kost bij een invoersignaal van lengte n. We maken een gevalsonderscheid: als de lijst lengte 1 heeft geven we deze direct terug (1 berekening). Bij een lijst van lengte groter dan 1 splitsen we de lijst op in de even en oneven entries en voeren we op beiden weer FFT uit. Vervolgens voeren we nog n maal een vast aantal (namelijk N) berekeningen uit om tot het eindresultaat te komen. In formulevorm geeft dit de *recurrente betrekking*

$$T(n) = \begin{cases} 1 & \text{als } n = 1 \\ 2 \cdot T(n/2) + N \cdot n & \text{anders.} \end{cases}$$

Gebruik nu stelling 2.7. De recurrente betrekking voor de complexiteit van de FFT is inderdaad van dezelfde vorm als die van $T(n)$ in bovenstaande stelling. Laat hiervoor namelijk $a = b = 2$, $c_0 = d = 1$ en $f(n) = N \cdot n$, waarvoor geldt dat

$$f(n) \in \theta(n^{\log 2/\log 2} \log^0 n) = \theta(n).$$

Dit betekent dat $T(n) \in \theta(n \log n)$. Hiermee hebben we bewezen dat de FFT en daarmee de iFFT binnen tijdscomplexiteit $\mathcal{O}(n \log n)$ lopen. □

2.3. Discrete Fourier Transform in meer dimensies

Een eigenschap van de DFT-algoritme is dat het op een natuurlijke manier uit te breiden is naar hogere dimensies. Per constructie is er dus een manier om de FFT ook te beschrijven voor hogere dimensies. Het idee hierbij is om het Tensorproduct te nemen van meerdere bases.

Definitie 2.9 (Multidimensionale Discrete Fourierbasis)**.** Gegeven zij een signaalruimte van de vorm $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \ldots \times \mathbb{R}^{n_m}$. We definiëren de multidimensionale discrete Fourierbasis behorende bij deze signaalruimte als

$$S_{\mathbf{n}} = S_{n_1} \otimes S_{n_2} \otimes \ldots \otimes S_{n_m} = \left\{ s_{\mathbf{k}}[\mathbf{j}] = s_{k_1}[j_1] \cdot s_{k_2}[j_2] \cdot \ldots \cdot s_{k_m}[j_m] \,\middle|\, s_{k_i} \in S_{n_i}, j_i \in \{1, \ldots, n_i\} \right\}$$

Met basisvectoren $s_{\mathbf{k}}$, waar $\mathbf{k}$ een indexvector uit $\{1, \ldots, n_1\} \times \ldots \times \{1 \ldots, n_m\}$ is. Het feit dat dit een basis is, kan gevonden worden in [12].

Wanneer we nu een m-dimensionaal signaal bekijken van lengte $n_1 \times \cdots \times n_m$, dan kunnen we hierop een multidimensionale Fouriertransformatie op definiëren door het inproduct te generaliseren naar onze m-dimensionale signaalruimte.

$$X[\mathbf{k}] = \tfrac{1}{\mathbf{n}} \sum_{\mathbf{j}=\mathbf{1}}^{\mathbf{n}} x[\boldsymbol{j}] s_{\mathbf{k}}^{-1}[\boldsymbol{j}] = \left(\prod_{i=1}^{m} \tfrac{1}{n_i} \right) \cdot \sum_{j_1=1}^{n_1} \ldots \sum_{j_m=1}^{n_m} x[j_1, \ldots, j_m] \cdot e^{-2\pi i \cdot k_m \cdot j_m / n_m} \cdot \ldots \cdot e^{-2\pi i \cdot k_1 \cdot j_1 / n_1}. \tag{2.3}$$

Echter, oals we voor de DFT al gezien hadden, is het uitrekenen van al deze inproducten erg tijdrovend. We zullen daarom de Multidimentionale Discrete Fouriertransformatie (MDFT) op een andere manier definiëren zodat we beter gebruik kunnen maken van de DFT- en FFT-algoritmen die we al gevonden hebben.

Algoritme (Multidimensionaal DFT-algoritme)**.** *Gegeven is een m-dimensionale input x en een huidig level t. We schrijven de algoritme met output X_t volgens*

$$X_t[\boldsymbol{k}] = \begin{cases} DFT(X_{t-1} \,|\, _{x_1=k_1, \ldots, x_{t-1}=k_{t-1}, x_{t+1}=k_{t+1}, \ldots, x_m=k_m})[k_t] & \text{als } t > 0, \\ x[k_t] & \text{als } t = 0. \end{cases} \tag{2.4}$$

We beweren dat X_m de multidimensionale Fouriertransformatie van een m-dimensionaal signaal geeft zoals in (2.3)*.*

Bewijs. We voeren inductie naar de dimensie m. De inductiehypothese wordt dat voor een dimensie m de vergelijking voor X_m uit de algoritme gegeven wordt door (2.3).

Als $m = 1$ dan geldt:

$$X_1[k_1] = DFT(X_0)[k_1] = DFT(x)[k_1],$$

wat inderdaad de DFT in 1 dimensie geeft.

We gaan vervolgens door met de inductiestap. Laat $m > 1$. Schrijf kort

$$\tilde{X}_{t-1}[k_t] := X_{t-1} \,|\, _{x_1=k_1, \ldots, x_{t-1}=k_{t-1}, x_{t+1}=k_{t+1}, \ldots, x_m=k_m}[k_t].$$

Dan

$$X_m[\boldsymbol{k}] = \mathrm{DFT}(\tilde{X}_{m-1})[k_m] = \frac{1}{n_m} \sum_{j_m=1}^{n_m} \tilde{X}_{m-1}[j_m] s_{k_m}^{-1}[j_m].$$

We vatten $\tilde{X}$ op als een vector van lengte n_m van $m-1$ dimensionale Fouriergetransformeerden. Dit mag omdat

$$X_{m-1}\,\big|_{\,x_1=k_1,\ldots,x_{m-1}=k_{m-1}}[k_m] = X_{m-1}\,\big|_{\,x_m=k_m}[k_1,\ldots,k_{m-1}]$$

en daarmee is $X_{m-1}\,\big|_{\,x_m=k_m}$ een $m-1$-dimensionaal object dat weer wordt gegeven door de relatie in (2.4).

Omdat we de aanname voor $m-1$ dimensies al bewezen hebben geldt nu:

$$X_m[\boldsymbol{k}] = \frac{1}{n_m}\sum_{j_m=1}^{n_m}\left(\frac{1}{n_{m-1}}\sum_{j_{m-1}=1}^{n_{m-1}}\left(\cdots\frac{1}{n_1}\sum_{j_1=1}^{n_1}x[j_1,\ldots,j_m]\cdot s_{k_1}^{-1}[j_1]\cdots\right)s_{k_{m-1}}^{-1}[j_{m-1}]\right)s_{k_m}^{-1}[j_m],$$

wat precies is wat we wilden aantonen. □

Een belangrijk gevolg van de definitie van deze algoritme is dat de DFT-term in (2.4) gemakkelijk vervangen kan worden door een FFT-term. Beiden geven immers dezelfde output. Daarmee hebben we ook direct een MFFT gevonden.

Opmerking. Omdat bovenstaand algoritme niet in lineaire tijd loopt, neemt de wachttijd snel toe bij grote signalen en hogere dimensie. Dit wordt al snel een praktisch bezwaar en is dan ook een reden waarom we geen driedimensionale signalen hebben bekeken bij de implementatie van de Fouriertransformatie.

2.4. Compressie van een signaal onder FFT

Tot zover is de Discrete Fourieranalyse besproken met de gedachte van perfecte reconstructie, aan de hand van een complete set coëfficiënten. Het doel van dit project is echter om signalen te comprimeren: te reconstrueren aan de hand van een gelimiteerde dataset. Om deze analyse te versimpelen, zullen in deze sectie een aantal inzichten aan bod komen die de fout van zo'n reconstructie relateren aan de grootte van de dataset.

Lemma 2.10 (Riemann-Lebesque [2])**.** *Wanneer $g \in L_1(\mathbb{R})$, dan geldt*

$$G(z) = \left|\int_{-\infty}^{\infty} g(t)e^{-2\pi izt}dt\right| \to 0 \text{ voor } z \to \infty.$$

Stelling 2.11 (Daling van de coëfficiënten van de Fouriergetransformeerde)**.** *Laat $f \in L_2([a,b])$. Stel er is een n zó dat $f \in C^n$ en voor $0 \le j \le n$ geldt dat $f^{(j)}(a) = f^{(j)}(b)$ – f en haar afgeleides zijn periodiek. Dan geldt dat de Fouriergetransformeerde $\hat{f}[k]$ in absolute waarde daalt met k volgens $o(|k|^{-n})$.*

Bewijs. Laat $f \in L_2([a,b])$ zodat f voldoet aan de voorwaarden in de stelling voor een bepaalde n. De Fouriergetransformeerde van f wordt gegeven door:

$$\hat{f}[k] = \tfrac{1}{\sqrt{b-a}}\langle f, \phi_k\rangle.$$

De constante in deze vergelijking heeft geen invloed op de orde, dus richten we ons op het inproduct. We schrijven dit uit tot een integraal en voeren vervolgens – f is immers differentieerbaar – partiële integratie uit:

$$\begin{aligned} |\langle f, \phi_k \rangle| = & \left| \int_a^b f(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right| = \left| \tfrac{b-a}{2\pi i k} \left[f(x) \cdot e^{-2\pi i k \frac{x-a}{b-a}} \right]_a^b \right| + \left| \tfrac{b-a}{2\pi i k} \right| \left| \int_0^1 f'(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right| \\ = & \quad |f(b) \cdot 1 - f(a) \cdot 1| + \tfrac{b-a}{2\pi |k|} \left| \int_a^b f'(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right| = \tfrac{b-a}{2\pi |k|} \left| \int_a^b f'(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right|. \end{aligned}$$

Hier hebben we gebruikt dat alle afgeleiden van 0 tot n periodiek zijn. Omdat f van de klasse C^n is en de afgeleiden weer periodiek zijn kunnen we dit herhaald toepassen en we verkrijgen daarmee

$$\left| \int_a^b f(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right| = (\tfrac{2\pi}{b-a} |k|)^{-n} \left| \int_a^b f^{(n)}(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right|.$$

Vermenigvuldig beide kanten met $(\frac{2\pi}{b-a}|k|)^n$ om te vinden dat

$$(\tfrac{2\pi}{b-a}|k|)^n |\langle f, \phi_k \rangle| = \left| \int_a^b f^{(n)}(x) e^{-2\pi i k \frac{x-a}{b-a}} \, dx \right|.$$

We willen graag lemma 2.10 toepassen op de rechterkant. Merk daartoe op dat f continu is op een gesloten interval, met als gevolg dat zij uniform continu is en zij haar maximum en minimum hier dus aanneemt.

Daarmee is de integraal van $|f|$ begrensd en dus is $f \in L_1([a,b])$. Maar een functie die integreerbaar is op $[a,b]$ en daarbuiten nul, is integreerbaar op $\mathbb{R}$. Dus we mogen Riemann-Lebesgue gebruiken om te zien dat

$$(\tfrac{2\pi}{b-a}|k|)^n |\langle f, \phi_k \rangle| \to 0 \text{ als } |k| \to \infty.$$

Maar dit betekent precies dat $|\langle f, \phi_k \rangle| \in o((\frac{2\pi}{b-a}|k|)^{-n}) = o(|k|^{-n})$. □

De analyse van de coëfficiënten die we hier gegeven hebben vertaalt direct naar de fout die we krijgen bij reconstructie van een Fouriergetransformeerde functie aan de hand van een kleinere set coëfficiënten.

Gevolg 2.12. Schrijf $f|_N$ voor de reconstructie met de eerste N basisfuncties. Dan hebben we de relatie

$$||f - f|_N||^2_{L_2([a,b])} = \sum_{k=1}^{\infty} |\langle f - f|_N, \phi_k \rangle|^2 = \sum_{k=1}^{\infty} |\langle f, \phi_k \rangle - \langle f|_N, \phi_k \rangle|^2 = \sum_{k=N+1}^{\infty} |\langle f, \phi_k \rangle|^2,$$

vanwege de Parsevalgelijkheid (1.1). Door stelling 2.11 zijn er k_0 en c zodat $|\langle f, \phi_k \rangle| < c \cdot k^{-n}$ voor $k > k_0$. Dus

$$\sum_{k=N+1}^{\infty} |\langle f, \phi_k \rangle|^2 \leq \sum_{k=N+1}^{\infty} c \cdot k^{-2n} < c \cdot \int_N^{\infty} x^{-2n} dx = c \cdot \left[\frac{x^{1-2n}}{1-2n} \right]_N^{\infty} = c \cdot \frac{N^{1-2n}}{2n-1}.$$

Dan

$$N^{2(n-1)} \cdot \sum_{N+1}^{\infty} |\langle f, \phi_k \rangle|^2 < N^{2(n-1)} \cdot c \cdot \frac{N^{1-2n}}{2n-1} = \frac{c}{2n-1} N^{-1} \to 0 \quad \text{als } N \to \infty,$$

wat precies betekent dat

$$\|f - f|_N\|^2_{L_2([a,b])} \in o\left(N^{-2(n-1)}\right) \implies ||f - f|_N||_{L_2([a,b])} \in o\left(N^{-(n-1)}\right).$$

De fout in het discrete geval

Wanneer we een discrete functie bekijken zullen we het criterium van *gladheid* niet kunnen gebruiken. We beroepen ons daarom op de analogie tussen het discrete en non-discrete geval.

De gladheid van een discrete functie $f : A \to \mathbb{R}$ wordt bepaald door de verschillen $f[x] - f[x+1]$, wanneer de verschillen klein zijn en de verzameling A groot is komt dit overeen met de afgeleide van een continue functie en zeggen we dat de functie glad is. Zo kunnen we analoga geven voor hogere afgeleiden.

We zullen in onze resultaten de discrete fout in kaart brengen door de zogenaamde PSNR-functie te gebruiken, een logaritmische schaal die werkt over discrete signalen (cf. §1.2) welke ook als elementen uit ℓ_2 kunnen worden beschouwd. Zie hiervoor hoofdstuk 5.

3. Wavelets

De Fouriertransformatie bestaat al honderden jaren en is een grote speler geworden in de *signal processing*. Een groot nadeel van deze transformatie is dat zij slecht reageert op discontinue signalen door de globale dragers van de basisfuncties. Hierdoor worden alle Fouriercoëfficiënten beïnvloed door een discontinuïteit. In de laatste dertig jaar is daarom een nieuwe transformatie ontwikkeld die beter blijkt om te gaan met discontinuïteiten. We hebben het hier over de *Wavelettransformatie*.

Definitie 3.1. Een wavelet is simpelweg een functie $\psi : \mathbb{R} \to \mathbb{R}$ die voldoet aan

$$\int_{-\infty}^{\infty} \psi(t)dt = 0.$$

Met deze functie ψ kunnen we een familie functies $\psi_{u,s}$ bouwen door middel van schaling en translatie:

$$\psi_{u,s}(t) := \sqrt{s}\psi\left(s \cdot t - u\right).$$

Deze familie geeft aanleiding tot een Wavelettransformatie W_f van f in (u, s):

$$W_f(u, s) = \int_{-\infty}^{\infty} f(t)\psi^*_{u,s}(t)dt.$$

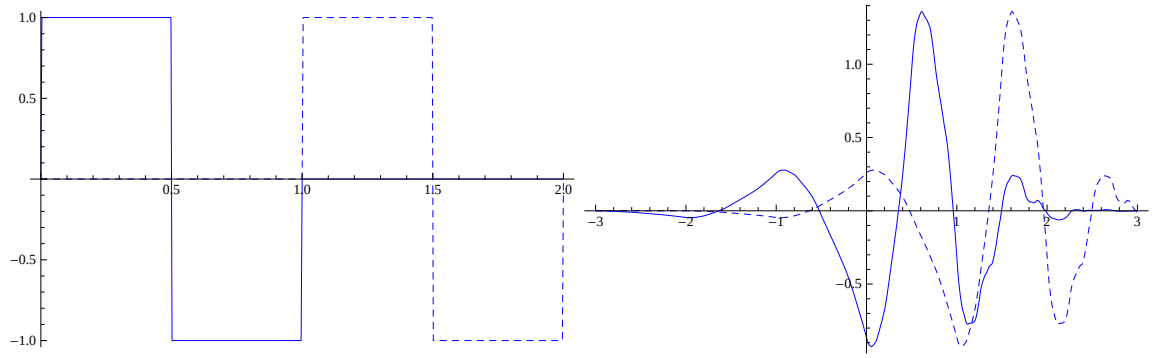
Het is verder mogelijk om wavelets te construeren die met deze schaling en translatie een basis voor de $L_2(\mathbb{R})$ vormen. Over het algemeen kijken we dan naar

$$\left\{\psi_{j,n}(t) = \sqrt{2^j}\psi\left(2^j t - n\right) : (j, n) \in \mathbb{Z}^2\right\}.$$

De kunst is nu om de basiselementen loodrecht op elkaar te laten staan, zodat er een orthogonale (en met de juiste factoren zelfs een orthonormale) basis gevormd wordt. Zie figuur 3.1 voor twee wavelets.

Het gedrag bij discontinuiteiten van de Fouriertransformatie maakt compressie van signalen met harde randen moeilijk. Veel wavelets worden daarom zó geconstrueerd dat dit probleem (deels) verholpen wordt. We zijn namelijk op zoek naar een wavelet die een eindige drager heeft. Het blijkt dat deze wavelets bestaan en dat er zelfs een hele grote verzameling van zulke wavelets is, elk met eigen gewilde eigenschappen.

Omdat wij naar de toepassing van wavelets binnen de beeldcompressie bekijken, zijn we natuurlijk vooral geïnteresseerd in discrete toepassingen. We kijken dus naar een benadering



Figuur 3.1.: Links: De Haarwavelet. Rechts: De Daubechies-2-wavelet. De gestippelde grafiek is een translatie naar rechts en staat in beide gevallen loodrecht op de continue lijn die de waveletfunctie voorstelt.

van een functie f. Hiervoor gebruiken we een rij geneste ruimtes die uiteindelijk naar de $L_2(\mathbb{R})$ toe gaat:

$$V_0 \subset V_1 \subset \ldots \subset L_2(\mathbb{R}). \tag{3.1}$$

We noemen deze rij een multiresolutie.

Definitie 3.2. Een rij geneste ruimtes $\{V_j : j \in \mathbb{N}_0\}$ zoals in (3.1) heet een multiresolutie wanneer voldaan wordt aan de volgende eigenschappen:

$$\forall j,k : f(t) \in V_j \implies f(t - 2^j k) \in V_j, \tag{3.2}$$

$$\forall j : V_{j-1} \subset V_j, \tag{3.3}$$

$$\forall j : f(t) \in V_j \iff f(t/2) \in V_{j-1}, \tag{3.4}$$

$$\bigcup_{j=0}^{\infty} V_j = \lim_{j\to\infty} V_j = L_2(\mathbb{R}), \tag{3.5}$$

$$\text{Er is } \phi : \mathbb{R} \to \mathbb{R} \text{ zo dat } \{\phi(t-n) : n \in \mathbb{Z}\} \text{ een orthonormale basis voor } V_0 \text{ is.} \tag{3.6}$$

Voorbeeld. We bekijken een multiresolutie van ruimtes over stuksgewijs constante functies, de ruimte V_j die we hiervoor bekijken is:

$$V_j = \left\{ g(t) \in L_2(\mathbb{R}) \,\middle|\, g(t) \text{ constant voor } t \in [n2^{-j}, (n+1)2^{-j}) \right\} \text{ met } n \in \mathbb{Z}.$$

De basisfunctie ϕ voor V_0 wordt in dit geval $\phi(t) = 1_{[0,1)}(t)$.

3.1. Schalingsfuncties

Wanneer we nu een orthonormale basis hebben voor V_0, willen we graag een orthonormale basis voor V_j construeren.

Stelling 3.3 ([6, T7.1])**.** *Laat $\{V_j\}$ een multiresolutie en laat $\{\phi(t-n)\}$ de basis voor V_0. Laat verder*

$$\phi_{j,n}(t) := \sqrt{2^j}\phi\left(t2^j - n\right).$$

Dan is $\{\phi_{j,n} : n \in \mathbb{Z}\}$ een orthonormale basis voor V_j.

De functie ϕ heet ook wel de *schalingsfunctie*.

Benadering

De orthogonale projectie van f op V_j is, zoals we weten, de beste benadering van f in V_j. Deze is te vinden door

$$P_{V_j} f = \sum_{n=-\infty}^{\infty} \langle f, \phi_{j,n} \rangle \phi_{j,n}.$$

De coëfficiënten $a_j[n] = \langle f, \phi_{j,n} \rangle$ geven ons op deze manier een gedeeltelijke benadering van f op resolutie 2^j.

3.2. Filters

Wanneer we een schalingsfunctie ϕ definiëren (en dus een V_0), dan wordt V_1 hierdoor al bijna beschreven. We zullen daarom deze schalingsfunctie nader onderzoeken.

Per definitie van de multiresolutie weten we dat $V_{j-1} \subset V_j$. In het bijzonder geldt dat $\phi(t) \in V_0 \subset V_1$ en omdat $\{\sqrt{2}\phi(2t-n) : n \in \mathbb{Z}\}$ een orthonormale basis voor V_1 is, kunnen we $\phi(t)$ schrijven als

$$\phi(t) = \sum_{n=-\infty}^{\infty} \left\langle \sqrt{2}\phi\left(2t-n\right), \phi(t) \right\rangle \sqrt{2}\phi(2t-n). \tag{3.7}$$

Definitie 3.4. Deze inproducten hebben een speciale naam, want de rij $\{h[n] : n \in \mathbb{Z}\}$ met

$$h[n] := \left\langle \sqrt{2}\phi\left(2t-n\right), \phi(t) \right\rangle$$

wordt ook wel de *filter* van ϕ genoemd.

Stelling 3.5 ([6, T7.2])**.** *Laat $\phi \in L_2(\mathbb{R})$ een schalingsfunctie die ook integreerbaar is. Dan ligt de multiresolutie vast.*

Andersom, als $h[n]$ een filter is zodat de Discrete-Tijd Fouriergetransformeerde[1] *$\hat{h}(\omega)$ periodiek 2π is en continu differentieerbaar in een omgeving van $\omega = 0$ en als daarnaast geldt*

$$\begin{aligned} \forall \omega \in \mathbb{R} : |\hat{h}(\omega)|^2 + |\hat{h}(\omega+\pi)|^2 &= 2, \\ \hat{h}(0) &= \sqrt{2}, \\ \inf_{\omega \in [-\pi/2, \pi, 2]} |\hat{h}(\omega)| &> 0, \end{aligned}$$

[1] De DTFT van h is precies de DFT, met als enige verschil dat $\hat{h}$ een reële waarde als argument krijgt:

$$\hat{h}(\omega) = \sum_{n=-\infty}^{\infty} h[n] e^{-in\omega}$$

.

dan is de functie ϕ waarvan de Fouriergetransformeerde voldoet aan

$$\hat{\phi}(\omega) = \prod_{p=1}^{\infty} \frac{\hat{h}(2^{-p}\omega)}{\sqrt{2}}$$

een schalingsfunctie in $L_2(\mathbb{R})$.

We zullen verder niet ingaan op deze stelling maar enkel de gevolgen gebruiken, namelijk dat de multiresolutie vast ligt met een goede keuze van ϕ en dat voor een goed gekozen $h[n]$, ϕ ook vast ligt.

Voorbeeld. Bekijk weer het geval $\phi(t) = 1_{[0,1)}(t)$. Dan vinden we dat

$$h[n] = \left\langle \sqrt{2}\phi\left(2t-n\right), \phi(t) \right\rangle = \begin{cases} \frac{1}{\sqrt{2}} & \text{als } n \in \{0,1\} \\ 0 & \text{anders.} \end{cases}$$

3.3. Terugkeer van de wavelet

We weten dat V_{j-1} bevat is in V_j. Laat nu W_{j-1} het orthogonale complement van V_{j-1} in V_j, dan spannnen zij samen V_j op:

$$V_j = V_{j-1} \oplus W_{j-1} \tag{3.8}$$

De projectie van f op V_j kan dus geschreven worden als som van projecties:

$$P_{V_j} f = P_{V_{j-1}} f + P_{W_{j-1}} f. \tag{3.9}$$

Omdat $V_{j-1} \subset V_j$ is alle informatie over f die beschikbaar is in V_{j-1}, ook beschikbaar in V_j. We missen echter nog wat informatie; de 'details' die zichtbaar zijn in $P_{W_{j-1}} f$.

Het kan bewezen worden [6, T7.3] dat, voor een schalingsfunctie ϕ (en daarmee een filter h) er een wavelet ψ bestaat zo dat

$$\left\{ \psi_{j,n}(t) := \sqrt{2^j}\psi\left(2^j t - n\right) : n \in \mathbb{Z} \right\}$$

een orthonormale basis is voor W_j. Deze functie is dan een *orthogonale* wavelet, omdat $W_{j-1} \perp V_{j-1}$. We zouden nu de verzameling $\{\psi_{j,n} : (j,n) \in \mathbb{Z}^2\}$ kunnen gebruiken als basis voor $L_2(\mathbb{R})$; dit is hij. In de praktijk hebben we echter liever dat de recursieve relatie in (3.8) ophoudt bij een bepaalde V_j, daarom formuleren we een andere basis.

Lemma 3.6. *De verzameling $\{\phi_{0,n} : n \in \mathbb{Z}\} \cup \{\psi_{j,n} : j \in \mathbb{N}_0, n \in \mathbb{Z}\}$ is een basis voor $L_2(\mathbb{R})$.*

Bewijs. We bekijken de definitie van de multiresolutie, die zegt dat

$$L_2(\mathbb{R}) = \lim_{j \to \infty} V_j.$$

Verder weten we dat

$$V_j = V_k \oplus W_k \oplus \cdots \oplus W_{j-1} = \ldots = V_0 \oplus W_0 \oplus \cdots \oplus W_{j-1}. \tag{3.10}$$

Combineer deze twee gelijkheden om te vinden dat

$$L_2(\mathbb{R}) = \lim_{j\to\infty} V_j = V_0 \oplus \left(\bigoplus_{j=0}^{\infty} W_j \right).$$

Nu moet wel gelden dat $\{\phi_{0,n} : n \in \mathbb{Z}\} \cup \{\psi_{j,n} : j \in \mathbb{N}_0, n \in \mathbb{Z}\}$ een basis voor $L_2(\mathbb{R})$ is. □

Vanwege de relatie $W_{j-1} \subset V_j$, waaruit volgt dat $\psi(t) \in W_0 \subset V_1$, kunnen we $\psi(t)$ schrijven in termen van $\{\sqrt{2}\phi(2t-n) : n \in \mathbb{Z}\}$, de orthonormale basis van V_1:

$$\psi(t) = \sum_{n=-\infty}^{\infty} \left\langle \psi(t), \sqrt{2}\phi(2t-n) \right\rangle \sqrt{2}\phi(2t-n).$$

Ook deze inproducten hebben een speciale naam: de rij $g[n]$ met

$$g[n] := \left\langle \psi(t), \sqrt{2}\phi(2t-n) \right\rangle$$

wordt ook wel de filter van ψ genoemd. De twee filters zijn gerelateerd aan elkaar volgens de vergelijking [3, V13][8, P958]

$$g[n] = (-1)^n h[1-n]. \tag{3.11}$$

Met een filter h (die voldoet aan bepaalde eigenschappen: zie stelling 3.5) worden dus een schalingsfunctie ϕ en een filter g met waveletfunctie ψ geconstrueerd.

Voorbeeld. We keren nog een laatste keer terug naar het voorbeeld waarin $\phi(t) = 1_{[0,1)}$. We vinden met de gelijkheden uit voorgaande paragrafen dat

$$\psi(t) = \sum_{n=-\infty}^{\infty} (-1)^n h[1-n]\sqrt{2}\phi(2t-n),$$

en omdat $h[0] = h[1] = \frac{1}{\sqrt{2}}$, $h[n] = 0$ voor $n \in \mathbb{Z} \setminus \{0,1\}$, zoals we eerder vonden. Dit herschrijft daarom tot

$$\psi(t) = \sqrt{2}\left(\phi(2t) - \phi(2t-1)\right)$$

met als gevolg dat

$$\psi(t) = \begin{cases} 1 & \text{als } t \in [0, 1/2) \\ -1 & \text{als } t \in [1/2, 1) \\ 0 & \text{anders.} \end{cases}$$

Deze wavelet ψ wordt ook wel de Haarwavelet genoemd en is uitgevonden door Alfred Haar in 1909, hoewel het onderzoeksgebied van de wavelets toen nog niet bestond. In het vervolg zullen we verdere aandacht aan deze wavelet besteden.

3.4. Het kiezen van een wavelet

Bij het kiezen of vinden van een wavelet is men over het algemeen op zoek naar bepaalde eigenschappen. Voor compressie zijn we op zoek naar een wavelet die een klein aantal grote coëfficiënten en een groot aantal kleine teweeg brengt: een soort concentratie van de belangrijke informatie. Dit wordt vooral bepaald door drie factoren: gladheid van f (waar we niks aan kunnen doen), de grootte van de drager (welke hierna aan bod komt) en de zogenaamde orde van de wavelet.

Definitie 3.7. Wanneer de waveletfunctie loodrecht staat op alle polynomen van graad $p-1$ of lager ($\langle \psi, q \rangle = 0$), spreken we van een wavelet van orde p. Dit komt overeen met de uitspraak dat

$$\int_{-\infty}^{\infty} x^k \psi(x) dx = 0 \text{ voor } k \in \{0, \dots p-1\}.$$

Gevolg van deze eigenschap is dat we van de functie f elk polynoom van graad $p-1$ af mogen trekken zonder een verschil in inproduct:

$$\langle f, \psi_{j,n} \rangle = \langle f - q, \psi_{j,n} \rangle \text{ voor } q \text{ een polynoom van graad } p-1.$$

Intuïtief is deze eigenschap gewild: we willen immers een heel stel vrijheidsgraden. We zullen dit argument in een volgende sectie formaliseren.

Een andere eigenschap waar we eerder al naar verlangden, is om een wavelet te vinden met eindige drager, zodat discontinuïteiten alleen lokaal zichtbaar zijn. We zullen hiervoor de dragers van $h[n], \psi$ en ϕ aan elkaar relateren.

Compacte drager

Hoewel ϕ een functie uit L_2 is, en h een functie uit ℓ_2, is het toch mogelijk het begrip drager in beide ruimtes te beschrijven alsof ze hetzelfde zijn. Laat daartoe h' de stuksgewijs constante functie van $\mathbb{R}$ naar $\mathbb{R}$ die x stuurt naar $h\left[\lfloor x \rfloor\right]$. Dan is de drager van h gedefiniëerd als de drager van h'.

Stelling 3.8 ([6, P7.2])**.** *De volgende relaties gelden voor de dragers.*

1. *De schalingsfunctie ϕ heeft een compacte drager dan en slechts dan als het filter $h[n]$ een compacte drager heeft, en deze zijn hetzelfde.*

2. *Als de drager van ϕ gelijk is aan $[N_1, N_2]$ dan is de drager van ψ gelijk aan $[(N_1 - N_2 + 1)/2, (N_2 - N_1 + 1)/2]$.*

Bewijs 1. Als ϕ een compacte drager heeft dan $h[n]$ ook: we weten dat

$$h[n] = \left\langle \sqrt{2}\phi\left(2t - n\right), \phi(t) \right\rangle,$$

dus er kunnen maar eindig veel n ongelijk nul zijn. De omgekeerde bewering staat bewezen in [8, P965-967].

Om deze dragers gelijk te krijgen: stel dat de drager van $h[n]$ gelijk $[N_1, N_2]$ is en die van ϕ is $[K_1, K_2]$. Via (3.7) is de drager van de rechterkant gelijk aan $[N_1/2 + K_1/2, N_2/2 + K_2/2]$. We concluderen dat $K_1 = N_1$ en $K_2 = N_2$. □

Bewijs 2. Kijk nu naar

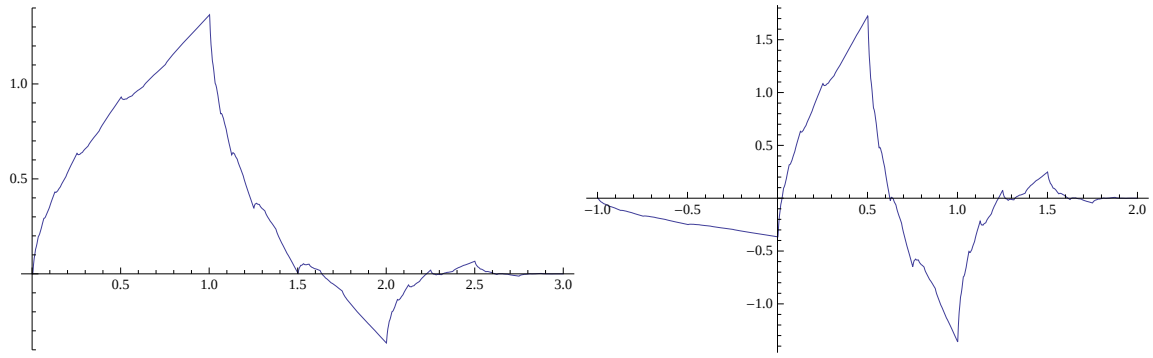
$$\psi(t) = \sum_{n=-\infty}^{\infty} g[n]\phi(2t-n) = \sum_{n=-\infty}^{\infty} (-1)^n h[1-n]\phi(2t-n).$$

Met de informatie uit het begin van de stelling kunnen we de drager van de rechterkant vinden: $[N_1/2 - N_2/2 + 1/2, N_2/2 - N_1/2 + 1/2]$. Het linkerlid heeft dezelfde drager en die moet zo wel gelijk zijn aan $[(N_1 - N_2 + 1)/2, (N_2 - N_1 + 1)/2]$. □

Daubechieswavelets

Hoewel de constructie van de Daubechieswavelet buiten het spectrum van dit artikel valt,[2] willen we toch een kort licht schijnen op deze speciale familie van wavelets. Deze worden gemaakt met de noties van eerder, namelijk dat we de drager willen minimaliseren maar de orde maximaliseren. Daubechies heeft bewezen[8] dat een filter h met orde p, minimaal een drager van lengte $2p$ moet hebben. De zogenaamde Daubechieswavelet van orde p heeft precies een filter van lengte $2p$. In het bijzonder is de Haarwavelet de eerste in de familie van Daubechieswavelets.

Wij hebben in het praktische deel van ons project aandacht besteed aan de zogenaamde Daubechies-2 wavelet die haar naam ontleent aan het feit dat zij van orde 2 is.



Figuur 3.2.: Links: de Daubechies-2 schalingsfunctie. Rechts: de Daubechies-2 waveletfunctie.

[2] Voor een goede beschrijving van deze constructie, zie [6] of [8].

3.5. Fast Wavelet Transform

Door de recursieve relatie van de ruimtes in (3.8) herhaald toe te passen, kunnen we de ruimte V_j schrijven in termen van V_0 en een scala aan orthogonaal-complementsruimtes W_k volgens (3.10).

We willen een functie f benaderen in de ruimte V_j door $P_{V_j}f$ te schrijven in de basis van de ruimtes V_0 en de verschillende ruimtes W_k. Hiervoor dienen we de inproducten te berekenen van f met de basisvectoren in deze ruimtes. We onderscheiden dan de *approximatie*-coëfficiënten $a_j[n]$ en de *detail*-coëfficiënten $d_j[n]$, die f geven als projectie op respectievelijk V_j en W_j.

Het is echter veel rekenwerk om al deze coëfficiënten uit te rekenen. Dit gebeurt immers door het berekenen van integralen. We beperken ons daarom tot het uit rekenen van de coëfficiënten op het niveau j en proberen vervolgens een recursieve relatie te vinden om hieruit de approximatie- en detailcoëfficiënten van het volgende niveau te vinden.

Vanwege de relatie in (3.8) kunnen we de basisfuncties uit de ruimtes W_{j-1}, V_{j-1} schrijven in termen van de basisfuncties van de ruimte V_j. We schrijven daarvoor $\phi_{j-1,n}$ om in termen van de basisfuncties $\phi_{j,k}$ en doen dit ook voor $\psi_{j-1,n}$:

$$\phi_{j-1,n} = \sum_{k=-\infty}^{\infty} \langle \phi_{j-1,n}, \phi_{j,k} \rangle \phi_{j,k}, \tag{3.12}$$

$$\psi_{j-1,n} = \sum_{k=-\infty}^{\infty} \langle \psi_{j-1,n}, \phi_{j,k} \rangle \phi_{j,k}. \tag{3.13}$$

We rekenen vervolgens de inproducten uit in termen van filtercoëfficiënten, bekijk:

$$\begin{aligned}
\langle \phi_{j-1,n}, \phi_{j,k} \rangle &= \textstyle\int_{-\infty}^{\infty} \sqrt{2^{j-1}} \phi(2^{j-1}t - n) \sqrt{2^j} \phi^\star(2^j t - k) \mathrm{d}t \\
&= \textstyle\int_{-\infty}^{\infty} \frac{1}{2^{j-1}} 2^{j-1} \sqrt{2} \phi(t') \phi^\star(2t' - k + 2n) \mathrm{d}t' \\
&= \langle \phi(t), \sqrt{2}\phi(2t - k + 2n) \rangle \\
&= h[k - 2n],
\end{aligned}$$

$$\begin{aligned}
\langle \psi_{j-1,n}, \phi_{j,k} \rangle &= \textstyle\int_{-\infty}^{\infty} \sqrt{2^{j-1}} \psi(2^{j-1}t - n) \sqrt{2^j} \phi^\star(2^j t - k) \mathrm{d}t \\
&= \textstyle\int_{-\infty}^{\infty} \frac{1}{2^{j-1}} 2^{j-1} \sqrt{2} \psi(t') \phi^\star(2t' - k + 2n) \mathrm{d}t' \\
&= \langle \psi(t), \sqrt{2}\phi(2t - k + 2n) \rangle \\
&= g[k - 2n],
\end{aligned}$$

waarbij we de coördinaattransformatie $t' = 2^{j-1}t - n$ hebben gebruikt. Door vervolgens aan beide zijden van vergelijkingen (3.12), (3.13) het inproduct met f te nemen kunnen we de coëfficiënten van de resolutie 2^{j-1} schrijven in termen van de coëfficiënten op de resolutie 2^j:

$$a_{j-1}[n] = \langle f, \phi_{j-1,n} \rangle = \sum_{k=-\infty}^{\infty} h[k - 2n] \langle f, \phi_{j,k} \rangle = \sum_{k=-\infty}^{\infty} h[k - 2n] a_j[k] = (a_j \star \bar{h})[2n], \tag{3.14}$$

$$d_{j-1}[n] = \langle f, \psi_{j-1,n}\rangle = \sum_{k=-\infty}^{\infty} g[k-2n]\langle f, \phi_{j,k}\rangle = \sum_{k=-\infty}^{\infty} g[k-2n]a_j[k] = (a_j \star \bar{g})[2n], \quad (3.15)$$

waarbij $\bar{f} : x \mapsto f(-x)$. De relatie die hier gevonden is geeft aanleiding tot een algoritme.

Algoritme (Fast Wavelet Transform)**.** *Gegeven een rij coëfficiënten $a_j \in \mathbb{R}^{2^j}$ definiëeren we een recursief algoritme* $\mathrm{FWT} : \mathbb{R}^{2^j} \to \mathbb{R}^{2^j}$ *met een gevalsonderscheid over j.*
Als $j = 0$ dan geldt

$$\mathrm{FWT}(a_j)[n] = a_j[n].$$

Als $j > 0$ bereken a_{j-1} en d_{j-1} volgens (3.14), (3.15)*. Hiermee berekenen we de* FWT *als:*

$$\mathrm{FWT}(a_j)[n] = \begin{cases} \mathrm{FWT}(a_{j-1})[n] & \text{als } n \leq 2^{j-1} \\ d_{j-1}[n] & \text{als } n > 2^{j-1} \end{cases}. \quad (3.16)$$

We gaan uit van een eindige filter en beschouwen V_j als een ruimte van functies op een interval in $\mathbb{R}$, dan versimpelen de oneindige sommaties in (3.14), (3.15) tot een eindige som.

We kunnen nu de transformatie inverteren aan de hand van het volgende algoritme

Algoritme (Inverse Fast Wavelet Transform)**.** *Gegeven een rij coëfficiënten $x_j \in \mathbb{R}^{2^j}$ definieëren we een recursief algoritme* $\mathrm{iFWT} : \mathbb{R}^{2^j} \to \mathbb{R}^{2^j}$ *met een gevalsonderscheid in j.*
Als $j = 0$ dan geldt

$$\mathrm{iFWT}(x_j)[n] = x_j[n]$$

Als $j > 0$, laat $x_{j-1}[n] = x_j[n]$ en $d_{j-1}[n] = x_j[n + 2^{j-1}]$ voor $1 \leq n \leq 2^{j-1}$. Bereken hiermee $a_{j-1} = iFWT(x_{j-1})$, dan geldt

$$\mathrm{iFWT}(x_j)[n] = (\breve{a}_{j-1} \star h)[n] + (\breve{d}_{j-1} \star g)[n], \quad (3.17)$$

waarbij $\breve{y}[2n] = y[n]$ en $\breve{y}[2n+1] = 0$.

Stelling 3.9. *De iFWT (links) samengesteld met de FWT geeft de identiteit.*

Bewijs. We bekijken de verschillende gevallen. Voor het geval $j = 0$ geldt duidelijk dat

$$\mathrm{iFWT}(\mathrm{FWT}(a_0)) = \mathrm{iFWT}(a_0) = a_0.$$

We zullen dus verder moeten bewijzen dat (3.17) een inverse vormt voor (3.16). Schrijf hiervoor de basisfuncties van V_j in termen van de basisfuncties van V_{j-1} en W_{j-1}, ofwel:

$$\phi_{j,n} = \sum_{k=-\infty}^{\infty} \langle \phi_{j,n}, \phi_{j-1,k}\rangle \phi_{j-1,k} + \sum_{-\infty}^{\infty} \langle \phi_{j,n}, \psi_{j-1,k}\rangle \psi_{j-1,k}.$$

Deze inproducten komen weer overeen met de filtercoëfficiënten. Nemen we dus aan beide zijden het inproduct dan volgt

$$a_j[n] = \sum_{k=-\infty}^{\infty} h[n-2k]a_{j-1}[k] + \sum_{-\infty}^{\infty} g[n-2k]d_{j-1}[k].$$

Door in de sommatie de variabele k te vervangen door $k' = 2k$ vereenvoudigt dit tot

$$a_j[n] = \sum_{k'=-\infty}^{\infty} h[n-k']\breve{a}_{j-1}[k'] + \sum_{-\infty}^{\infty} g[n-k']\breve{d}_{j-1}[k'].$$

Daarmee geeft de iFWT een inverse voor de FWT. □

3.6. Analyse van de Wavelettransformatie

Met de theoretische beschouwing van wavelets en de Fast Wavelet Transform achter de rug, zullen we kijken naar practische obstakels.

Eindige signalen

Een van de eerste aannames die we tot nu toe steeds maakten is dat de signalen oneindig lang zijn. Wanneer echter de functie f een compacte drager heeft, worden een aantal zaken wat lastiger. Neem als eerste aan dat de drager van f binnen $[0,1]$ ligt. [3] In dit geval zou het kunnen dat de waveletfuncties met een drager die $t = 0$ of $t = 1$ doorsnijden, niet meer de gewenste eigenschappen hebeben. Er zijn in de literatuur oplossingen voor dit probleem gevonden maar hier zullen wij verder niet op in gaan.

Verder namen we ook aan dat de signalen non-discreet zijn. In de praktijk zullen we discrete functies bekijken die bijvoorbeeld op de resolutie 2^j in elk interval een waarde aannemen. We bekijken de multiresolutie

$$V_0 \subset V_1 \subset \ldots \subset V_j.$$

We weten nu dat er in V_j een functie h bevat ligt zodat $\langle h, \phi_{j,k}\rangle = f[k]$, deze komt overeen met onze f. Dit discrete signaal kan dus perfect geschreven worden in de waveletbasis op resolutie 2^j. Om deze reden wordt de Fast Wavelet Transform veel gebruikt bij de analyse van discrete signalen.

Signaaluitbreiding

Een probleem waar we in het geval van eindige signalen nog meer mee te maken krijgen is dat de algoritme niet goed omgaat met de 'randen' van de functie. De convolutie heeft ineens informatie nodig die *buiten het definitiegebied* van het signaal ligt. Eerder in sectie 1.1 hebben we al gezien hoe signalen naar een tweemacht uitgebreid kunnen worden. Precies dezelfde methoden kunnen gebruikt worden om het signaal nog verder uit te breiden.

[3] Door translatie en dilatie kan elk signaal met compacte basis omgevormd worden tot een signaal met drager binnen $[0,1]$. We verliezen hier dus geen algemeenheid.

Om niet te veel tijd te verliezen met het ondersteunen van meerdere mogelijkheden hebben wij er voor gekozen om *periodic padding* op alle signalen toe te passen. Dit omdat de zogenaamde *circulaire convolutie* ingebouwd zit in de bibliotheek die wij gebruikt hebben.

Complexiteit van de algoritme

Als de lengte van de filter h gelijk is aan K, en de lengte van het originele signaal a_L gelijk is aan $N = 2^L$, kunnen we voor $j \in \{0, \ldots, L\}$ zien dat a_j en d_j beide 2^j elementen bevatten. Vervolgens kunnen a_{j-1} en d_{j-1} gemaakt worden door $2^j K$ operaties zodat elke stap van de algoritme $2^j \cdot K$ operaties kost. Daarmee kost het hele algoritme

$$\sum_{j=0}^{L} 2^j \cdot K = K \sum_{j=0}^{L} 2^j = K \cdot (2^{1+L} - 1) \simeq 2 \cdot K 2^L = 2KN$$

operaties. Dus deze DWT is een $\theta(KN)$ algoritme. Ook de complexiteit van de inverse wordt zo van orde KN.

3.7. Meer dimensies: de Mallatdecompositie

Met een familie van orthonormale wavelets $\{\psi_{j,n} : j \in \mathbb{N}_0, n \in \mathbb{Z}\}$ voor de ruimte $L_2(\mathbb{R})$ volgt een natuurlijke voortzetting van de waveletfamilie naar twee dimensies door

$$\{\psi_{j_1,n_1}(x_1)\psi_{j_2,n_2}(x_2) : j_i \in \mathbb{N}_0, n_0 \in \mathbb{Z}\}.$$

We zullen zien dat we samen met een verzameling tweedimensionale schalingsfuncties (analoog aan lemma 3.6) een basis kunnen vormen voor de $L_2(\mathbb{R}^2)$. Merk nu op dat we met zulke wavelets op de x_1-as met resolutie 2^{j_1} kijken terwijl de x_2-as een resolutie 2^{j_2} kent.

Mallat vond dit iets om te vermijden [6, §7.7] en legt in zijn analyse dan ook de eis $j_1 = j_2 =: j$ op. Wij zullen in het vervolg óók kijken naar het zogenaamde Tensorproduct, wat de eis $j_1 = j_2$ *niet* oplegt, dit volgt in sectie 3.8.

In 1 dimensie hebben de notie van 'een resolutie' geformaliseerd in het begrip multiresolutie. De definitie van deze multiresolutie zullen we nu voortzetten naar een meerdimensionaal geval. Wanneer we spreken over een separeerbare multiresolutie, bedoelen we een ruimte $V_j^{(2)} := V_j \otimes V_j$, waarbij V_j tot een één-dimensionale multiresolutie behoort. In [6, A.5] wordt bewezen dat, gegeven een orthonormale basis $\{\phi_{j,m} : m \in \mathbb{Z}\}$ voor V_j, de verzameling

$$\{\phi^{(2)}_{j,n=(n_1,n_2)} := \phi_{j,n_1} \otimes \phi_{j,n_2} : n \in \mathbb{Z}^2\} \tag{3.18}$$

een orthonormale basis voor $V_j^{(2)}$ is.

Voorbeeld. Bekijk weer V_j, de ruimte van stuksgewijs constante functies op het interval

$$[2^{-j}m, 2^{-j}(m+1)) \quad m \in \mathbb{Z}.$$

We vinden voor $V_j^{(2)}$ de ruimte van stuksgewijs constante functies op vierkanten van de vorm $[2^{-j}n_1, 2^{-j}(n_1+1)) \times [2^{-j}n_2, 2^{-j}(n_2+1))$. De tweedimensionale schalingsfunctie wordt nu:

$$\phi^{(2)}(x_1,x_2) = \phi(x_1)\phi(x_2) = \begin{cases} 1 & \text{als } x_1 \in [0,1) \text{ en } x_2 \in [0,1) \\ 0 & \text{anders.} \end{cases}.$$

Tweedimensionale Waveletfuncties

Voor de tweedimensionale ruimtes $V_{j+1}^{(2)}$ die we gevonden hebben willen we nu weer een recursieve relatie vinden. We weten dat $V_j^{(2)}$ bevat is in $V_{j+1}^{(2)}$, bekijk daarom het orthogonale complement $\boldsymbol{U}_j \perp V_j^{(2)}$:

$$V_{j+1}^{(2)} = \boldsymbol{U}_j \oplus V_j^{(2)} \tag{3.19}$$

Om een orthogonale waveletbasis voor $\boldsymbol{U}_j$ te vinden, gaan we als volgt te werk. Gegeven een waveletbasis voor V_j geinduceerd door de functies ϕ en ψ, maken we tweedimensionale wavelets in de ruimte $V_j^{(2)}$ volgens:

$$\psi^1(x_1,x_2) = \phi(x_1)\psi(x_2) \quad \psi^2(x_1,x_2) = \psi(x_1)\phi(x_2) \quad \psi^3(x_1,x_2) = \psi(x_1)\psi(x_2). \tag{3.20}$$

Deze functies kunnen we vervolgens schuiven en dilateren. Hiermee verkrijgen we

$$\psi^k_{j,n=(n_1,n_2)}(x_1,x_2) = 2^j\psi^k\left(2^jx_1 - n_1, 2^jx_2 - n_2\right) \tag{3.21}$$

en we zullen bewijzen dat deze functies tezamen met $\phi^{(2)}_{0,n}$ een basis vormen voor $L_2(\mathbb{R})$

Stelling 3.10 ([6, T7.24])**.** *Laat $\phi^{(2)}$ zoals in* (3.18)*, de ψ^k's zoals in* (3.21) *gedefinieerd. Dan kunnen we een basis $\boldsymbol{\Psi}_j$ maken voor $\boldsymbol{U}_j$ volgens:*

$$\boldsymbol{\Psi}_j := \{\psi^1_{j,n}, \psi^2_{j,n}, \psi^3_{j,n} : n \in \mathbb{Z}^2\}.$$

Het samennemen van deze bases met een basis voor $V_0^{(2)}$ geeft bovendien

$$\boldsymbol{\Phi} := \{\phi^{(2)}_{0,n} : n \in \mathbb{Z}^2\} \cup \{\psi^1_{j,n}, \psi^2_{j,n}, \psi^3_{j,n} : n \in \mathbb{Z}^2, j \in \mathbb{N}_0\},$$

wat een basis vormt voor $L_2(\mathbb{R}^2)$.

Bewijs. We schrijven de ruimte $V_{j+1}^{(2)}$ uit als een tensorproduct van ruimtes

$$V_{j+1}^{(2)} = V_{j+1} \otimes V_{j+1}$$

en splitsen de termen, die allen één-dimensionale ruimtes zijn, op in de volgende niveau's ($V_{j+1} = V_j \oplus W_j$) om te vinden dat:

$$\begin{aligned} V_{j+1} \otimes V_{j+1} &= (V_j \oplus W_j) \otimes (V_j \oplus W_j) \\ &= (V_j \otimes V_j) \oplus (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j). \end{aligned}$$

Gebruik makende van de relatie (3.19) leiden we af dat

$$\boldsymbol{U}_j = (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j).$$

Omdat de ψ^k's allen tensorproducten zijn van functies in de ruimtes V_j en W_j volgt dat $\boldsymbol{\Psi}_j = \{\psi^1_{j,n}, \psi^2_{j,n}, \psi^3_{j,n} \,|\, n \in \mathbb{Z}^2\}$ een basis is voor $\boldsymbol{U}_j$.

Met de basis van $\boldsymbol{U}_j$ kunnen we vervolgens een basis voor $L_2(\mathbb{R}^2)$ vinden. We schrijven daarvoor

$$L_2(\mathbb{R}^2) = \lim_{j\to\infty} V^{(2)}_j = \lim_{j\to\infty} (V^{(2)}_0 \oplus \boldsymbol{U}_0 \oplus \cdots \oplus \boldsymbol{U}_{j-1}) = V^{(2)}_0 \oplus \left(\bigoplus_{j=0}^{\infty} \boldsymbol{U}_j \right),$$

zodat $\boldsymbol{\Phi} = \{\phi^{(2)}_{0,n} : n \in \mathbb{Z}^2\} \cup \{\psi^1_{j,n}, \psi^2_{j,n}, \psi^3_{j,n} : n \in \mathbb{Z}^2, j \in \mathbb{N}_0\}$ een basis is voor $L_2(\mathbb{R}^2)$. □

Gevolg 3.11. We kunnen de laatste paar regels van het bewijs ook gedeeltelijk toepassen door te stoppen bij een ruimte V_j. We vinden dan dat:

$$\left\{\phi^{(2)}_{0,n} : n \in \mathbb{Z}^2\right\} \cup \left\{\psi^1_{p,n}, \psi^2_{p,n}, \psi^3_{p,n} : n \in \mathbb{Z}^2, p \in \{0, \ldots, j\}\right\}$$

een orthonormale basis voor $V^{(2)}_j$ is.

Bovenstaande basis bevat dus het tensorproduct van twee schalingsfuncties op één niveau en verder combinaties van schalings- en waveletfuncties van dezelfde resolutie op alle niveaus. Net zoals in het eendimensionale geval kunnen we met deze basis een algoritme formuleren.

Naar een tweedimensionaal algoritme

Nu we volgens (3.19) een relatie hebben gevonden tussen de ruimtes, namelijk

$$V^{(2)}_{j+1} = (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j) \oplus (V_j \otimes V_j), \tag{3.22}$$

kunnen we het eendimensionale algoritme uitbreiden naar twee dimensies. Hiervoor kijken we wederom naar de inproducten van een functie f met onze basisfuncties. We definiëren daarvoor weer de *approximatie*-coëfficiënt en drietallen van *detail*-coëfficiënten volgens:

$$a_j[n] := \langle f, \phi^{(2)}_{j,n}\rangle, \quad d^p_j[n] := \langle f, \psi^p_{j,n}\rangle \text{voor} \quad p \in \{1, 2, 3\}.$$

We zullen in het bijzonder de basis-functies op het niveau j kunnen ontbinden door gebruik te maken van (3.18):

$$\phi^{(2)}_{j,(n_1,n_2)} = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \langle \phi_{j,n_1} \otimes \phi_{j,n_2}, \phi_{j+1,k_1} \otimes \phi_{j+1,k_2}\rangle \phi^{(2)}_{j+1,(k_1,k_2)} \tag{3.23}$$

$$\psi^p_{j,(n_1,n_2)} = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \langle \psi^p_{j,(n_1,n_2)}, \phi_{j+1,k_1} \otimes \phi_{j+1,k_2}\rangle \phi^{(2)}_{j+1,(k_1,k_2)} \tag{3.24}$$

We maken nu gebruik van [10, §3.4] om het inproduct horende bij het tensorproduct van twee *Hilbertruimten* uit te schrijven als:

$$\langle a \otimes b, c \otimes d \rangle = \langle a, c \rangle \cdot \langle b, d \rangle,$$

waarbij de inproducten behoren bij de respectievelijke ruimtes. Dit versimpelt de inproducten die we zoeken volgens:

$$\langle \phi_{j,n_1} \otimes \phi_{j,n_2}, \phi_{j+1,k_1} \otimes \phi_{j+1,k_2} \rangle = \langle \phi_{j,n_1}, \phi_{j+1,k_1} \rangle \langle \phi_{j,n_2}, \phi_{j+1,k_2} \rangle = h[k_1 - 2n_1] \cdot h[k_2 - 2n_2],$$

waarbij we de filterrelatie voor één dimensie toepassen. We kunnen dit nog een stuk verbeteren door het product $h[\_] \cdot h[\_]$ om te schrijven naar een tensorproduct, namelijk:

$$(h \otimes h)(x_1, x_2) = h[x_1] \cdot h[x_2]$$

Deze stappen kunnen met hetzelfde argument toegepast worden op de ψ^k's. We verkrijgen dan met een blik op de definities in (3.20) de vergelijkingen:

$$\langle \psi^1_{j,(n_1,n_2)}, \phi^{(2)}_{j+1,(k_1,k_2)} \rangle = (h \otimes g)[k_1 - 2n_1, k_2 - 2n_2]$$

$$\langle \psi^2_{j,(n_1,n_2)}, \phi^{(2)}_{j+1,(k_1,k_2)} \rangle = (g \otimes h)[k_1 - 2n_1, k_2 - 2n_2]$$

$$\langle \psi^3_{j,(n_1,n_2)}, \phi^{(2)}_{j+1,(k_1,k_2)} \rangle = (g \otimes g)[k_1 - 2n_1, k_2 - 2n_2]$$

Vervolgens richten we ons weer op de approximatie- en detail-coëfficiënten door aan beide zijden van de vergelijkingen (3.23), (3.24) het inproduct met f te nemen. We kunnen dit aan de hand van onze nieuwe 2-dimensionale filters opschrijven met een convolutie in twee dimensies:

$$a_j[n_1, n_2] = (a_{j+1} \star (\bar{h} \otimes \bar{h}))[2n_1, 2n_2] \tag{3.25}$$

$$d^1_j[n_1, n_2] = (a_{j+1} \star (\bar{h} \otimes \bar{g}))[2n_1, 2n_2] \tag{3.26}$$

$$d^2_j[n_1, n_2] = (a_{j+1} \star (\bar{g} \otimes \bar{h}))[2n_1, 2n_2] \tag{3.27}$$

$$d^3_j[n_1, n_2] = (a_{j+1} \star (\bar{g} \otimes \bar{g}))[2n_1, 2n_2]. \tag{3.28}$$

waarbij de tweedimensionale convolutie gegeven wordt door

$$(x \star y)[n_1, n_2] := \sum_{p_1=-\infty}^{\infty} \sum_{p_2=-\infty}^{\infty} x[n_1 - p_1, n_2 - p_2] \cdot y[n_1 - p_1, n_2 - p_2].$$

Algoritme (Tweedimensionale Fast Wavelet Transform)**.** *Gegeven een matrix van coëfficiënten* $a_j \in \mathbb{R}^{2^j} \times \mathbb{R}^{2^j}$*, definieëren we een algoritme* $\mathrm{FWT}_2 : \mathbb{R}^{2^j} \times \mathbb{R}^{2^j} \to \mathbb{R}^{2^j} \times \mathbb{R}^{2^j}$*.*
Als $j = 0$ *dan geldt*

$$\mathrm{FWT}_2(a_j)[n_1, n_2] = a_j[n_1, n_2]$$

Als $j > 0$ *bereken* a_{j-1},d^1_{j-1},d^2_{j-1} *en* d^3_{j-1} *uit* a_j *volgens* ((3.25) − (3.28))*. Dan geldt*

$$\mathrm{FWT}_2(a_j)[n_1, n_2] = \begin{cases} \mathrm{FWT}_2(a_{j-1})[n_1, n_2] & \text{als } n_1 \leq 2^{j-1} \text{ en } n_2 \leq 2^{j-1} \\ d^1_{j-1}[n_1, n_2] & \text{als } n_1 \leq 2^{j-1} \text{ en } n_2 > 2^{j-1} \\ d^2_{j-1}[n_1, n_2] & \text{als } n_1 > 2^{j-1} \text{ en } n_2 \leq 2^{j-1} \\ d^3_{j-1}[n_1, n_2] & \text{als } n_1 > 2^{j-1} \text{ en } n_2 > 2^{j-1} \end{cases} \tag{3.29}$$

Van dit tweedimensionale algoritme kunnen nu ook de inverse algoritme bepalen, analoog aan het eendimensionale geval.

Algoritme (Inverse Tweedimensionale Fast Wavelet Transform)**.** *Gegeven een matrix van coëfficiënten* $x_j \in \mathbb{R}^{2^j} \times \mathbb{R}^{2^j}$*, definiëeren we hierop het algoritme* $\mathrm{iFWT}_2 : \mathbb{R}^{2^j} \times \mathbb{R}^{2^j} \to \mathbb{R}^{2^j} \times \mathbb{R}^{2^j}$*. Als* $j = 0$ *dan geldt*

$$\mathrm{iFWT}_2(x_j)[n_1, n_2] = x_j[n_1, n_2]$$

Als $j > 0$*, splits dan* x_j *in zijn vier kwadranten; laat voor* $k_1, k_2 \in \{1, \ldots, 2^{j-1}\}$

$$\begin{aligned} y_{j-1}[k_1, k_2] &= x_j[k_1, k_2] \\ d^1_{j-1}[k_1, k_2] &= x_j[k_1, k_2 + 2^{j-1}] \\ d^2_{j-1}[k_1, k_2] &= x_j[k_1 + 2^{j-1}, k_2] \\ d^3_{j-1}[k_1, k_2] &= x_j[k_1 + 2^{j-1}, k_2 + 2^{j-1}] \end{aligned}$$

Bereken hiermee vervolgens $a_{j-1} = \mathrm{iFWT}_2(y_{j-1})$*, dan geldt*

$$\begin{aligned} \mathrm{iFWT}_2(a_j)[n_1, n_2] = {} & \breve{a}_{j-1} \star (h \otimes h)[n_1, n_2] + \breve{d}^1_{j-1} \star (h \otimes g)[n_1, n_2] \\ & + \breve{d}^2_{j-1} \star (g \otimes h)[n_1, n_2] + \breve{d}^3_{j-1} \star (g \otimes g)[n_1, n_2] \end{aligned} \tag{3.30}$$

waarbij

$$\breve{y}[n_1, n_2] = \begin{cases} y[n_1/2, n_2/2] & \text{als } 2|n_1 \text{ en } 2|n_2 \\ 0 & \text{anders.} \end{cases}$$

Stelling 3.12. *De* iFWT_2 *(links) samengesteld met de* FWT_2 *geeft de identiteit.*

Bewijs. We bekijken de verschillende gevallen. Voor het geval $j = 0$ geldt duidelijk dat

$$\mathrm{iFWT}_2(\mathrm{FWT}_2(a_0)) = \mathrm{iFWT}_2(a_0) = a_0.$$

Voor het geval $j > 0$ merken we op dat wanneer de iFWT_2 werkt voor $j' < j$ de coëfficiëntenmatrices a, d^1, d^2, d^3 precies zijn wat de FWT_2 zou geven op dit niveau. We zullen dus verder moeten bewijzen dat (3.30) een inverse vormt voor (3.29). Schrijf hiervoor de basisfuncties van $V_j^{(2)}$ in termen van de basisfuncties van de tensorproductenruimte van V_j en W_j zoals in (3.22):

$$\begin{aligned} \phi_{j,(n_1,n_2)} = {} & \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \langle \phi^{(2)}_{j,(n_1,n_2)}, \phi^{(2)}_{j-1,(k_1,k_2)} \rangle \phi^{(2)}_{j-1,(k_1,k_2)} \\ & + \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \langle \phi^{(2)}_{j,(n_1,n_2)}, \psi^1_{j-1,(k_1,k_2)} \rangle \psi^1_{j-1,(k_1,k_2)} \\ & + \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \langle \phi^{(2)}_{j,(n_1,n_2)}, \psi^2_{j-1,(k_1,k_2)} \rangle \psi^2_{j-1,(k_1,k_2)} \\ & + \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \langle \phi^{(2)}_{j,(n_1,n_2)}, \psi^3_{j-1,(k_1,k_2)} \rangle \psi^3_{j-1,(k_1,k_2)}. \end{aligned}$$

Deze inproducten tussen basisfuncties komen weer overeen met de filtercoëfficiënten. Nemen we dus aan beide zeiden het inproduct met f dan volgt de vergelijking voor de approximatiecoëfficiënten

$$\begin{aligned} a_j[n_1,n_2] = &\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}(h\otimes h)[n_1-2k_1,n_2-2k_2]a_{j-1}[k_1,k_2] \\ &+\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}(h\otimes g)[n_1-2k_1,n_2-2k_2]d^1_{j-1}[k_1,k_2] \\ &+\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}(g\otimes h)[n_1-2k_1,n_2-2k_2]d^2_{j-1}[k_1,k_2] \\ &+\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}(g\otimes g)[n_1-2k_1,n_2-2k_2]d^3_{j-1}[k_1,k_2]. \end{aligned}$$

Door in de sommatie de variabelen k_1, k_2 te vervangen door $k'_1 = 2k_1$ respectievelijk $k'_2 = 2k_2$ vereenvoudigt dit tot

$$\begin{aligned} a_j[n_1,n_2] = &\sum_{k'_1=-\infty}^{\infty}\sum_{k'_2=-\infty}^{\infty}(h\otimes h)[n_1-k'_1,n_2-k'_2]\breve{a}_{j-1}[k'_1,k'_2] \\ &+\sum_{k'_1=-\infty}^{\infty}\sum_{k'_2=-\infty}^{\infty}(h\otimes g)[n_1-k'_1,n_2-k'_2]\breve{d}^1_{j-1}[k'_1,k'_2] \\ &+\sum_{k'_1=-\infty}^{\infty}\sum_{k'_2=-\infty}^{\infty}(g\otimes h)[n_1-k'_1,n_2-k'_2]\breve{d}^2_{j-1}[k'_1,k'_2] \\ &+\sum_{k'_1=-\infty}^{\infty}\sum_{k'_2=-\infty}^{\infty}(g\otimes g)[n_1-k'_1,n_2-k'_2]\breve{d}^3_{j-1}[k'_1,k'_2], \end{aligned}$$

wat precies de convolutie in (3.30) is. Daarmee geeft de iFWT een inverse voor de FWT. □

Meer dan twee dimensies

De Mallatdecompositie is op een natuurlijke manier voort te zetten naar meerdimensionale signalen. De precieze definitie is notationeel nogal ingewikkeld en laten we hier daarom achterwege. Informeel geldt dat we een m-dimensionale ruimte $V_j^{(m)}$ weer kunnen schrijven zoals (3.22) waarbij we dit maal het tensorproduct nemen van m factoren, V_{j-1} en W_{j-1}. De basisfuncties van deze ruimtes zijn vervolgens de tensorproducten van m factoren die ofwel ϕ of ψ zijn. Vervolgens kan het algoritme verder doorgezet worden naar de ruimte $V_{j-1}^{(m)}$. In termen van algoritmen vertaalt dit alles naar een m dimensionale convolutie van a_j met filters die het tensorproduct zijn van m factoren h of g.

Voorbeeld. In drie dimensies maken we een Mallat-decompositie van de ruimte $V_{j+1}^{(3)}$ door:

$$\begin{aligned} V_{j+1}^{(3)} = V_j^{(3)} \oplus \left(V_j^{(2)} \otimes W_j\right) \oplus (V_j \otimes W_j \otimes V_j) \oplus \left(W_j \otimes V_j^{(2)}\right) \\ \oplus \left(V_j \otimes W_j^{(2)}\right) \oplus (W_j \otimes V_j \otimes W_j) \oplus \left(W_j^{(2)} \otimes V_j\right) \oplus W_j^{(3)} \end{aligned}$$

De basisfuncties en filters die hier bijhoren zijn analoog aan de bijbehorende ruimtes. Deze filters kunnen we vervolgens toepassen op bijvoorbeeld filmmateriaal.

Eindige signalen in n dimensies

De notie van eindige signalen is al eerder langsgekomen. We bekijken functies met een compacte drager $[0, 1]$. Het gevolg hiervan is dat de complete waveletbasis teveel elementen bevat. Alleen wavelets waarvan de drager het interval doorsnijdt, zijn voor ons interessant. Wanneer we meer dimensies bekijken, praten we over een n-dimensionaal eenheidsinterval $[0, 1]^n =: \square$. Wederom zijn alleen de functies van belang wiens drager $\square$ doorsnijdt.

3.8. Tensorproduct

Herinner dat we voor de Mallat-decompositie gebruik maakte van de gelijkheid:

$$V_j^{(2)} = (V_{j-1} \otimes W_{j-1}) \oplus (W_{j-1} \otimes V_{j-1}) \oplus (W_{j-1} \otimes W_{j-1}) \oplus (V_{j-1}^{(2)})$$

Door dit herhaald toe te passen kregen we een basis van de vorm:

$$\{\phi^{(2)}_{0,(n_1 1, n_2)} \,|\, n_1, n_2 \in \mathbb{Z}\} \cup \{\psi^k_{j,(n_1,n_2)} \,|\, k = 1, 2, 3 \quad j \in \mathbb{N}_0 \quad n_1, n_2 \in \mathbb{Z}\}$$

We zullen echter zien dat er ook een andere manier is om deze ruimte te decomposeren. Bedenk dat we in 1 dimensie de decompositie zoals in (3.10) gebruiken, namelijk:

$$V_j = V_0 \oplus W_0 \oplus \cdots \oplus W_{j-1}$$

Daarmee schrijven we de ruimte $V_j^{(2)}$ als het tensorproduct van V_j met zichzelf:

$$V_j^{(2)} = V_j \otimes V_j = (V_0 \oplus W_0 \oplus \cdots W_{j-1}) \otimes (V_0 \oplus W_0 \oplus \cdots W_{j-1})$$

Dit geeft aanleiding tot een nieuwe basis, namelijk het tensorproduct van de basis van V_j met zichzelf, we duiden deze basis dan ook aan als de *Tensorbasis* [4]. Schrijf namelijk

[4] In de literatuur wordt de Mallatdecompositie ook regelmatig een tensorproduct genoemd. De verwarring ontstaat hier doordat in de Mallatbasis de basisfuncties ook tensorproducten zijn van wavelet- en schalingsfuncties. We doelen echter in onze naamgeving op *het tensorproduct van twee bases.*

$A^2 = \{a \otimes b \,|\, a, b \in A\}$ voor het *Tensorproduct* van een basis met zichzelf, dan krijgen we:

$$\begin{aligned}\boldsymbol{\Phi}_T :=& (\{\phi_{0,n} \,|\, n \in \mathbb{Z}\} \cup \{\psi_{j,n} \,|\, j \in \mathbb{N}_0 \quad n \in \mathbb{Z}\})^2 \\ =& \{\phi_{0,n_1} \otimes \phi_{0,n_2} \,|\, n_1, n_2 \in \mathbb{Z}\} \cup \{\phi_{0,n_1} \otimes \psi_{j,n_2} \,|\, j \in \mathbb{N}_0 \quad n_1, n_2 \in \mathbb{Z}\} \\ & \cup \{\psi_{j,n_1} \otimes \phi_{0,n_2} \,|\, j \in \mathbb{N}_0 \quad n_1, n_2 \in \mathbb{Z}\} \\ & \cup \{\psi_{j_1,n_1} \otimes \psi_{j_2,n_2} \,|\, j_1, j_2 \in \mathbb{N}_0 \quad n_1, n_2 \in \mathbb{Z}\}.\end{aligned} \tag{3.31}$$

Een karakteristiek van de Mallatbasis is dat deze $\psi \otimes \psi$ functies bevat van enkel dezelfde schaal, terwijl we bij de Tensorbasis te maken hebben met j_1 en j_2 die ongelijk aan elkaar zijn. De Mallatbasis moest door deze eis aangevuld worden met functies van de vorm $\phi \otimes \psi$ en $\psi \otimes \phi$. Dit is bij de Tensorbasis niet meer aan de orde (met uitzondering van enkele samenstellingen van ϕ_0 met ψ_j's)

Met deze theoretische kennis willen we een algoritme bedenken dat een tweedimensionaal signaal schrijft in termen van de Tensorbasis. We zullen zien dat dit neerkomt op het uitvoeren van FWT op alle rijen en kolommen van de ingangssignaal, die we voorstellen als een matrix.

Algoritme (Tweedimensionale Tensor Fast Wavelet Transform)**.** *Gegeven een ingangssignaal $a_j \in \mathbb{R}^{2^j} \times \mathbb{R}^{2^j}$, definiëren we een algoritme* $\mathrm{TFWT}_2 : \mathbb{R}^{2^j} \times \mathbb{R}^{2^j} \to \mathbb{R}^{2^j} \times \mathbb{R}^{2^j}$ *als volgt. Bereken $\tilde{a}_j$ zo dat geldt:*

$$\tilde{a}_j[n_1, n_2] = \mathrm{FWT}(a_j \,|_{\, x_1 = n_1})[n_2]$$

Dan wordt de TFWT_2 *gegeven door:*

$$\mathrm{TFWT}_2(a_j)[n_1, n_2] = \mathrm{FWT}(\tilde{a}_j \,|_{\, x_2 = n_2})[n_1]$$

We zullen aantonen dat de TFWT_2 ook inderdaad een decompositie geeft in termen van de Tensorbasis. We zijn namelijk op zoek naar een matrix van de vorm:

$$\begin{bmatrix} \langle f, \phi_0 \otimes \phi_0 \rangle & \langle f, \phi_0 \otimes \psi_0 \rangle & \cdots & \langle f, \phi_0 \otimes \psi_j \rangle \\ \langle f, \psi_0 \otimes \phi_0 \rangle & \langle f, \psi_0 \otimes \psi_0 \rangle & \cdots & \langle f, \psi_0 \otimes \psi_j \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle f, \psi_j \otimes \phi_0 \rangle & \langle f, \psi_j \otimes \psi_0 \rangle & \cdots & \langle f, \psi_j \otimes \psi_j \rangle \end{bmatrix} \tag{3.32}$$

We weten dat wanneer we de FWT nemen in één coördinaat door de andere coördinaat vast te nemen, dit ons voor elke vaste waarde een vector geeft volgens:

$$y \mapsto \begin{bmatrix} \langle f \,|_{\, x_2 = y}, \phi_0 \rangle & \langle f \,|_{\, x_2 = y}, \psi_0 \rangle & \cdots & \langle f \,|_{\, x_2 = y}, \psi_j \rangle \end{bmatrix}$$

Hieruit kunnen we ook een vector van functies maken door de rol van functie en matrix-index om te wisselen:

$$\begin{bmatrix} y \mapsto \langle f \,|_{\, x_2 = y}, \phi_0 \rangle & y \mapsto \langle f \,|_{\, x_2 = y}, \psi_0 \rangle & \cdots & y \mapsto \langle f \,|_{\, x_2 = y}, \psi_j \rangle \end{bmatrix}$$

Wanneer we vervolgens voor elke functie in deze vector de FWT bereken zullen we weer een vector inproducten krijgen voor elke coördinaat in de originele vector, schrijf deze vector van

vectoren (equivalent met een matrix) uit als $M[k_1, k_2]$, [5] dan volgt:

$$M[k_1, k_2] = \langle y \mapsto \langle f \,\big|_{x_2=y}, \psi_{k_1} \rangle, \psi_{k_2} \rangle$$

We kunnen dit omschrijven door de definitie van het inproduct op functieruimten toe te passen:

$$\langle y \mapsto \langle f \,\big|_{x_2=y}, a \rangle, b \rangle = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x_1, x_2) \cdot a(x_1) \mathrm{d}x_1 \cdot b(x_2) \mathrm{d}x_2 = \langle f, a \otimes b \rangle$$

Waardoor de matrix M identiek is aan (3.32).

Het vinden van een inverse voor dit algoritme is hiermee ook triviaal, de iFWT geeft samengesteld met de FWT immers de identiteit, zodat het toepassen op kolommen en rijen van de iFWT de TFWT_2 inverteert.

Meer dan twee dimensies

Er is een natuurlijke voortzetting van het Tensorproduct naar meerdimensionale signalen. Dit doen we door voor bijvoorbeeld $V_j^{(m)}$ de basis te kiezen die voortkomt uit een tensorproduct van m maal de basis van V_j. Vervolgens is de algoritme te veralgemeniseren naar een algoritme dat over elke coördinaat de FWT uitvoert, met bijbehorende inverse.

Omdat de stap in complexiteit van de notatie vele male groter is dan de benodigde denkstap laten we hier een rigoreuze behandeling van de meerdimensionale Tensor Fast Wavelet Transform achterwege.

Mengvormen van Tensor en Mallat

Zoals uitgelegd in de secties 3.8 en 3.7 kunnen zowel de Mallatdecompositie als het Tensorproduct toegepast worden op 3 of meer dimensies. We zullen beargumenteren dat de beide vormen ook naar eigen inzicht gemengd kunnen worden. We schrijven daarvoor de m-dimensionale ruimte $V_j^{(m)}$ op als een tensorproduct:

$$\begin{aligned} V_j^{(m)} &= V_j^{(k-1)} \otimes V_j \otimes V_j^{(m-k)} \\ &= V_j^{(k-1)} \otimes (V_0 \otimes \cdots \otimes W_{j-1}) \otimes V_j^{(m-k)} \end{aligned}$$

Hier zien we dat de k-de term V_j uitgeschreven is volgens de gewone FWT decompositie. Het staat ons nu vrij om de termen $V_j^{(k-1)}$ en V_j^{m-k} te ontbinden op een andere manier. De gebruikte algoritmes volgen analoog aan de implementatie van het Tensorproduct en de Mallatdecompositie door coördinaten vast te houden.[6]

[5] Hier wordt het inproduct met ϕ_0 weggelaten om de schrijfwijze te verduidelijken. We zien dan ϕ_0 als een ψ_k met een speciale index k.

[6] De notatie voor de mengvorm en de specifieke bewijzen laten we achterwege. Deze leiden teveel af van het doel van het project.

Voorbeeld. We kunnen in 3 dimensies een Tensor-Mallat mengvorm maken:

$$\begin{aligned} V_j^{(3)} = &\Big(V_{j-1}^{(2)} \oplus (W_{j-1} \otimes V_{j-1}) \oplus (V_{j-1} \otimes W_{j-1}) \oplus (W_{j-1} \otimes W_{j-1})\Big) \\ &\otimes (V_0 \oplus \cdots \oplus W_{j-1}) \end{aligned} \tag{3.33}$$

Hier is de eerste term ontbonden volgens de Mallatdecompositie en hebben we met het Tensorproduct een normale FWT decompositie hier recht op gezet.

Gevolg 3.13. Wanneer we in 3.33 de eerste term opvatten als een ruimte van afbeeldingen op een bepaald moment en de tweede term als de tijdruimte die het verloop van een afbeelding geeft, dan kunnen we deze decompositie toepassen op videomateriaal.

3.9. Analyse van de fout van beide decomposities

Bij compressie is men op zoek naar een manier om stukjes data weg te kunnen gooien of te schrijven op zo'n manier dat het minder ruimte inneemt. Wij zijn in het bijzonder geïnteresseerd in *hoe dichtbij* we bij perfecte reconstructie zitten wanneer we een vooraf bepaald hoeveelheid data opslaan. Er is al uitgebreid onderzoek gedaan naar hoe dit werkt bij de waveletbasis en een aantal resultaten hiervan zullen we opnemen in ons verslag.

Laat f een functie in $L_2(\square)$ met aftelbare orthonormale basis $\mathcal{B} = \{g_m\}$. Dan valt f in deze basis te schrijven als (zie Lemma 1.1)

$$f = \sum_{m=0}^{\infty} \langle f, g_m \rangle g_m. \tag{3.34}$$

Wanneer we niet de hele basis bekijken, maar slechts een N aantal elementen pakken, krijgen we een verzameling $\mathcal{B}_N \subset \mathcal{B}$ zodat

$$f|_{\mathcal{B}_N} := \sum_{g_m \in \mathcal{B}} \langle f, g_m \rangle g_m.$$

In het bijzonder zijn we op zoek naar de *fout* van de reconstructie $\|f - f|_{\mathcal{B}_N}\|$:

$$\|f - f|_{\mathcal{B}_N}\|^2 = \left\| \sum_{g_m \in \mathcal{B}} \langle f, g_m \rangle g_m - \sum_{g_m \in \mathcal{B}_N} \langle f, g_m \rangle g_m \right\|^2 = \left\| \sum_{g_m \in \mathcal{B} \setminus \mathcal{B}_N} \langle f, g_m \rangle g_m \right\|^2 = \sum_{g_m \in \mathcal{B} \setminus \mathcal{B}_N} |\langle f, g_m \rangle|^2.$$

Duidelijk moge zijn dat voor $N \to \infty$, $\|f - f|_{\mathcal{B}_N}\|^2 \to 0$.

Definitie 3.14 (Sobolevruimte)**.** Een Sobolevruimte $H^p(\Omega)$ over $\Omega \subset \mathbb{R}$ is de verzameling van alle functies $u \in L_2(\Omega)$ waarvoor geldt dat

$$\frac{\partial^k u}{\partial x^k} \in L_2(\Omega), k \leq p.$$

De norm op H^p die hierbij hoort is gedefiniëerd als

$$\|u\|_{H^p(\Omega)} := \sum_{k \leq p} \left\| \frac{\partial^k u}{\partial x^k} \right\|_{L_2(\Omega)}.$$

In n dimensies:

$$H^{(p_1,p_2,\ldots,p_n)}(\square) := \bigotimes_{k=1}^{n} H^{p_k}([0,1]).$$

Fout van de Mallatdecompositie

Wij zijn op het geïnteresseerd in de Mallat-waveletbasis $\mathbf{\Phi}$ die we vonden in stelling 3.10.Deze is duidelijk aftelbaar, dus we kunnen (3.34) gebruiken.

Definiëer $J_M := \{l \in \mathbb{N}^n : \|l\|_\infty \leq M\}$. Laat $\mathbf{\Phi}_M := \{\boldsymbol{\psi}_{\boldsymbol{\lambda}} \in \mathbf{\Phi} : |\boldsymbol{\lambda}| \in J_M\}$ met $|\boldsymbol{\lambda}| = (|\lambda_1|, \ldots, |\lambda_n|)$ de verzameling basisfuncties tot een niveau M.

Stelling 3.15 (Fout van Mallatdecompositie)**.** *Wanneer $f \in H^{(p,\ldots,p)}(\square)$, zal de fout $\|f - f|_{\mathbf{\Phi}_M}\|$ bij een Mallatdecompositie met de basisfuncties tot niveau M precies $\theta(N^{-p/n})$ zijn, met $N := \#\mathbf{\Phi}_M$ het aantal basisfuncties tot niveau M.*

Bewijs. We maken gebruik van de zogenaamde Jacksonongelijkheid [7] die zegt dat, gegeven de ruimte $\mathbb{P}_{p-1}$ van polynomen van orde $p-1$, geldt:

$$\inf_{q \in \mathbb{P}_{p-1}} ||f - q||_{L_2(\square)} \simeq 2^{-jp} ||f||_{H^{(p,\ldots,p)}(\square)}$$

wanneer $f \in H^{(p,\ldots,p)}(\square)$.

Voor elke dimensie zitten er $\theta(2^M)$ basisfuncties in $\{\psi_\lambda : |\lambda| \leq M\}$. Er zijn n dimensies dus $2^{Mn} = N$ basisfuncties in totaal. Bekijk de fout:

$$\|f - f|_{\mathbf{\Phi}_M}\|^2_{L_2(\square)} = \sum_{\boldsymbol{\psi} \in \mathbf{\Phi} \setminus \mathbf{\Phi}_M} |\langle f, \boldsymbol{\psi} \rangle|^2 \simeq \sum_{|\boldsymbol{\lambda}| > M} 2^{-|\boldsymbol{\lambda}|2p} \simeq 2^{-2Mp},$$

waarbij het laatste gelijkheidsteken voortkomt uit de relatie

$$\sum_{k=M+1}^{\infty} 2^{-kp} = \frac{2^{-Mp}}{2^p - 1}$$

en de notie dat p constant is voor een keuze van de wavelet. De fout is nu 2^{-Mp}. Omschrijven geeft dat dit overeenkomt met een fout van $N^{-p/n}$. □

Fout van het Tensorproduct

We bekijken een aftelbare basis van $L_2(\square)$. In 1 dimensie wordt deze basis gevonden door $\{\psi_\lambda : \lambda \in \nabla\}$, waarbij ∇ precies de indexverzameling van deze basis is.

Volgens [4, L3.1.7] is

$$\boldsymbol{\Phi}_T = \Phi \otimes \cdots \otimes \Phi = \{\boldsymbol{\psi}_{\boldsymbol{\lambda}} := \psi_{\lambda_1} \otimes \cdots \otimes \psi_{\lambda_n} : \lambda_i \in \nabla\}$$

met $\boldsymbol{\lambda} := (\lambda_1, \ldots, \lambda_n) \in \boldsymbol{\nabla} := \nabla^n$ een orthogonale basis voor $L_2(\square)$.

Laat vervolgens $I_M := \{l \in \mathbb{N}_0^n : ||l||_1 \leq M\}$ en maak de *sparse grid index set* $\boldsymbol{\nabla}_M := \{(\boldsymbol{j}, \boldsymbol{n}) \in \boldsymbol{\nabla} : \boldsymbol{j} \in I_M\}$.

We zullen nu een aantal lemma's enumereren die we willen gebruiken om de fout af te schatten. De bewijzen hiervoor staan in [4] en zullen we niet verder behandelen.

Lemma 3.16. *[4, P3.2.3] Voor $f \in H^{(p,\ldots,p)}(\square)$ geldt dat de fout van de benadering op basis van de sparse grid index set $\boldsymbol{\nabla}_M$ voldoet aan*

$$\|f - f|_{\boldsymbol{\nabla}_M}\|_{L_2(\square)} \lesssim 2^{-pM} M^{(n-1)/2} \|f\|_{H^{(p,\ldots,p)}(\square)}.$$

Lemma 3.17. *[4, L3.3.1] Het aantal elementen in $\boldsymbol{\nabla}_M$ is proportioneel met $2^M M^{n-1}$.*

Lemma 3.18. *[4, L3.4.1] Wanneer er voor twee functies h, g geldt dat $h(J) \lesssim J^{-s} \log_2(J)^\mu$ en $g(J) \simeq= \log_2(J)^\nu J =: N$ voor zekere μ en ν, dan geldt:*

$$(h \circ g^{-1})(N) \lesssim N^{-s} \log_2 N^{\mu + \nu s}.$$

Dit is voldoende om een goede afschatting te maken van de *fout van het Tensorproduct.*

Stelling 3.19 (Fout van het Tensorproduct)**.** *Laat $f \in H^{(p,\ldots,p)}(\square)$ en $N = \#\boldsymbol{\nabla}_M$. Dan:*

$$\|f - f|_{\boldsymbol{\nabla}_M}\|_{L_2(\square)} \lesssim N^{-p} \log_2(N)^{(n-1)(1/2+p)} \|f\|_{H^{(p,\ldots,p)}(\square)}.$$

Bewijs. We kunnen met het lemma 3.16 de fout af schatten. Er geldt dat

$$\|f - f|_{\boldsymbol{\nabla}_M}\|_{L_2(\square)} \lesssim M^{(n-1)/2} 2^{-Mp} \|f\|_{H^{(p,\ldots,p)}(\square)}.$$

We verplaatsen alle termen die niet van M afhangen naar het linkerlid, dit geeft:

$$\frac{\|f - f|_{\boldsymbol{\nabla}_M}\|_{L_2(\square)}}{\|f\|_{H^{(p,\ldots,p)}(\square)}} \lesssim M^{(n-1)/2} 2^{-Mp}.$$

We definiëren de functie die voor een gegeven niveau 2^M het linkerlid van deze vergelijking geeft door:

$$\begin{aligned} h : \mathbb{N} &\to \mathbb{R} \\ 2^M &\mapsto \frac{\|f - f|_{\boldsymbol{\nabla}_M}\|_{L_2(\square)}}{\|f\|_{H^{(p,\ldots,p)}(\square)}}. \end{aligned}$$

Vervolgens definiëren we de functie g die het aantal wavelets geeft van dit niveau:

$$\begin{aligned} g : \mathbb{N} &\to \mathbb{N} \\ 2^M &\mapsto N = \#\boldsymbol{\nabla}_M. \end{aligned}$$

Gebruik lemma 3.17 om te vinden dat

$$g(2^M) = N \simeq M^{n-1}2^M.$$

Met deze definities kunnen we lemma 3.18 toepassen. Neem $J = 2^M$ dan voldoen h en g aan de voorwaarden van het lemma voor $s = p$, $\mu = \frac{n-1}{2}$ en $\nu = n - 1$, waaruit volgt dat:

$$(h \circ g^{-1})(N) \lesssim N^{-p} \log_2(N)^{(n-1)\left(\frac{1}{2}+p\right)}.$$

Maar dit betekent precies dat de orde van h herschreven wordt in termen van N door

$$\frac{\|f - f|_{\boldsymbol{\nabla}_M}\|_{L_2(\square)}}{\|f\|_{H^{(p,\ldots,p)}(\square)}} \lesssim N^{-p} \log_2(N)^{(n-1)(1/2+p)},$$

en dit is precies wat we wilden bewijzen. □

Vergelijking van Mallat en Tensor

We hebben nu gevonden dat voor een voldoend gladde f en met een beperkte hoeveelheid coëfficiënten, de Mallatdecompositie slechts een convergentiesnelheid $N^{-p/n}$ bereikt, terwijl het Tensorproduct een snelheid $N^{-p} \log_2(N)^{(n-1)(p+1/2)}$ heeft. De zogenaamde *curse of dimensionality* kan dus verbroken worden door het gebruik van een Tensorproduct.

In de praktijk hebben we echter nooit te maken met compleet gladde functies. Gevolg is dat deze stellingen niet helemaal opgaan. Niet *helemaal*, want we hebben wavlets geconstrueerd met een compacte basis, zodat alleen lokaal wavelets nodig zijn om een discontinuïteit te reconstrueren; we 'zoomen we in' op de functie. Lokaal is de functie f vaak wel glad genoeg zodat de benadering daar goed is.

Een gevolg van de verschillen in lokale gladheid is echter dat we wavelets beter moeten selecteren: meer wavelets rond een discontinuiteit en minder op de gladde delen. Hiervoor gebruiken we een zogenaamde *adaptieve* algoritme die selecteert op de grootte van de coëfficiënten.

Vergelijking met Fouriertransformatie

Wanneer we het convergentiegedrag van de Waveletcoëfficiënten vergelijken met de daling van de coëfficiënten van de Fouriertransformatie dan zien we dat de convergentie van zijn coëfficiënten loopt volgens $N^{-(p-1)}$ voor $f \in C^p$. Hierdoor valt te verwachten dat de Fouriertransformatie beter werkt voor dit soort functies.

Zodra de functie echter een discontinuiteit vertoont treden er problemen op met de Fouriertransformatie. Het reconstrueren van de discontinuïteit wordt enorm bemoeilijkt en de reconstructie voert harde randen over in zachte randen. In zo'n situatie verwachten we dan ook dat de Wavelettransformatie betere resultaten vertoont.

4. Onze implementie

Een discreet signaal in 1 dimensie kan over het algemeen geschreven worden als een vector reële getallen van een bepaalde lengte (de signaallengte). Wanneer we afbeeldingen willen beschrijven, bekijken we daarom vaak matrices van grijswaardes, waarbij de vier kanalen (rood, groen, blauw, doorzichtigheid) elk een eigen matrix hebben die bij elkaar de representatie geven van een kleurenafbeelding.

Omdat het menselijk oog niet heel nauwkeurig is – en door de limitaties van computers – slaat men de matrixcoëfficiënten niet op als reële getallen. In plaats daarvan worden vaak integerwaardes tussen 0 en 255 gebruikt. Omdat elk van de 3 kleurenkanalen zo 256 waardes krijgt, is het kleurenpallet toch groot. In elk van onze implementaties worden beelden gescheiden in haar vier kanalen en wordt op elk kanaal afzonderlijk het algoritme toegpast. De reden voor deze keuze is dat de verschillende kanalen niet veel met elkaar van doen hoeven hebben.

Bewegend beeld is zo op precies dezelfde manier te zien als een driedimensionale matrix waarbij we meerdere afbeeldingen achter elkaar plakken.

In dit hoofdstuk zullen we de *Python*-code die we geschreven hebben toelichten. De code zelf is te vinden in de appendix A; we zullen hier per sectie naar verwijzen.

4.1. Implementatie van Fouriertransformatie in Python

Met de theoretische basis uit voorgaande secties is de implementatie van het *Fast Fourier Transform*-algoritme nu aan bod gekomen. In deze sectie zullen we uitweiden over enkele genomen hordes en gemaakte keuzes. De resultaten volgen in een latere sectie.

Fast Fourier Transform Implementatie [Listings: A.1, A.2, A.3]

De Pythoncode die we gebruikt hebben om het FFT-algoritme in te programmeren volgt precies het schema van de eerder gegeven pseudocode. Dit algoritme werkt – zoals eerder beschreven – enkel op signalen waarvan de lengte N een macht van 2 is. We hebben daarom signaalextensie toe moeten passen in door middel van zero-padding om het programma ook op andersvormige signalen te laten werken. Zie hiervoor de functies `FFT` en `zero_pad`.

Vervolgens hebben we deze code toegepast in twee dimensies. We maken hier gebruik van de

definitie van de MFFT-algoritme, dat grofweg zegt dat een meerdimensionaal algoritme kan worden geconstrueerd door herhaald het 1-dimensionale geval toe te passen. Voor 2 dimensies in het bijzonder betekent dit dat we simpelweg FFT konden toepassen op rijen en kolommen, zoals we doen in `FFT_2D`.

We hebben dit toegepast door middel van function-mapping zoals in §1.1. Met het oog op duidelijkheid is de Zero-Padding-fase uit de code weggelaten en de algoritme verwacht dus nog een $2^n \times 2^m$ matrix. Dit is echter gemakkelijk te implementeren door ook `zero_padding` toe te passen op rijen en kolommen.

Compressie [Listings: A.4, A.5, A.6]

Zoals in de introductie van ons verslag al aan bod gekomen is, hebben we compressie bereikt door de kleinste coëfficiënten weg te gooien en aan de slag te gaan met een kleinere lijst. Dit brengt twee problemen met zich mee: als eerste, welke waarde moet er als *cutoff* genomen worden en ten tweede, hoe slaan we de overgebleven coëfficiënten op?

Cutoff

Hoewel ons eerste programma een voorgegeven cutoff verwachtte, wilden we liever op compressieratio selecteren. Daarom hebben latere versies de `find_cutoff` routine gebruikt. We sorteren hierbij de coëfficiënten die we vinden op grootte en kiezen de cutoff als de coëfficiënt die de lijst opdeelt ten opzichte van de ingegeven `ratio`. Zo krijgen we precies de drempel die we willen.

Opslaan van de coëfficiënten

Voor het opslaan van de coëfficiënten hebben we een adaptieve basis gebruikt, waarvoor we moesten bijhouden *welke* coëfficiënten er opgeslagen werden. Hier hebben wij een feature van Python gebruikt, de *dictionary*. Een dictionary in Python is niks meer dan een lijst van (naam, waarde)-paren. De naam is in ons geval de plek in de matrix en de waarde is het (complexe) getal wat in die cel hoort te staan. Dit geeft aanleiding tot het schrijven van twee routines: `mat2dict` en `dict2mat`.

De `mat2dict` functie selecteert direct de coëfficiënten op basis van een ingegeven `cutoff`. Duidelijk moge zijn dat we, door deze twee functies na elkaar uit te voeren, een matrix terugkrijgen met nullen waar eerst coëfficiënten kleiner dan `cutoff` stonden.

Het .wvg-beeldformaat

Hoewel we een eigen beeldformaat gemaakt hebben, is het ons niet gelukt om de afbeeldingen echt in minder ruimte op te slaan dan hun ongecomprimeerde versie. Dit heeft met twee dingen te maken. Ten eerste is de manier waarop wij de dictionaries opgeslagen hebben niet optimaal geweest.[1] Ten tweede is de Fouriertransformatie een complex algoritme in de zin dat zij complexe coëfficiënten teruggeeft. Door het isomorfisme $\mathbb{C} \simeq \mathbb{R}^2$ moesten wij zodoende twee waardes voor elke coëfficiënt opslaan.

4.2. Wavelets in 1 dimensie [Listings: A.7, A.8]

Een orthogonale wavelet wordt discreet helemaal gekenmerkt door haar filter h. Zoals beschreven in de sectie over de FWT, hebben we daarmee (via vergelijkingen 3.11, 3.14, 3.15, 3.17) feitelijk het algoritme al helemaal in handen. Definieer

$$\texttt{dec\_l} := \bar{h} \quad \texttt{dec\_h} := \bar{g} \quad \texttt{rec\_l} := h \quad \texttt{rec\_h} := g.$$

Hiermee hebben we twee routines geschreven die beiden één van de stappen van de recursief gedefiniëerde FWT respectievelijk iFWT uitvoeren: `next` en `prev`. Deze functies implementeren de convoluties die we gebruikt hebben in de definitie van de discrete wavelet transformaties. We passen deze routines herhaald toe op de approximatie-coëfficiënten in de routines `fwt` en `ifwt`, daarmee is de implementatie gedaan.

4.3. Wavelets in twee dimensies [Listings: A.9, A.10, A.11]

Zoals eerder beschreven is, kunnen we in meer dimensies twee kanten op: de Mallatdecompositie en het Tensorproduct. We zullen aan beiden wat aandacht besteden.

Mallatdecompositie

De implementatie van de Mallatdecompositie maakt gebruik van de functies die we voor het één dimensionale geval gebruik hebben, namelijk `next` en `prev`. We gebruiken deze in de nieuwe routines `next_2d` en `prev_2d` die een function-mapping doen van `next` en `prev` over het deel van de matrix dat we bekijken. Deze routines worden weer herhaald uitgevoerd op het approximatie-kwadrant van de matrix door de functies `fwt2d` en `ifwt2d`.

Omdat we hier in twee richtingen werken, moet het signaal bovendien aangevuld worden tot een vierkante tweemacht, [2] hiervoor gebruikten we de tweedimensionale `zero_pad_2d`.

[1] Later bedachten we nog een manier die ongeveer 50% van de oorspronkelijke ruimte in zou nemen, door niet (index, waarde)-paren op te slaan maar met een zogenaamde bitmap.

[2] Het is mogelijk om een rechthoek te transformeren maar daar hebben wij hier geen aandacht aan besteed.

Tensorproduct

Het Tensorproduct in meer dimensies uit gaat in principe net zoals de Fouriertransformatie in meer dimensies door op rijen en kolommen de algoritme in 1 dimensie toe te passen. Dit wordt gedaan door de functies `fwt2d_tensor` en `ifwt2d_tensor`.

4.4. Hogere dimensies [Listings: A.12, A.13]

In hogere dimensies wordt het nog interessanter. In bijvoorbeeld drie dimensies kunnen we – naast de Mallatdecompositie en het Tensorproduct – ook de eerder genoemde mengvorm toepassen. Wij hebben er voor gekozen om in dit geval de Mallatdecompositie in het (x, y)-vlak te gebruiken, en het Tensorproduct in de z-richting hier loodrecht op.

Mallatdecompositie

De Mallatdecompositie in drie dimensies is gemakkelijk voort te zetten uit de tweedimensionale variant, er moet nu namelijk tijdens elke stap naast een `next` of `prev`-stap in de y-en z-richting ook nog een stap in de z-richting gedaan worden. Dit hebben we geimplementeerd in de `next_3d` en `prev_3d` functies die weer aangeroepen worden door de `fwt3d` en `ifwt3d` functies.

Mengvorm

Bij de mengvorm gaan we iets interessanters doen: zoals bij het tweedimensionale Tensorproduct voeren we weer function-mapping uit maar dit maal voeren we de tweedimensionale Mallatdecompositie uit op elk (x, y)-vlak en voeren we vervolgens de FWT uit op elke rij in de z-richting. Dit hebben we geïmplementeerd in `fwt3d_mix` en `ifwt3d_mix`.

5. Resultaten

In dit hoofdstuk zullen we een kwantitatieve vergelijking tussen de verschillende algoritmes maken. Hierbij zullen we het begrip *Peak Signal To Noise Ratio* introduceren, een veelgebruikte methode om de reconstructiequaliteit van een lossy compressie-algoritme te testen.

Definitie 5.1 (Peak Signal To Noise Ratio (PSNR))**.** Gegeven een signaal $f : \mathbb{Z}^k \to \mathbb{R}^k$ en een reconstructie $\hat{f}$, is de PSNR gelijk aan

$$\mathrm{PSNR}(f) := 20 \cdot \log_{10}(M) - 10 \cdot \log_{10}(S),$$

waarbij

$$M := \max(\max_{\boldsymbol{j} \in \mathbb{Z}^k}(f[\boldsymbol{j}]), \max_{\boldsymbol{j} \in \mathbb{Z}^k}(\hat{f}[\boldsymbol{j}]))$$

en

$$S := \|f - \hat{f}\|_{\ell_2}^2 = \sum_{\boldsymbol{j} \in \mathbb{Z}^k} (f[\boldsymbol{j}] - \hat{f}[\boldsymbol{j}])^2.$$

Hoe hoger de PSNR is, hoe beter de reconstructie over het algemeen zal zijn.

Gebruikte testbeelden

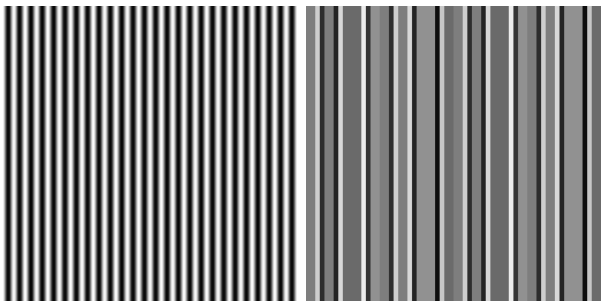
Met een beetje kwade wil kan altijd een beeld geconstrueerd worden dat met minder coëfficiënten is te schrijven in de ene basis dan in de andere: voor een extreem voorbeeld zie figuur 5.1. Omdat het object van studie een beeldcompressie-algoritme is, hebben wij ons gericht op een realistische variëteit beeldmateriaal waarvoor dit algoritme gebruikt zou kunnen worden. De afbeeldingen kunnen ruwweg verdeeld worden in een aantal categorieën:

Fotorealistische beelden worden gekarakteriseerd door het ontbreken van scherpe randen hoewel er veel *details* belangrijk zijn. Ook is het verloop in het beeld voornamelijk *vloeiend* (zie figuren 5.5 - 5.7).

Cartoony beelden zijn over het algemeen minder complex (weinig belangrijke details en minder vloeiend) en bestaat in het algemeen uit *kleurvlakken* omgeven met *scherpe randen* (zie figuren 5.8 - 5.10).

Computer Graphics spant een grotere categorie op die de *scherpe randen* deelt met de Cartoony beelden maar ook veel *vloeiende* verlopen heeft in plaats van kleurvlakken (zie figuren 5.2 - 5.4).

Pixel Art lijkt op de Cartoony beelden vanwege de *scherpe randen* en *kleurvlakken* maar heeft bijna altijd een enorme hoeveelheid *details* aangezien het beeld pixel voor pixel vervaardigd is (zie figuur 5.12).

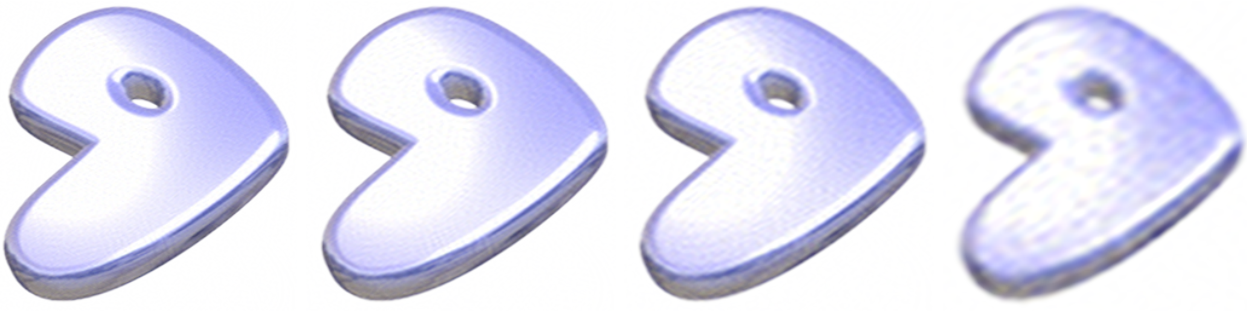


Figuur 5.1.: Een periodiek signaal wordt perfect door de Fouriertransformatie gereconstrueerd door 1 % van de data; de wavelettransformatie geeft hier een echter een slechte reconstructie.

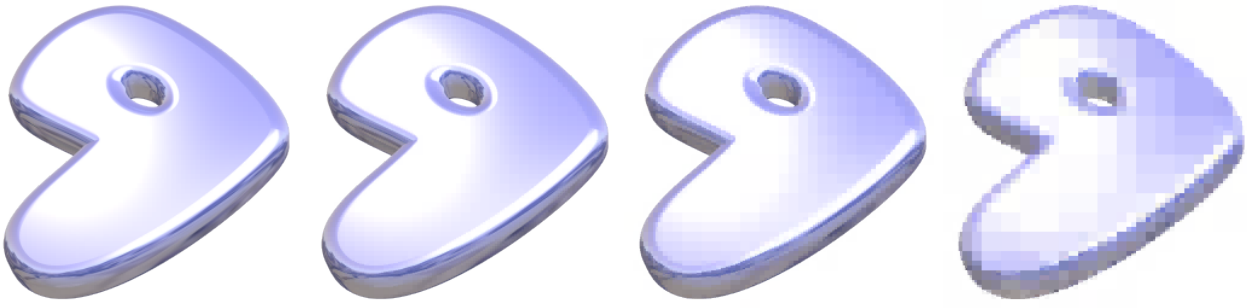
3D signalen

Behalve de illustraties die op de volgende pagina's te zien zijn, hebben we ook een aantal bewegende beelden (3D-signalen) gecomprimeerd. Deze kunt u als lezer vinden door uw webbrowser te sturen naar `http://goo.gl/PXTca`.

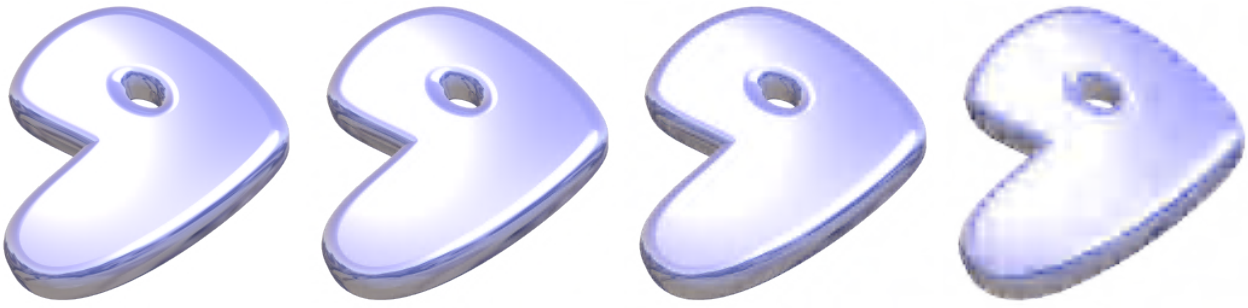
We hebben bij deze 3D-signalen niet Fourier vergeleken met Wavelets. Dit was het gevolg van praktische bezwaren (de Fouriertransformatie deed er simpelweg te lang over). Wel hebben we de Mallatdecompositie vergeleken met onze mengvorm: in het (x, y)-vlak gebruiken we de Mallatdecompositie in twee dimensies, met daar loodrecht op een Tensorproduct in de tijdsrichting.



Figuur 5.2.: Fourier op compressieniveaus 0.15, 0.10, 0.05, 0.01.

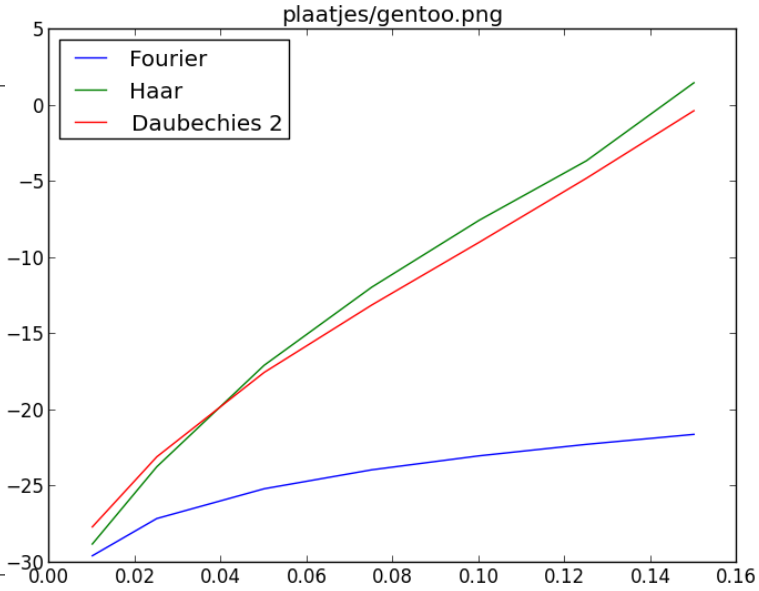


Figuur 5.3.: Haar op compressieniveaus 0.15, 0.10, 0.05, 0.01.



Figuur 5.4.: Daubechies 2 op compressieniveaus 0.15, 0.10, 0.05, 0.01.

Compr	Fourier	Haar	DB2
0.150	−21.564987	1.508043	−0.325757
0.125	−22.220604	−3.607729	−4.753701
0.100	−22.974077	−7.518578	−8.964313
0.075	−23.893526	−11.909545	−13.080699
0.050	−25.134504	−17.028756	−17.498265
0.040	−25.786567	−19.471026	−19.515331
0.030	−26.597406	−21.936183	−21.791275
0.020	−27.693270	−25.283096	−24.377904
0.010	−29.532364	−28.763393	−27.638580

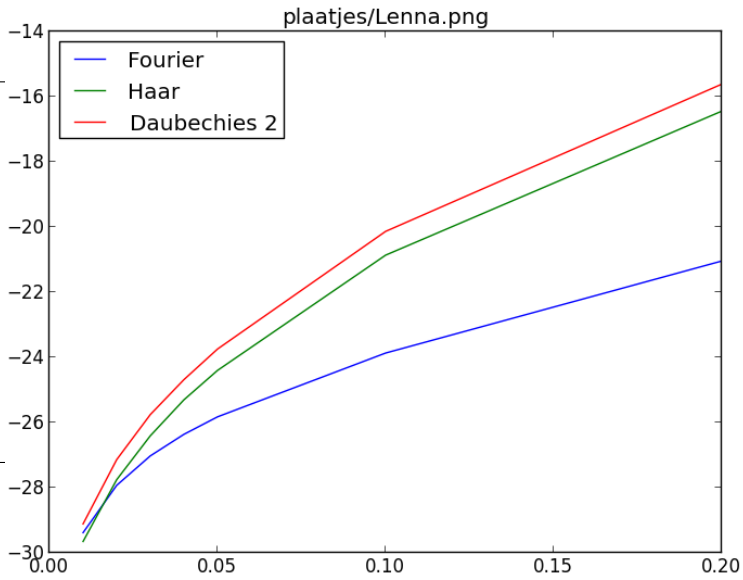


Figuur 5.5.: Fourier op compressieniveaus 0.10, 0.05, 0.03, 0.01.

Figuur 5.6.: Haar op compressieniveaus 0.10, 0.05, 0.03, 0.01.

Figuur 5.7.: Daubechies 2 op compressieniveaus 0.10, 0.05, 0.05, 0.01.

Compr	Fourier	Haar	DB2
0.200	−21.043701	−16.443978	−15.615197
0.100	−23.867860	−20.863769	−20.134351
0.050	−25.825943	−24.400495	−23.745103
0.040	−26.362758	−25.303459	−24.684358
0.030	−27.024421	−26.407813	−25.757180
0.020	−27.928360	−27.754692	−27.139561
0.010	−29.380617	−29.651147	−29.111878

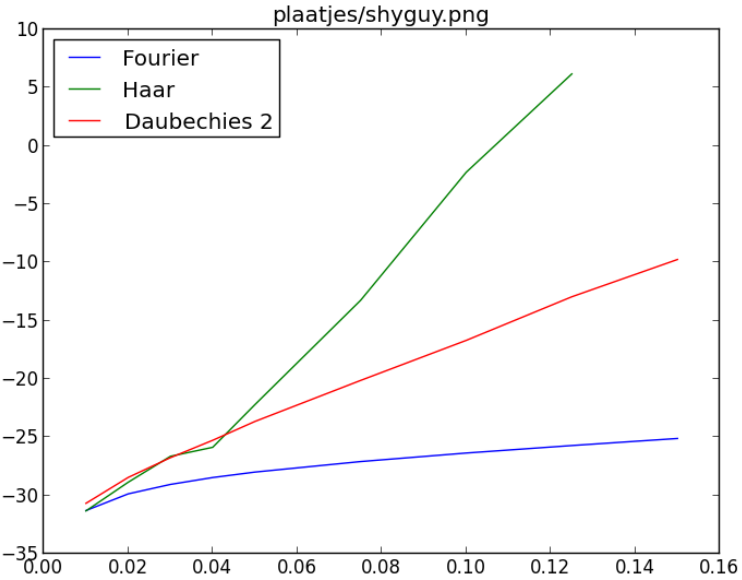


Figuur 5.8.: Fourier op compressieniveaus 0.10, 0.05, 0.03, 0.01.

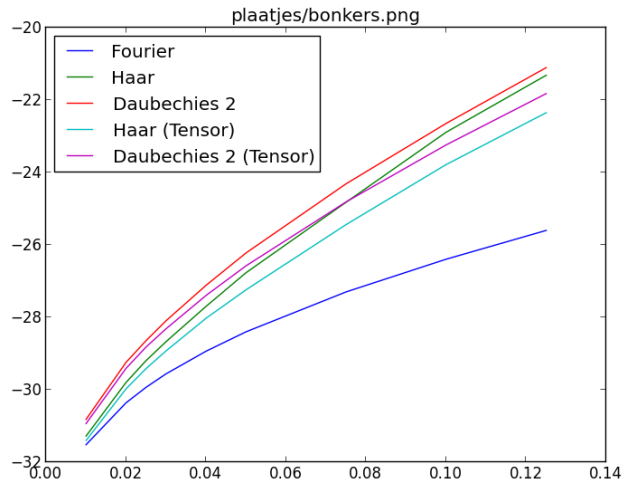
Figuur 5.9.: Haar op compressieniveaus 0.10, 0.05, 0.03, 0.01.

Figuur 5.10.: Daubechies 2 op compressieniveaus 0.10, 0.05, 0.05, 0.01.

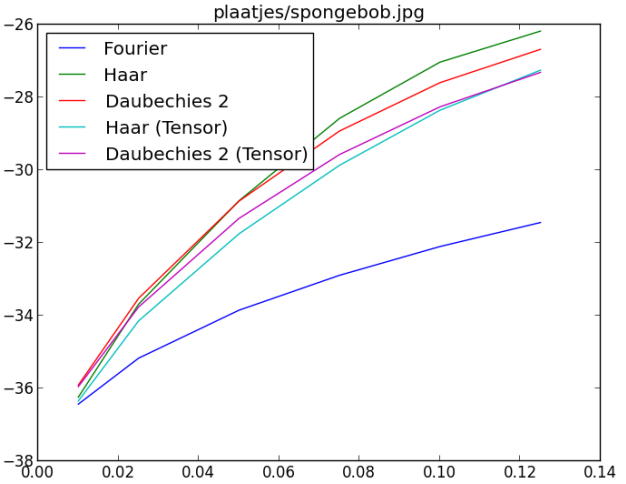
Compr	Fourier	Haar	DB2
0.125	−25.709419	6.188436	−12.939796
0.100	−26.352690	−2.252695	−16.684428
0.075	−27.084570	−13.247047	−20.121718
0.050	−27.987835	−22.194021	−23.640705
0.040	−28.455719	−25.861219	−25.259163
0.030	−29.045916	−26.633403	−26.750137
0.020	−29.857696	−28.858775	−28.445793
0.010	−31.289113	−31.320495	−30.669577



Figuur 5.11.: Links: `plaatjes/spongebob.jpg`. Midden: compressieniveau 0.025 met Daubechies 2. Rechts: compressie 0.025 met Daubechies-2 Tensor.



(a) Grafiek horende bij de onderstaande afbeeldingen.



(b) Grafiek horende bij de bovenstaande afbeeldingen.

Figuur 5.12.: Links: `plaatjes/bonkers.png`. Midden: compressieniveau 0.025 met Haar. Rechts: compressie 0.025 met Haar Tensor.

6. Discussie

In dit laatste hoofdstuk zullen we een analyse maken van het praktische deel van ons project. We beginnen met een discussie van de gebruikte methoden en bruikbaarheid van de gevonden resultaten. Verder worden de resultaten besproken voor het comprimeren van afbeeldingen, het toepassen van het Tensorproduct en als laatste het gebruik van de Tensor/Mallat mengvorm op filmmateriaal.

6.1. Onderzoeksmethoden

Gebruik van PSNR als maatstaf voor kwaliteit

Omdat we te maken hebben met lossy beeldcompressie dienen we dit verlies in kaart te brengen op een objectieve en kwantitatieve manier. De PSNR is een gebruikelijke grootheid voor dit soort analyses: er wordt gezegd dat dit voor eenzelfde beginsignaal een goede reflectie geeft van de menselijke beleving van de kwaliteit van de reconstructie. Uit onderzoek [11] blijkt dat de PSNR een goede maatstaf is zolang er in zeker zin *ceteris paribus* is aangehouden; voor hetzelfde beeldmateriaal presteert deze maat goed.

Adaptieve basis

We hebben in onze implementatie een adaptieve basis gebruikt: de compressie-algoritme behoudt coëfficiënten die in absolute waarde het grootst zijn. We beweren dat dit de beste reconstructie geeft, preciezer: de totale kwadratische fout is zo het kleinst. We beroepen ons daarvoor op de Parsevalgelijkheid. Wanneer we een selectie S doen van basisfuncties die we behouden wordt de totale fout na reconstructie

$$\|f - f|_S\|_{L_2}^2 = \sum_{k \notin S} |\langle f, k \rangle|^2 = \sum_{k \notin S} |c_k|^2,$$

waar c_k de coëfficienten is van de basisfunctie k. Deze sommatie is nu het kleinst wanneer alle termen in de sommatie zo klein mogelijk zijn.

6.2. Compressie van afbeeldingen

Prestatie van Fourier vs. Wavelets

Uit de PSNR-grafieken blijkt dat de Fouriertransformatie een slechtere reconstructie levert voor dezelfde dataset dan zowel de Haar- als de Daubechies-2-wavelettransformaties, ongeacht de categorie waartoe het beeld behoort.

Wavelet vs. Wavelet

Tussen de twee wavelets zien we verder wat we theoretisch verwachten: Haar presteert goed bij harde randen vanwege zijn kleine drager terwijl Daubechies voor vloeiend/fotografisch materiaal een goede reconstructie geeft.

Convergentie bij kleine dataset

Uit de grafiek blijkt dat het verschil in kwaliteit tussen de verschillende compressie-algoritmes afneemt wanneer de hoeveelheid data die opgeslagen wordt, kleiner wordt. De verschillen in prestatie nemen waarschijnlijk af omdat bij een kleine dataset alleen een *globale* approximatie gemaakt kan worden. Het voordeel van de Wavelettransformatie, namelijk die van een *lokale* drager, is daardoor niet meer van belang.

Ook is het dubieus of de PSNR nog een goede maatstaf is voor de menselijke perceptie wanneer de algoritmes zo'n slechte maar zeer verschillende benadering geven van het originele beeld. We zullen daarom hier geen uitspraak over kunnen doen.

6.3. Het Tensorproduct op afbeeldingen

Voor de volledigheid hebben we er voor gekozen óók het Tensorproduct te bekijken bij een aantal van onze afbeeldingen: zie figuren 5.11-5.12. We hebben er voor gekozen om niet alle afbeeldingen in dit formaat weer te geven: deze twee afbeeldingen geven een goede representatie.

Het eerste wat op te merken valt, zijn de horizontale en verticale balken die bij het Tensorproduct tevoorschijn komen. Deze zijn in de Mallatdecompositie een stuk minder aanwezig. De reden hiervoor is simpel aan te wijzen. Bij de Mallatdecompositie bekijken we wavelets met een drager die in beide richtingen even groot is (een vierkant dus). Het Tensorproduct legt deze eis niet op met als gevolg dat er een wavelet is met een drager die heel breed is en ook weer heel kort: een balk. Als deze balk door de compressie-algoritme weggelaten wordt,

mist de reconstructie hier een benadering en kan er een gekke horizontale/verticale balk in de reconstructie onstaan.

Het tweede wat we opmerkten is dat de PSNR structureel lager is bij het Tensorproduct in vergelijking met de Mallatdecompositie. Dit komt ook wel overeen met wat we zien in de figuren 5.11 en 5.12. Merk wel op dat vooral figuur 5.11 een extreem voorbeeld is en dat de verschillen in figuur 5.12 al een stuk minder overheersend zijn.

Het is interessant om op te merken dat onze metingen – dat het Tensorproduct consistent lagere PSNR-waardes oplevert – in strijd zijn met de claims die gemaakt worden in [19], hoewel er teveel onbekende factoren zijn om te weten of dit misschien een andere reden heeft.

6.4. Gemengde decompositie op 3D signalen

Het bekijken van 3D signalen was een interessante toevoeging aan ons project maar de analyse van filmmateriaal is moeilijker gebleken. Dit had twee redenen.

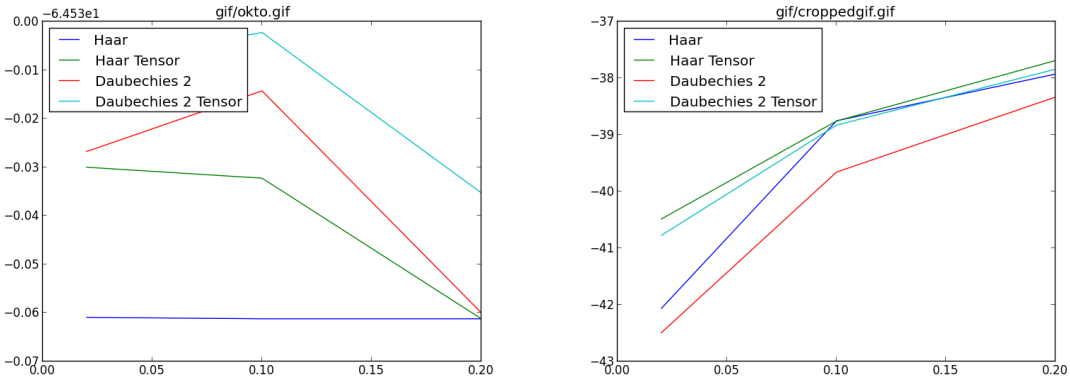
Ten eerste is onze implementatie, hoewel efficiënt, niet erg snel. Het comprimeren van een afbeelding duurt ongeveer dertig seconden en het comprimeren van filmmateriaal duurt al snel het vijftigvoudige. Hierdoor wordt het comprimeren van voldoende 3D signalen een langdurige taak.

Daarnaast hebben we veel moeite gehad om passend beeldmateriaal te vinden. Uit het eerste punt volgt dat dit beeldmateriaal klein in afmeting moet zijn. Als de afbeelding klein is, is het echter moeilijk om een bepaalde compressie-ratio te halen met een goede reconstructie. Dit maakte de zoektocht lastig.

PSNR

Hoewel de PSNR in het geval van de afbeeldingen een prima representatie geeft van de kwaliteit van de reconstructie, blijkt het bij het beeldmateriaal dat wij bekeken hebben minder goed te werken. Zie bijvoorbeeld de grafiek links in 6.1, hier is te zien dat de PSNR slechts weinig verschilt tussen de verschillende compressieniveau's zodat we geen nuttige conclusies kunnen trekken. Daarentegen zien we in grafiek rechts in 6.1 een patroon dat we verwachten, daling van de PSNR bij kleinere compressiepercentages.

Aangezien we niet erg veel beeldmateriaal bekeken hebben zullen we de PSNR als maat loslaten en ons richten op de optische verschillen die we kunnen aanwijzen tussen de twee reconstructies.



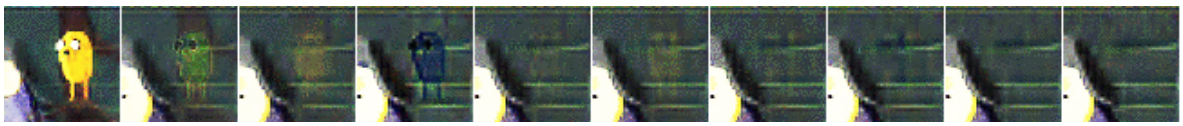
Figuur 6.1.: Links: de grafiek horende bij de dansende octopus. Rechts: de grafiek horende bij de scene met de typmachine en de boom.

Mallatdecompositie versus de mengvorm

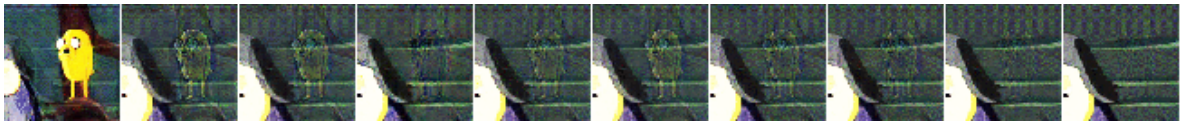
Wat we wel direct opmerkten is het enorme verschil in detail tussen de twee compressiemethoden. Dit komt omdat het Tensorproduct sneller convergeert bij continue signalen (zie stellingen 3.15 en 3.19). Omdat een bewegend beeld over het algemeen continu is in de tijdsdimensie, is het gevolg niet vreemd.

Wanneer er wél een grote sprong in de tijd optreedt, zoals bij een scenewisseling, is de fout van het Tensorproduct ineens een stuk groter dan die van de Mallatdecompositie. Dit is vooral duidelijk bij de typmachinescene, die voor de Mallatdecompositie en de mengvorm weergegeven is in figuren 6.2a en 6.2b. Het gele mannetje vervaagt in de Mallatdecompositie gedurende ongeveer 5 tijdseenheden terwijl de randen in de mengvorm voor 7 frames zichtbaar blijven.

Dit resultaat is ten eerste te wijten aan de grote(re) drager van de Daubechieswavelet, waardoor harde randen slechter gereconstrueerd worden. Het effect wordt echter versterkt in de mengvorm doordat hier veel wavelets een langwerpige drager in de tijdsrichting hebben, wanneer de scherpe randen in frame 71 gereconstrueerd moeten worden zijn deze ook goed te zien in de volgende frames.



(a) Mallatdecompositie



(b) Mengvorm

Figuur 6.2.: `croppedgif.gif` ingezoomd op de verstoring, frames 71 t/m 80 met 2% compressie door de Daubechies 2-wavelet met de verschillende decomposities.

7. Populaire samenvatting

Vandaag de dag is men steeds meer bezig met het delen van informatie. Omdat het belangrijk is dat deze informatie ergens opgeslagen wordt, zijn we altijd op zoek naar *compressie*: het 'inpakken' van informatie zodat het minder ruimte in beslag neemt.

Bij deze compressie zijn twee types te onderscheiden. Bijvoorbeeld tekst moet na de compressie perfect ge*reconstrueerd* kunnen worden. Dit soort signalen zullen wij niet behandelen. Wij zijn geïnteresseerd in *lossy* compressie, wat betekent dat we 'oninteressante informatie' gewoon weg kunnen gooien met de hoop dat de reconstructie goed genoeg lijkt op het origineel.

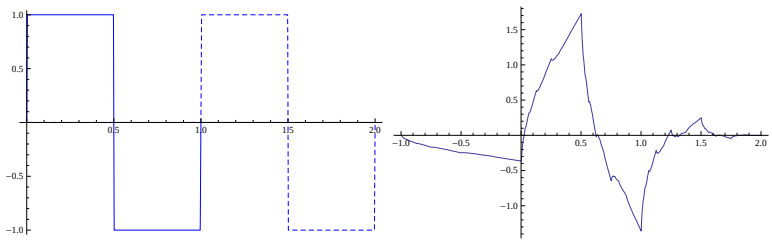
In het bijzonder zullen wij ons vizier richten op beeldmateriaal, eerst stil en daarna bewegend. Foto's worden veelal opgeslagen in het zogenaamde JPEG-beeldformaat: `.jpg`. De wiskunde achter dit compressie-algoritme is niet triviaal en wij zullen hier dan ook grondig op ingaan in ons verslag. De Fouriertransformatie wordt in *signaalverwerking* al honderden jaren toegepast om signalen (functies, geluid, foto's, films) te schrijven in een andere *basis*, namelijk de basis van de complexe e-machten $e^{2\pi it}$. We hopen hierbij maar dat ons signaal goed te benaderen valt in deze basis: een klein aantal grote coëfficiënten en een groot aantal kleine.

Het lot wil dat dit helemaal niet altijd goed werkt, de Fouriertransformatie werkt namelijk niet goed wanneer een afbeelding harde randen heeft. Daarm is er in de laatste dertig jaar een nieuwe speler op het gebied van signaalverwerking opgedoken, de *Wavelettransformatie*. We spreken dan over een nieuwe versie van JPEG: JPEG-2000.

De wavelettransformatie maakt gebruik van de wavelet, een rëele functie waarvoor geldt dat $\int_{-\infty}^{\infty} \psi(t)dt = 0$; deze heeft dus net als een golf even hoge pieken als dalen, vanwaar de naam. Dit op zich is nog niet handig, maar er zijn wavelets met een zogenaamde *compacte drager*: de functie is identiek 0 buiten een *begrensd* en *gesloten* gebied. Hierin ligt dan het verschil met de complexe e-machten van de Fouriertransformatie, waarvan de drager gelijk is aan $\mathbb{R}$. Het effect van deze compacte drager is nu dat niet elke wavelet last heeft van een harde rand in het plaatje, maar slechts een deel van de functies. Zie figuur 7.1 voor twee van de wavelets die we gebruikt hebben.

Een orthogonale waveletbasis is nu een verzameling $\{\psi_{j,n} : j \in \mathbb{N}_0, n \in \mathbb{Z}\}$ waarbij een basisfunctie $\psi_{j,n}(t) = 2^{j/2}\psi(2^j t - n)$ de waveletfunctie is met een verschuiving over n en een dilatie met factor 2^j. Wat dit in feite betekent is dat we gaan inzoomen met onze wavelets: het signaal wordt op verschillende (zoom)*niveau's* bekeken. De orthogonaliteitseigenschap houdt tenslotte in dat voor alle basisfuncties geldt dat $\langle \psi_{a,b}, \psi_{c,d} \rangle = \delta_{a,c} \cdot \delta_{b,d}$.

Zie wederom figuur 7.1 voor een grafiek van $\psi_{0,0} = \psi$ en $\psi_{0,-1}$ voor twee wavelets. De



Figuur 7.1.: Links: De Haarwavelet. Rechts: De Daubechies-2 wavelet.

Haarwavelet is de oudste (en meest simpele) wavelet in gebruik en staat centraal in ons verslag. Rechts is de Daubechies-2 wavelet te zien, die andere interessante eigenschappen heeft.

De Fouriertransformatie en Wavelettransformatie geven ons op deze manier een wiskundig onderbouwde manier om signalen in één dimensie (bijvoorbeeld geluid) te comprimeren naar een fractie van haar oorsponkelijke grootte. We hopen dan dat de reconstructie nog dichtbij het origineel ligt (en het geluid goed te verstaan is). In twee dimensies gebruiken we voor de Fouriertransformatie het zogenaamde *Tensorproduct*. Dit is een natuurlijke manier om meer ruimtes (in ons geval van functies) loodrecht op elkaar te zetten tot een n-dimensionale ruimte. In twee dimensies krijgen de basiselementen nu een tweedimensionale vorm, we bekijken in de ene richting een wavelet en in de andere richting een andere; de drager wordt hiermee een rechthoek.

Bij de Fouriertransformatie blijkt zo'n voortzetting naar meer dimensies met het Tensorproduct gewoon goed te gaan. Bij de Wavelettransformatie zijn er twee mogelijkheden die beide gebruikt worden in de praktijk. In twee dimensies kunnen we óf in beide richtingen even snel inzoomen (dit is de zogenaamde Mallatdecompositie), óf niet (ons 'Tensorproduct'). De elementen van de Mallatdecompositie hebben over het algemeen een vierkante drager terwijl deze in het Tensorproduct een (langwerpige) rechthoek kunnen worden. Dit verschil heeft zichtbare gevolgen.

In het praktische deel van ons verslag hebben we beide decomposities bekeken met interessante resultaten. Het blijkt namelijk dat voor signalen met 'redelijk weinig' harde randen het Tensorproduct een goede reconstructie geeft. Dit feit hebben we geëxploiteerd door een mengvorm van de Mallatdecompositie en het Tensorproduct te gebruiken in de analyse van 3D-signalen. Ook dit gaf interessante en mooie resultaten.

Figuur 7.2.: Logo van VW op drie manieren gecomprimeerd met 1% van de originele data.

A. Pythoncode

Listing A.1: FFT algoritme in Python, voert de pseudocode uit zoals in sectie 2.2

```
def FFT( xs ):
    N = len(xs)
    if N <= 1:                # randconditie
        return xs
    else:
        even = FFT(xs[0::2])  # voer FFT uit op even indices
        odd  = FFT(xs[1::2])  # voer FFT uit op oneven indices

        return [0.5*(even[k] + exp(-2j*pi*k/N)*odd[k]) for k in range(N/2)] +
               [0.5*(even[k] - exp(-2j*pi*k/N)*odd[k]) for k in range(N/2)]
```

Listing A.2: Zero-padding algoritme in Python, voegt nullen toe tot een tweemacht is bereikt

```
def zero_pad( xs ):
    N_old = len(xs)
    N_new = 2**ceil(log(N_old,2))     # rond logaritme af voor kleinste tweemacht
    return [xs[k] if (k < N_old) else 0 for k in range(N_new)]
```

Listing A.3: 2-Dimensionaal FFT algoritme

```
def FFT_2D( xss ):
    xss = map(FFT, xss)      # voer FFT uit op rijen

    xss_t = transpose(xss)   # verwissel rijen met kolommen
    xss_t = map(FFT, xss_t)  # voer FFT uit op kolommen
    xss   = transpose(xss_t) # maak verwisseling ongedaan

    return xss
```

Listing A.4: Het vinden van een goede cutoff-waarde gegeven een gewenst compressieniveau

```
def find_cutoff( mat, ratio ):
	if ratio < 0 or ratio > 1: return
	length = reduce( lambda x, y: x*y, array.shape ) #aantal cellen in de matrix
	vector = reshape(array, length)
#reshape naar 1 lange vector
	sort(modulus(vector))
#sorteer de vector
	return vector[ int( ratio*len(vector) ) ]
```

Listing A.5: Matrix naar dictionary conversie

```
def mat2dict( mat, cutoff ):
	N, M = mat.shape
	dict = {}
	for y in range(N):
		for x in range(M):
			if modulus( mat[y][x] ) >= cutoff:
				dict[(y,x)] = mat[y][x]
	return (dict, N, M)
```

Listing A.6: Dictionary naar matrix conversie

```
def dict2mat( dict, N, M ):
	mat = []
	for y in range( N ):
		row = []
		for x in range( M ):
			i = (y, x)
			row[x] = dict[i] if i in dict else 0.0
		mat[y] = row
	return mat
```

Listing A.7: De FWT

```
def next(signal):
       low = downsampling_convolution( signal, dec_l )
       hi = downsampling_convolution( signal, dec_h )
       return concatenate(low, hi)

def fwt(signal):
       signal = zero_pad(signal)
       steps = int( log( len(signal), 2 ) ) #neem de 2-log van de lengte

       for i in range( steps ):
              k = len(output) / 2**i #eerst alles, dan de helft, dan een kwart
              signal[0:k] = next( signal[0:k] )
       return signal
```

Listing A.8: De iFWT

```
def prev(signal):
       N = len(signal)
       low = upsampling_convolution( signal[:N//2], rec_l )
       hi = upsampling_convolution( signal[N//2:], rec_h )
       return sum(low, hi)  #elementsgewijze sommatie

def ifwt( wavelet, signal, original_length ):
       steps = int( log( len(signal), 2) ) #neem de 2-log van de lengte

       for i in range( steps ):
              j = steps - i - 1 #we gaan andersom
              k = len(signal) / 2**j
              signal[0:k] = prev( signal[0:k])
       return signal[0:original_length] #het signaal was aangevuld tot tweemacht
```

Listing A.9: De Mallatdecompositie in 2 dimensies

```
def next_2d( signal ):
       x, y = signal.shape
       for j in range(x): #1d-fwt op de rijen
              signal[j, :] = next( signal[j, :] )
       for i in range(y): #1d-fwt op de kolommen
              signal[:, i] = next( signal[:, i] )
       return signal

def fwt2d(wavelet, signal):
       signal = zero_pad_2d(signal)
       steps = int( log( len(signal), 2 ) ) #neem de 2-log van de lengte

       for i in range( steps ):
              k = len(output) / 2**i #eerst alles, dan de helft, dan een kwart
              signal[0:k,0:k] = next_2d( signal[0:k,0:k] )
       return signal
```

Listing A.10: De inverse Mallatdecompositie in 2 dimensies

```
def prev_2d( signal ):
       x, y = signal.shape
       for j in range(x): #1d-ifwt op de rijen
              signal[j, :] = prev( signal[j, :] )
       for i in range(y): #1d-ifwt op de kolommen
              signal[:, i] = prev( signal[:, i] )
       return signal

def ifwt2d( signal, N, M ): #N en M zijn originele lengte en breedte
       steps = int( log( len(signal), 2) ) #neem de 2-log van de lengte

       for i in range( steps ):
              j = steps - i - 1 #we gaan andersom
              k = len(signal) / 2**j
              signal[0:k,0:k] = prev_2d( signal[0:k,0:k] )
       return signal[0:N,0:M]
```

Listing A.11: De 2D FWT in het Tensorgeval

```
def fwt2d_tensor(signal):
	signal = zero_pad(signal)
	steps = int( log( len(signal), 2 ) ) #neem de 2-log van de lengte

	for i in range( steps ):
		k = len(output) / 2**i #eerst alles, dan de helft, dan een kwart
		for p in range( len( signal ) ):
			signal[p, 0:k] = next( signal[p, 0:k] )
		for q in range( len( signal ) ):
			signal[0:k, q] = next( signal[0:k, q] )
	return signal

def ifwt2d_tensor( signal, N, M ):
	steps = int( log( len(signal), 2) ) #neem de 2-log van de lengte

	for i in range( steps ):
		j = steps - i - 1 #we gaan andersom
		k = len(signal) / 2**j
		for q in range( len( signal ) ):
			signal[0:k, q] = prev( signal[0:k, q] )
		for p in range( len( signal ) ):
			signal[p, 0:k] = prev( signal[p, 0:k] )
	return signal[0:N,0:M] #het signaal was aangevuld tot een tweemacht
```

Listing A.12: De Mallatdecompositie in 3 dimensies

```
def next_3d( signal ):
	x, y, z = signal.shape
	for j in range(x): #2d-fwt op de rijen
		signal[j, :, :] = next_2d( signal[j, :, :] )
	for k in range(y):
		for l in range(z): #1d-fwt op de kolommen
			signal[:,k,l] = next( signal[:,k,l] )
	return signal

def fwt3d(wavelet, signal):
	signal = zero_pad_3d(signal)
	steps = int( log( len(signal), 2 ) ) #2-log van de lengte in 1 richting

	for i in range( steps ):
		k = len(output) / 2**i #eerst alles, dan de helft, dan een kwart..
		signal[0:k,0:k,0:k] = next_3d( signal[0:k,0:k,0:k] )
	return signal

def prev_3d( signal ):
	x, y, z = signal.shape
	for j in range(x): #2d-ifwt op de rijen
		signal[j, :, :] = prev_2d( signal[j, :, :] )
	for k in range(y):
		for l in range(z): #1d-ifwt op de kolommen
			signal[:,k,l] = prev( signal[:,k,l] )
	return signal

def ifwt3d( wavelet, signal, N, M, O ): #N, M en O zijn lengte, breedte en diepte
	steps = int( log( len(signal), 2) )

	for i in range( steps ):
		j = steps - i - 1 #we gaan andersom
		k = len(signal) / 2**j
		signal[0:k,0:k,0:k] = prev_3d( signal[0:k,0:k,0:k] )
	return signal[0:N,0:M,0:O]
```

Listing A.13: De mengvorm in 3 dimensies

```
def fwt3d_mix(signal):
       signal = zero_pad_3d(signal)
       steps = int( log( len(signal), 2 ) ) #2-log van de lengte in 1 richting

       for i in range( steps ):
              m = len(output) / 2**i #eerst alles, dan de helft, dan een kwart..
              for j in range(len( signal )): #2d-fwt op de rijen
                     signal[j, 0:k, 0:k] = next_2d( signal[j, 0:k, 0:k] )
              for k in range(len( signal )):
                     for l in range(len( signal )): #1d-fwt op de kolommen
                            signal[0:k,m,l] = next( signal[0:k,m,l] )

       return signal

def ifwt3d_mix( signal, N, M, O ): #N, M en O zijn lengte, breedte en diepte
       steps = int( log( len(signal), 2) )

       for i in range( steps ):
              j = steps - i - 1 #we gaan andersom
              m = len(output) / 2**j #eerst alles, dan de helft, dan een kwart..
              for n in range(len( signal )): #2d-fwt op de rijen
                     signal[n, 0:k, 0:k] = prev_2d( signal[n, 0:k, 0:k] )
              for k in range(len( signal )):
                     for l in range(len( signal )): #1d-fwt op de kolommen
                            signal[0:k,m,l] = prev( signal[0:k,m,l] )
       return signal[0:N,0:M,0:O]
```

Bibliografie

[1] Mohamad Akra, Louay Bazzi, *On the solution of linear recurrence equations*. Computational Optimization and Applications, 10(2):195 - 210, 1998.

[2] Bochner S., Chandrasekharan K. *Fourier Transforms*. Princeton University Press. 1949.

[3] `http://djj.ee.ntu.edu.tw/Wavelet_Filter.pdf`.

[4] Tammo Jan Dijkstra, *Adaptive tensor product wavelet methods for solving PDEs*, 2009.

[5] `http://www.uio.no/studier/emner/matnat/math/MAT-INF2360/v12/tensorwavelet.pdf`.

[6] Stéphane Mallat, *A Wavelet Tour of Signal Processing*.

[7] `http://www.ams.org/journals/bull/1960-66-02/S0002-9904-1960-10426-0/S0002-9904-1960-10426-0.pdf`.

[8] Ingrid Daubechies, *Orthonormal Bases of Compactly Supported Wavelets*. AT&T Bell Laboratories, 1988.

[9] `http://www.encyclopediaofmath.org/index.php/Parseval_equality`.

[10] Joachim Weidmann, *Linear operators in Hilbert spaces*, 1980.

[11] Huynh-Thu Q., Ghanbar M. *Scope of validity of PSNR in image/video quality assesment* Electronics Letters, 2008.

[12] Ben Moonen, *Syllabus Topologie*, 2008.

[13] Numpy, `www.numpy.org`.

[14] Scipy, `www.scipy.org`.

[15] Python Image Library, `http://www.pythonware.com/products/pil/`.

[16] Carleson L. *On convergence and growth of partial sums of Fourier series*. Acta Mathematica, 116 (1): 135 157, 1966.

[17] Roland Priemer, *Introductory Signal Processing*, 1991.

[18] Shilov, G. E., and Gurevich, B. L., *Integral, Measure, and Derivative: A Unified Approach*, 1978.

[19] `http://videoprocessing.ucsd.edu/publications/Year_1999/File9.pdf`.

[20] Python Wavelets, `http://www.pybytes.com/pywavelets/ref/signal-extension-modes.html`.