Digitale signaalbewerking

Inhoudsopgave

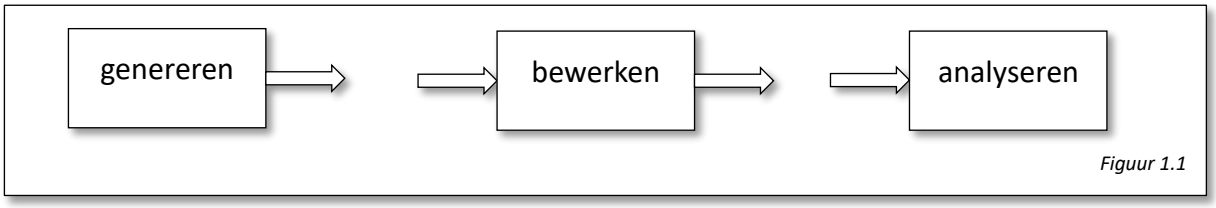
Digitale signaalbewerking 4
Inleiding 4
Tijd- en frequentiedomein 4
Filters 5
Versterking en verzwakking in dB's 6
Complexe getallen 8
Vectoren en matrices 9
Som- en productnotatie 10
Overzicht digitale signaalbewerking 1 11
Analoog en digitaal 13
Inleiding 13
Overzicht DSP-systeem 13
Bemonsteren en kwantiseren 13
Aliasing 14
Bemonsteringstheorema van Shannon-Nyquist 15
Reconstructie 15
Downsampling 16
Upsampling 18
Resampling 19
Undersampling 20
Oversampling 20
Signalen en systemen 21
Signalen 21
Delta- en stapfunctie 23
LTI-Systemen 24
Systeembeschrijvingen 25
Convolutie 27
IIR-filters 28
Fourier transformatie 30
Inleiding 30
Discrete Fourier Transform (DFT) 33
Eigenschappen de Fourier transformatie 34
Analyseren van signalen 34
Analyse en ontwerp van FIR-filters 36

Inleiding 36
Frequentiespectrum van de deltafunctie 36
Frequentiespectrum van een FIR-filter 36
Ontwerp met de Fouriertransformatiemethode 38
Windows 40
Ontwerp met windowed sinc methode 41
Convolutie in tijddomein en in frequentiedomein 41
Short-time Fourier Transform 42
Realisatie van FIR-filters 44
Inleiding 44
Realisatie van FIR-filters 45
Bijlage 1 – Eigenschappen DFT 48
Bijlage 2 – Rechthoek ↔ sinc-functie 49
Bijlage 3 – Voorbeeld implementatie FIR-filter in VHDL 50

Digitale signaalbewerking

Inleiding

Digitale signaalbewerking is het vakgebied dat gaat over het genereren, bewerken en analyseren van digitale signalen. Vaak wordt ook de term *Digital Signal Processing* of *DSP* gebruikt.



Figuur 1.1

Digitale signaalbewerking wordt op heel veel plaatsen toegepast. In vrijwel alle apparaten die iets doen met audio of video wordt het toegepast. Denk daarbij aan tv's, digitale radio's of telefoons. Je kunt ook denken aan apparaten die in een lab gebruikt worden, zoals signaalgeneratoren en oscilloscopen, die eigenlijk niets anders doen dan het genereren, bewerken en analyseren van digitale signalen. Veel apparatuur die in de muziekwereld gebruikt wordt is tegenwoordig digitaal, dus ook daar wordt DSP volop toegepast. Denk aan gitaareffecten, mengpanelen en digitale audio workstations. En wat te denken van communicatieapparatuur, zoals een ADSL- of een kabelmodem? Allemaal DSP!

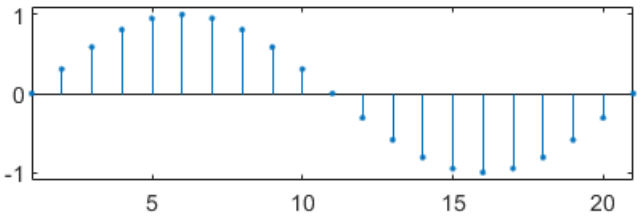
In dit hoofdstuk worden wat algemene onderwerpen behandeld. Voor een groot deel is dit bekende stof, dus vooral bedoeld om voorkennis te activeren.

Tijd- en frequentiedomein

In dit vakgebied gaat het vaak over digitale signalen. Zo'n signaal kun je op verschillende manieren bekijken. Stel dat je een microfoon gebruikt om een audiosignaal op te nemen. De microfoon meet de 'analoge' geluidsdruk en het signaal wordt in een audio-interface omgezet in een 'digitaal' signaal. Dat digitale signaal is niets anders een rij met meetwaarden, ook wel *samples* genoemd, en zou er bijvoorbeeld zo uit kunnen zien:

```
0  0.31  0.59  0.81  0.95  1.00  0.95  0.81  0.59  0.31
0 -0.31 -0.59 -0.81 -0.95 -1.00 -0.95 -0.81 -0.59 -0.31  0
```

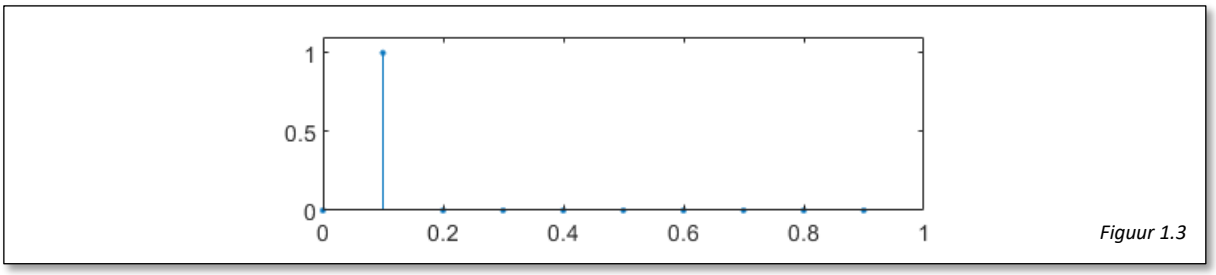
Als deze waarden grafisch worden uitgezet, dan ziet dat er als volgt uit.



Figuur 1.2

Op de verticale as staat de amplitude van het signaal, ofwel de samplewaarden. Op de horizontale as staan de samplenummers, in dit voorbeeld lopen die van 1 t/m 21. In veel gevallen zijn samples de meetwaarden die in de loop van de tijd worden genomen. De horizontale as is dan eigenlijk een tijd-as. We bekijken het signaal hier dus in het *tijddomein*.

Je kunt ook op een andere manier naar dit signaal kijken. Het is een sinusvormig signaal en een sinus heeft precies één frequentie. Als we in het *frequentiedomein* naar het signaal kijken, dan ziet het er als volgt uit.



Figuur 1.3

Op de verticale as staat weer de amplitude van het signaal. Op de horizontale as staat nu de frequentie. Er is precies één piek te zien, hetgeen betekent dat het signaal precies één frequentie heeft. Als het signaal bijvoorbeeld zou bestaan uit twee sinussen, dan zouden er in het frequentiedomein twee pieken te zien zijn. De plaats van de piek (in dit geval bij 0.1) geeft aan om welke frequentie het gaat. In dit geval is de frequentie één-tiende van de halve samplefrequentie, maar daar komen we later nog op terug.

De eenheid van frequentie is *hertz*, vaak afgekort tot *Hz* (let op het juiste gebruik van hoofd- en kleine letters). Eén hertz betekent één periode per seconde. Als de frequentie bijvoorbeeld 100 Hz is, dan wil dat zeggen dat het signaal zich 100 keer per seconde herhaalt.

Er wordt voor de grootheid frequentie ook wel een andere eenheid gebruikt, namelijk *rad/s*. Je kunt daarbij denken aan een ronddraaiende vector. Elke omwenteling van 360° of 2π rad is één periode. De snelheid waarmee de vector ronddraait, is de verandering van de hoek per tijdseenheid. Er wordt dan ook wel de term *hoekfrequentie* gebruikt.

Filters

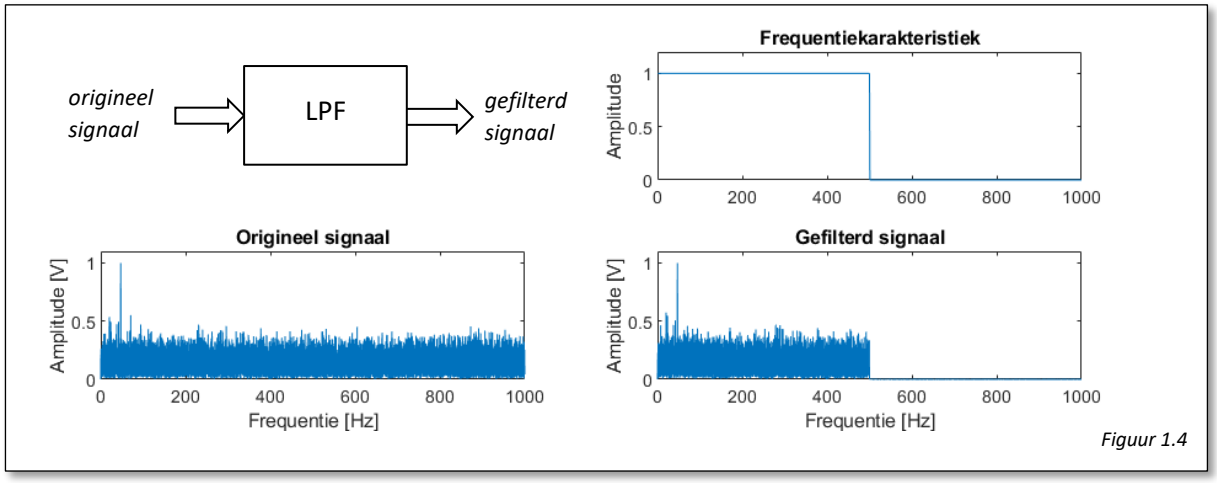
Een veelgebruikte toepassing van digitale signaalbewerking is een *digitaal filter*. Een digitaal filter is een systeem dat van een signaal bepaalde frequentiegebieden kan versterken of verzwakken. Filters die vaak voorkomen zijn:

- Laagdoorlaatfilter (Low Pass Filter, LPF)
- Hoogdoorlaatfilter (High Pass Filter, HPF)
- Banddoorlaatfilter (Band Pass Filter, BPF)
- Bandstopfilter (Band Stop Filter, BSF)

Een laagdoorlaatfilter laat lagere frequenties door en verzwakt de hogere frequenties in het signaal. In figuur 1.4 zie je de frequentiekarakteristiek van een laagdoorlaatfilter dat frequenties tot 500 Hz doorlaat.

In het voorbeeld is te zien dat er in het gefilterd signaal geen frequenties van hoger dan 500 Hz meer voorkomen. In de praktijk zijn de gefilterde frequenties niet helemaal uit het signaal verdwenen, maar zullen ze dermate verzwakt zijn, dat ze niet meer te zien zijn in het plaatje. In een audio signaal zullen deze frequenties niet of nauwelijks hoorbaar zijn.

Naast de hier genoemde standaard filters, bestaan er nog andere soorten. Er kunnen filters gemaakt worden met elke gewenste frequentiekarakteristiek. Systemen die bijvoorbeeld signalen versterken, differentiëren of integreren kunnen ook als filter beschouwd worden. Voor het gemak wordt als voorbeeld vaak een laagdoorlaatfilter gekozen.



Figuur 1.4

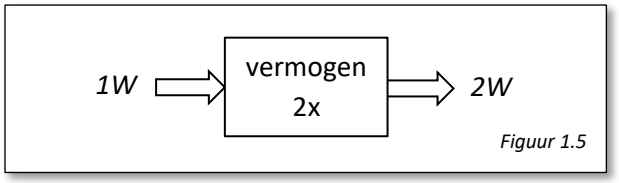
Versterking en verzwakking in dB's

In de techniek wordt de term *dynamisch bereik* gebruikt om de verhouding aan te geven tussen de kleinst mogelijke signaalverandering en de grootte van het signaal. Een signaal dat bijvoorbeeld een amplitude heeft van 10V, en in stapjes van 1 µV kan variëren, heeft een groot dynamisch bereik. Het is vaak lastig om signalen met een groot dynamisch bereik grafisch goed weer te geven. Er is dan namelijk een hele hoge resolutie nodig.

Er wordt daarom vaak met een logaritmische schaal gewerkt. Vaak wordt daarvoor de eenheid *decibel* gebruikt. De afkorting is *dB*. Let weer op het juiste gebruik van kleine letters en hoofdletters. Een decibel geeft een verhouding weer tussen twee vermogens, op een logaritmische schaal. Stel dat we een systeem hebben dat het vermogen van een signaal 2x versterkt, dan kan dat als volgt in dB worden uitgedrukt.

$$10 \cdot \log_{10} \frac{P_{uit}}{P_{in}} = 10 \cdot \log_{10} \frac{2}{1} \approx 3\ dB$$

De vermogensversterking is +3 dB.



Figuur 1.5

Het komt vaak voor dat er wordt uitgegaan van de versterking of verzwakking van de spanning, in plaats van het vermogen van een signaal. Stel dat een systeem de spanning van een signaal 2x versterkt. In dat geval ziet de berekening er als volgt uit. Voor het gemak gaan we ervan uit dat de in- en uitgangs-weerstanden dezelfde waarde hebben.

$$10 \cdot \log_{10} \frac{P_{uit}}{P_{in}} = 10 \cdot \log_{10} \frac{U_{uit}^2}{U_{in}^2} = 20 \cdot \log_{10} \frac{U_{uit}}{U_{in}} = 20 \cdot \log_{10} \frac{2}{1} \approx 6\ dB$$

Bij het berekenen van dB's is het dus van belang om te weten of het om het vermogen van een signaal gaat, of om de spanning. Hiernaast staat een tabel met een voorbeelden van dB-waarden met bijbehorende vermogens- en spannings-versterking. Een verzwakking wordt uitgedrukt met een negatieve dB-waarde.

Versterking in dB	Vermogen	Spanning
10 dB	10x	3.2x
6 dB	4x	2x
3 dB	2x	1.4x
0 dB	1x	1x
-3 dB	0.5x	0.7x
-6 dB	0.25x	0.5x
-10 dB	0.10x	0.3x

Matlab heeft een functie *log*, maar let op, dat is het natuurlijke logaritme dat wij kennen als *ln*. Om een dB-waarde te berekenen, kun je de functie *log10* gebruiken. Stel dat je het aantal dB's wil berekenen bij een vermogensversterking van 2x:

```
A=2;              % vermogensversterking
10*log10(A)       % geeft de waarde 3.0103, dus ongeveer 3 dB
```

Stel dat je het aantal dB's wil berekenen bij een spanningsversterking van 1000x:

```
A=1000;           % spanningsversterking
20*log10(A)       % geeft de waarde 60, dus precies 60 dB
```

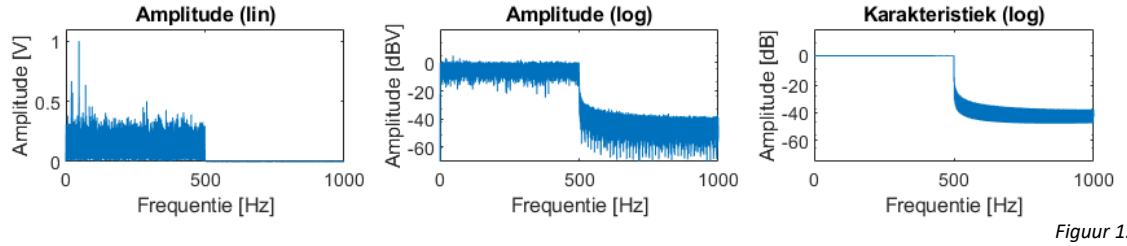
Het terugrekenen van dB's naar versterking gaat als volgt. Stel dat het aantal dB's gelijk is aan x en dat we de vermogensversterking P noemen en de spanningsversterking Q.

$$x = 10 \cdot \log_{10} P \quad \rightarrow \quad \frac{x}{10} = \log_{10} P \quad \rightarrow \quad P = 10^{x/10} \qquad\qquad Q = 10^{x/20}$$

In Matlab kan machtsverheffen met de operator ^ gedaan worden, of met de functie *power*.

```
x=25;             % versterking in dB
10^(x/10)         % geeft de vermogensversterking
power(10,x/20)    % geeft de spanningsversterking
```

In figuur 1.4 is een gefilterd signaal te zien, waarbij voor de amplitude een lineaire schaal is gebruikt. Hieronder is hetzelfde signaal nog een keer weergegeven, met zowel een lineaire als een logaritmische schaalverdeling. De hoge frequenties van het signaal zijn door het laagdoorlaatfilter met ongeveer 40 dB verzwakt en dat komt overeen met een verzwakking van ongeveer 100x. Op de lineaire schaal zijn deze kleine amplitudes niet te zien.



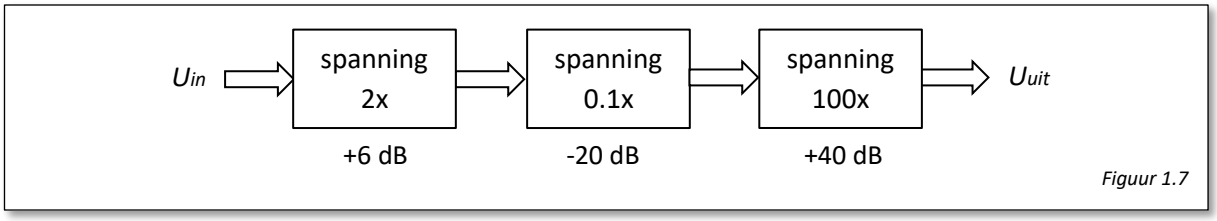
Figuur 1.6

Een decibel geeft in principe altijd de verhouding tussen twee getallen weer. Soms wordt voor één van de getallen een vaste waarde gekozen, zodat we de andere waarde in dB's kunnen uitdrukken.

Als we als referentiewaarde bijvoorbeeld 1 mW kiezen, dan kan het vermogen van een signaal uitgedrukt worden in de eenheid *dBm*. Kiezen we als referentie 1 V, dan kan de spanning uitgedrukt worden in *dBV*.

$$10 \cdot \log_{10} \frac{P}{1\,mW}\ [dBm] \qquad\qquad 20 \cdot \log_{10} \frac{U}{1\,V}\ [dBV]$$

Om de versterking te berekenen van een keten van systemen, kunnen de versterkings*factoren* met elkaar *vermenigvuldigd* worden. Als met dB's wordt gewerkt, dan kunnen de aantallen dB's *opgeteld* worden.



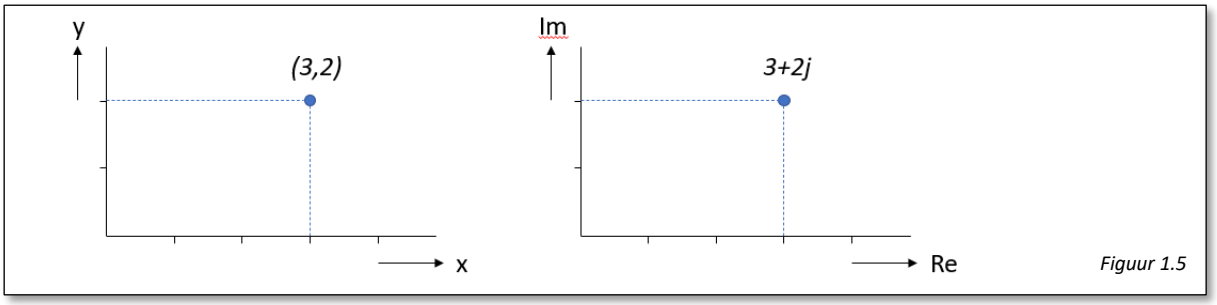
Figuur 1.7

De versterking van de keten in bovenstaand systeem is 2 · 0.1 · 100 = 20x. Stel dat de waarde van het ingangssignaal 10 V is, dan zal de waarde van het uitgangssignaal 200 V zijn.
In dB's uitgedrukt is de versterking 6 – 20 + 40 = 26 dB. Het ingangssignaal heeft een waarde van 20 dBV. Het uitgangssignaal is dan 20 dBV + 26 dB = 46 dBV, hetgeen overeenkomt met een waarde van 200 V.

Als laatste nog een voorbeeld met een vermogensberekening. Stel dat een signaal met een vermogen van 1 mW door een systeem 100x versterkt wordt. Het vermogen van het uitgangssignaal is dan 100 mW.
In dB's uitgedrukt hebben we het dan over een ingangssignaal van 0 dBm dat met 20 dB wordt versterkt. Het uitgangssignaal is dan 0 dBm + 20 dB = 20 dBm, hetgeen overeenkomt met een waarde van 100 mW.

Complexe getallen

Bij digitale signaalbewerking worden vaak complexe getallen gebruikt, daarom even een korte opfrisser. De term *complex* wordt niet gebruikt om aan te geven dat het een 'moeilijk' getal, maar een getal waar twee waarden in verstopt zitten. Net zoals je een locatie in een tweedimensionaal assenstelsel kunt aangeven met coördinaten x en y, kun je een locatie in het complexe vlak aangeven met een complex getal. Zo'n getal bestaat uit een *reëel* deel en een *imaginair* deel. Het imaginaire deel wordt aangegeven met een *i* of een *j*. In de wiskunde wordt meestal een *i* gebruikt, maar in de techniek wordt vaak een *j* gebruikt, omdat het symbool *i* al voor stroom wordt gebruikt.



Figuur 1.5

Het is gemakkelijk om te rekenen met complexe getallen. Stel we hebben twee complexe getallen:

$$p = 1 + 2j \qquad q = 3 + 4j$$

Bij optellen kunnen de reële en imaginaire delen afzonderlijk opgeteld worden:

$$p + q = 1 + 2j + 3 + 4j = 4 + 6j$$

Bij vermenigvuldigen kunnen 'haakjes weggewerkt worden', waarbij je aanneemt dat $j^2 = -1$:

$$p \cdot q = (1 + 2j)(3 + 4j) = 3 + 4j + 6j + 8j^2 = -5 + 10j$$

De *geconjugeerde* van een complex getal is een getal met dezelfde reële waarde, maar met de omgekeerde (negatieve) imaginaire waarde. Een geconjugeerde wordt aangegeven met een sterretje *.

Als $p = 1 + 2j$ dan is $p^* = 1 - 2j$.

Bij het delen van complexe getallen is het handig om zowel de teller als de noemer te vermenigvuldigen met de geconjugeerde van de noemer. Onder de deelstreep komt dan uiteindelijk een reële waarde te staan.

$$\frac{p}{q} = \frac{1+2j}{3+4j} = \frac{1+2j}{3+4j} \cdot \frac{3-4j}{3-4j} = \frac{3-4j+6j-8j^2}{9-12j+12j-16j^2} = \frac{11+2j}{25} = \frac{11}{25} + \frac{2}{25}j$$

Een locatie in een tweedimensionaal assenstelsel kun je aangeven met x- en y-coördinaten. Dat worden wel *cartesische coördinaten* genoemd. Je kunt de locatie ook aangeven met *poolcoördinaten*: de afstand van het punt tot de oorsprong van het assenstelsel en de hoek van de lijn door de oorsprong en het punt. Dat is bij complexe getallen precies hetzelfde.

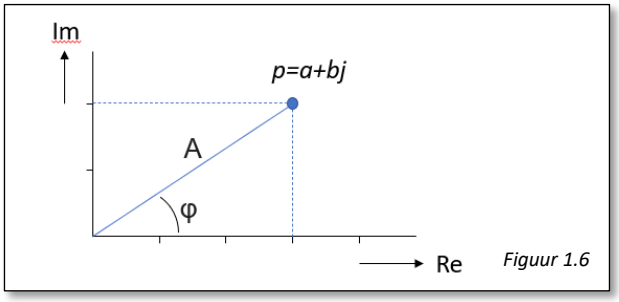
Gegeven een complex getal $p = a + bj$, dus *a* is het reële deel en *b* is het imaginaire deel:

$$a = Re(p) \qquad b = Im(p)$$

Er geldt dan het volgende:

$$A = \sqrt{a^2 + b^2} \qquad \varphi = \arctan\frac{b}{a}$$

$$a = A \cdot \cos\varphi \qquad b = A \cdot \sin\varphi$$



Figuur 1.6

De lijn door de oorsprong en de betreffende locatie wordt ook wel gezien als een vector. De lengte van deze vector (*A*) is altijd positief en wordt ook wel de *absolute waarde* genoemd, met als symbool $|A|$. In de signaalbewerking wordt deze lengte ook wel de *amplitude* genoemd. De hoek φ wordt dan de *fase* genoemd.

In Matlab kun je gemakkelijk met complexe waarden werken. Matlab herkent zowel de *i* als de *j* om het imaginaire deel van een complex getal aan te duiden:

```
p=1+2i;
q=3+4j;
p*q         % Vermenigvuldiging van p en q levert -5.0000 +10.0000i
```

Vectoren en matrices

Bij signaalbewerking worden vaak vectoren en matrices gebruikt om met signalen te rekenen. Daarom even een korte opfrisser.

Een gewoon getal, bijvoorbeeld het getal 5, wordt wel een *scalar* genoemd.

Een rij met getallen wordt een *vector* genoemd. Er bestaan rijvectoren, waarin de getallen in één rij achter elkaar staan. Er bestaan ook kolomvectoren, waarin de getallen in een kolom onder elkaar staan. Een rijvector heeft één afmeting, namelijk de lengte van de vector, ofwel het aantal getallen in de vector. Je zou kunnen zeggen dat een rijvector ééndimensionaal is.

Als een groep getallen meer dan één dimensie heeft, dan wordt gesproken over een *matrix*. Het meervoud van matrix is *matrixen* of *matrices*. Een tweedimensionale matrix kun je zien als een verzameling getallen die in een rechthoekig schema georganiseerd zijn. De rechthoek heeft twee afmetingen, namelijk het aantal rijen en het aantal kolommen. Een matrix kan ook meer dimensies hebben. Een driedimensionale matrix ziet er bijvoorbeeld uit als een kubus, en heeft drie afmetingen.

Je zou kunnen zeggen dat een rijvector een matrix is met afmetingen *[1 n]*, ofwel met één rij en *n* kolommen. Een kolomvector is dan een matrix met afmetingen *[n, 1]*. Een scalar kun je zien als een matrix met afmetingen *[1 1]*.

In Matlab kun je gemakkelijk scalars, vectoren en matrices aanmaken:

```
a=5;                        % een scalar
b=[1 2 3];                  % een rijvector met 3 kolommen
c=[1; 2; 3];                % een kolomvector met 3 rijen
d=[1 2 3 4; 5 6 7 8];       % een matrix met 2 rijen en 4 kolommen
```

De matrices in Matlab (en dus ook de scalars en vectoren) kunnen allerlei type waarden bevatten, zoals integers, floating point waarden of complexe getallen.

Bij afmetingen van (of locaties binnen) een matrix, wordt in Matlab altijd eerst de rij en daarna de kolom aangegeven. De nummering van de elementen van een matrix begint in Matlab bij 1.

```
d=[1 2 3 4; 5 6 7 8];       % een matrix met 2 rijen en 4 kolommen
```

Het commando `size(d)` geeft als antwoord `[2 4]`, omdat er twee rijen en vier kolommen zijn. Het commando `length(d)` is hetzelfde als `max(size(d))` en geeft de grootste afmeting van de matrix.

```
d(2,3)                      % geeft de waarde 7 (2e rij, 3e kolom)
```

Som- en productnotatie

Het is vaak nodig om alle getallen van een vector bij elkaar op te tellen. Neem als voorbeeld de volgende vector (Matlab notatie):

$$x = [3\ 5\ 7\ 6\ 4\ 2]$$

Om aan te geven dat je alle getallen van deze vector bij elkaar wilt optellen, kunnen onder andere de volgende notaties worden gebruikt:

$$\sum x \qquad \sum x[n] \qquad \sum_{n} x[n]$$

Het symbool is de Griekse hoofdletter *sigma* (met de S van som). In dit geval is de som van alle elementen van x gelijk aan $3 + 5 + 7 + 6 + 4 + 2 = 27$.

Er kan ook worden aangegeven dat bepaalde elementen van de vector opgeteld moeten worden:

$$\sum_{i=1}^{3} x = 3 + 5 + 7 = 15$$

Boven en onder het symbool staan de indices weergeven van de elementen die moeten worden opgeteld. In dit voorbeeld gaat het om de elementen 1 tot en met 3. Als de nummering van de elementen bij 1 start, dan gaat het dus om de eerste drie elementen van *x*, met de waarden 3, 5 en 7.

Er is een vergelijkbare schrijfwijze om het product van de elementen van een vector te beschrijven:

$$\prod_{i=1}^{6} x = 3 \cdot 5 \cdot 7 \cdot 6 \cdot 4 \cdot 2 = 5040$$

In dit geval wordt de Griekse hoofdletter *pi* gebruikt (met de P van product).

In Matlab zijn verschillende functies beschikbaar om met vectoren te rekenen.

```
x=[3 5 7 6 4 2];
sum(x)            % geeft de som van de elementen, ofwel 27
mean(x)           % geeft het gemiddelde van de elementen, ofwel 4.5
prod(x)           % geeft het product van de elementen, ofwel 5040
max(x)            % geeft de maximale waarde van de elementen, ofwel 7
min(x)            % geeft de minimale waarde van de elementen, ofwel 2
```

Om in Matlab een deel van een vector aan te geven wordt de volgende notatie gebruikt.

```
sum(x(1:3))       % geeft de som van elementen 1 t/m 3 van vector x
```

Een losse dubbele punt wordt gebruikt om een hele rij, kolom (of zelfs een hele vector of matrix) aan te duiden.

```
y=[1 2 3 4; 5 6 7 8];
sum(y(2,:))       % geeft de som van de elementen van de tweede rij,
                  % alle kolommen, ofwel 5+6+7+8=26
sum(y(:))         % geeft de som van alle elementen in de matrix
```

Overzicht digitale signaalbewerking 1

Het einddoel van het vak Digitale signaalbewerking 1 is dat je een zogenaamd FIR-filter kunt ontwerpen met elke gewenste doorlaatkarakteristiek. Een FIR-filter is een eenvoudig filter zonder terugkoppeling.

Als eerste kijken we naar de omzetting van signalen van ‘analoog’ naar ‘digitaal’ en andersom. We hebben het dan over termen als sampling, aliasing, kwantisatie, downsampling, upsampling, resampling, oversampling en undersampling.

Vervolgens worden verschillende eigenschappen van signalen en systemen onder de loep genomen.

We kijken ook naar de omzetting van signalen van het tijddomein naar het frequentiedomein en andersom. Daarvoor gebruiken we de Fourier transformatie.

En uiteraard worden de verschillende soorten filters behandeld, en kijken we naar een gestructureerde manier om een FIR-filter te ontwerpen.

Matlab wordt intensief gebruikt, zowel om signalen te genereren, te bewerken en te analyseren, als om filters te ontwerpen.

Matlab heeft een help- en een doc-functie om snel informatie over functies te krijgen.

```
help sum          % Geeft een helptekst over sum in het command window
doc sum           % Opent een browser window met info over sum
```

Analoog en digitaal

Inleiding

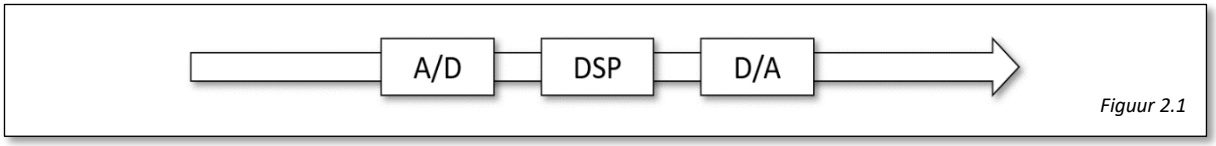
Binnen de wereld van de techniek worden vaak termen als *analoog* en *digitaal* gebruikt. We weten allemaal wel wat daar mee bedoeld wordt. Toch?

Als we het over een analoog signaal hebben, dan bedoelen we een signaal dat *continu in amplitude* is, én *continu in tijd*. Dat betekent dat het signaal binnen een gegeven bereik élke amplitude kan hebben. Ook kan het signaal op élk moment in de tijd een andere waarde hebben.

Daarnaast zijn er signalen die *discreet* zijn in amplitude. Deze signalen kunnen dus maar een beperkt aantal waarden voor de amplitude hebben. Signalen kunnen ook *tijddiscreet* zijn. Dat betekent dat ze slechts op een beperkt aantal momenten in de tijd een andere waarde kunnen krijgen. Als we het over een digitaal signaal hebben, dan bedoelen we meestal een signaal dat discreet is, zowel in tijd als in amplitude. In de wereld van de digitale signaalbewerking wordt dan ook vaak over *discrete* signalen gesproken.

Overzicht DSP-systeem

Stel dat we een analoog signaal willen bewerken. Het signaal wordt door een A/D-converter omgezet in een digitaal signaal, zodat het door de DSP bewerkt kan worden. Na bewerking kan het signaal door een D/A-converter weer omgezet worden naar een analoog signaal.



Figuur 2.1

De DSP kan een normale microprocessor of microcontroller zijn, maar vaak wordt een processor gebruikt die speciaal bedoeld is voor het bewerken van digitale signalen: een *Digital Signal Processor* of *DSP*. Bij het bewerken van signalen zijn veel vermenigvuldigingen en optellingen nodig. Een DSP heeft naast een normale ALU, meerdere zogenaamde Multiply-Accumulate Units (*MAC*'s) aan boord, zodat er meerdere vermenigvuldigingen en optellingen tegelijkertijd uitgevoerd kunnen worden. Daarnaast hebben DSP's vaak ondersteuning voor het gebruik van circulaire buffers, waardoor gemakkelijker (en sneller) met rijen getallen gewerkt kan worden. Overigens is het ook mogelijk om een FPGA te gebruiken om digitale signalen te bewerken.

Bemonsteren en kwantiseren

Om van een analoog signaal een digitaal signaal te maken, moet het analoge signaal eerst bemonsterd en daarna gekwantiseerd worden. Bij het *bemonsteren* (of *samplen*) wordt de amplitude van het signaal herhaaldelijk gemeten, op regelmatige tijdstippen. Het tijdsinterval tussen twee samples is de *sample-periode* (T_s). De frequentie waarmee gesampled wordt is de *samplefrequentie* ($f_s = 1/T_s$). Door het bemonsteren wordt het analoge signaal *tijddiscreet* gemaakt.

Bij het *kwantiseren* wordt het signaal *amplitudediscreet* gemaakt. Het digitale signaal kan nog maar een beperkt aantal amplitudewaarden hebben. Het aantal mogelijke waarden hangt samen met het aantal bits dat gebruikt wordt om elk amplitudeniveau te coderen. Als er bijvoorbeeld 8 bits/sample gebruikt worden, dan zijn 2^8 ofwel 256 niveaus mogelijk. En als er bijvoorbeeld 1000 niveaus nodig zijn, dan zijn er $^2\log 1000$ ofwel 10 bits/sample nodig.

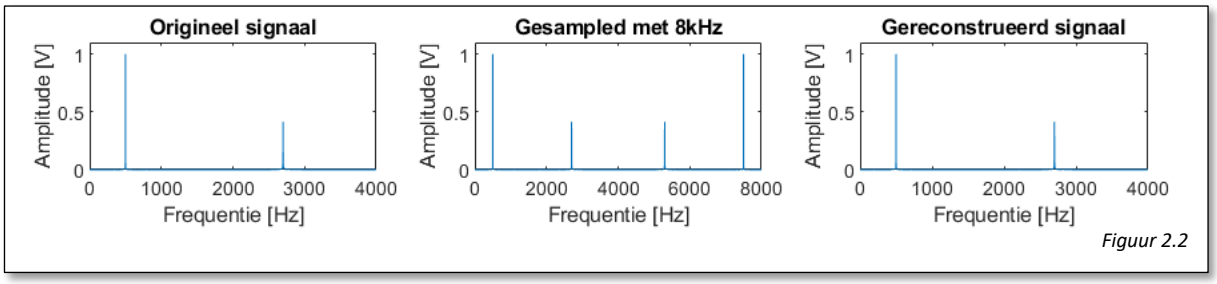
Bij het kwantiseren moet bij elk sample de waarde van de amplitude afgerond worden naar de waarde van het dichtstbijzijnde niveau. De afrondingsfouten die hierdoor ontstaan, zijn bij het ene sample klein en bij een ander sample wat groter. De maximale afwijking is de helft van het verschil tussen twee amplitudeniveaus. De afrondingsfouten hebben een onvoorspelbaar en willekeurig karakter en hebben de eigenschappen van een ruissignaal. Er wordt ook wel gesproken over *kwantisatieruis*. Hoe meer amplitudeniveaus er zijn, des te kleiner de afrondingsfouten en des te lager het niveau van de kwantisatieruis.

Aliasing

Door het bemonsteren verandert het frequentiespectrum van het signaal. In het tijddiscrete signaal herhalen alle frequentiecomponenten van het analoge signaal zich rondom veelvouden van de samplefrequentie.

Stel dat een signaal met een sinus van 500 Hz wordt gesampled met een samplefrequentie f_s van 8000 Hz. In het bemonsterde (tijddiscrete) signaal zijn dan frequentiecomponenten te vinden met frequenties $n \cdot f_s \pm 500\ [Hz]$ met $n \in \mathbb{Z}$. Er zitten dus componenten in het signaal met frequenties zoals 500, 7500, 8500, 15500, 16500 Hz. Er zitten ook componenten in met negatieve frequenties, zoals -500 en -7500 Hz.

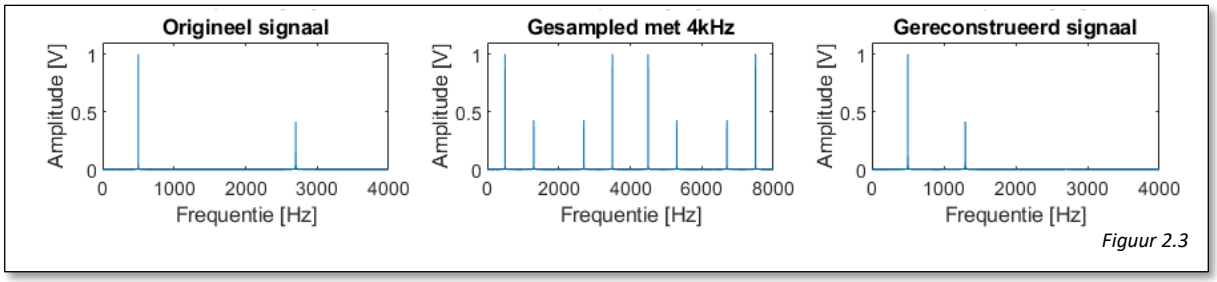
Als een signaal bemonsterd wordt, dan is het de bedoeling dat het signaal zelf niet verandert. Dat betekent dat we vanuit het tijddiscrete signaal het originele signaal weer zouden moeten kunnen reconstrueren, bijvoorbeeld door middel van een laagdoorlaatfilter. Dat lukt alleen als de samplefrequentie hoog genoeg is, namelijk minimaal tweemaal de bandbreedte van het analoge signaal. Dit wordt wel het bemonsteringstheorema van Shannon en Nyquist genoemd.



Figuur 2.2

In bovenstaand figuur wordt een signaal met frequentiecomponenten van 500 Hz (amplitude 1 V) en 2700 Hz (amplitude 0.5V) bemonsterd met een samplefrequentie van 8000Hz. Deze samplefrequentie is hoger dan de bandbreedte van het signaal, dus het originele signaal kan goed gereconstrueerd worden vanuit het bemonsterde signaal.

Als de samplefrequentie niet hoog genoeg is, dan vallen frequentiecomponenten die door het bemonsteren zijn ontstaan, binnen het frequentiegebied van het originele signaal. In dat geval kan het originele signaal niet meer gereconstrueerd worden uit het bemonsterde signaal. We spreken in dat geval van *aliasing*.



Figuur 2.3

In figuur 2.3 wordt hetzelfde signaal, met frequentiecomponenten van 500 en 2700 Hz, gesampled met een samplefrequentie van 4000 Hz. Deze frequentie is lager dan tweemaal de hoogste frequentie die in het signaal voorkomt, dus er ontstaat aliasing.

De frequentiecomponent van 2700 Hz wordt gespiegeld rondom de samplefrequentie van 4000 Hz. Daardoor ontstaat een component bij de frequentie 4000 – 2700 = 1300 Hz.

Om het originele signaal uit het bemonsterde signaal terug te winnen, wordt een laagdoorlaatfilter met een kantelfrequentie die de helft is van de samplefrequentie, ofwel 2000 Hz. De frequentiecomponent van 2700 Hz wordt door dit filter tegengehouden, maar de component bij 1300 Hz wordt doorgelaten. Het gereconstrueerde signaal bevat nu de componenten bij 500 en 1300 Hz.

Bemonsteringstheorema van Shannon-Nyquist

Het bemonsteringstheorema van Nyquist-Shannon is de stelling dat wanneer een analoog signaal naar een tijddiscreet signaal wordt geconverteerd, de bemonsteringsfrequentie minstens tweemaal zo hoog moet zijn als de bandbreedte van het signaal, om het originele signaal precies te kunnen reconstrueren vanuit het bemonsterde signaal.

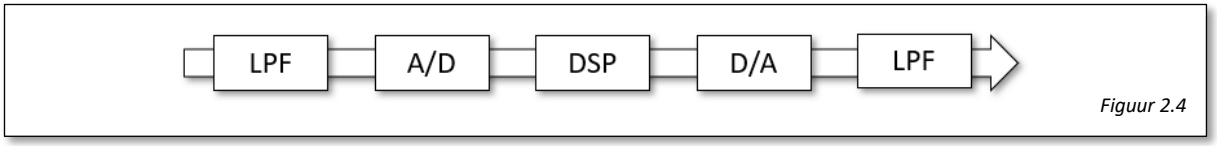
Het theorema is ooit bewezen door Claude Shannon en Harry Nyquist. De halve samplefrequentie wordt ook wel de *Nyquistfrequentie* genoemd.

Als de samplefrequentie niet hoog genoeg is, dan ontstaat aliasing, en dat is in de meeste gevallen een ongewenst verschijnsel. Om te zorgen dat er geen aliasing kan ontstaan, wordt een analoog signaal meestal gefilterd voordat het aan de A/C-converter wordt aangeboden. Het filter wordt wel een *anti-aliasing-filter* genoemd, en houdt alle frequenties hoger dan de halve samplefrequentie tegen.

Reconstructie

Na het bewerken van het digitale signaal is het vaak nodig om er weer een analoog signaal van te maken. Het discrete signaal is alleen gedefinieerd op de samplemomenten. Tussen twee samples is het signaal niet gedefinieerd. Als het signaal wordt aangeboden aan een D/A-converter, dan gaat deze ervan uit dat het uitgangssignaal de laatste samplewaarde moet krijgen, totdat de volgende samplewaarde wordt aangeboden. Daardoor ontstaat een soort ‘trapjessignaal’, met oneindig steile flanken op de momenten dat er een nieuw sample wordt aangeboden. Deze flanken zaten natuurlijk niet in het originele signaal.

Om van het ‘trapjessignaal’ weer een vloeiend signaal te maken, wordt een laagdoorlaatfilter toegepast. Dit filter wordt wel een *reconstructiefilter* genoemd, en zorgt ervoor dat het gereconstrueerde signaal exact gelijk is aan het originele signaal (op de kwantiseringsruis na).

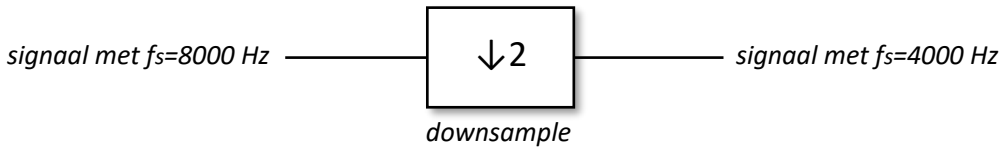


Figuur 2.4

Het laagdoorlaatfilter (LPF) voor in de keten is bedoeld om aliasing tegen te gaan. Het laagdoorlaatfilter achter in de keten is het reconstructiefilter.

Downsampling

De bedoeling van downsampling is om de samplefrequentie van een digitaal signaal te verlagen, zonder de informatie van het signaal te veranderen. Als het bijvoorbeeld om een audiosignaal gaat, dan is het de bedoeling dat de audio na het downsamplen nog hetzelfde klinkt.

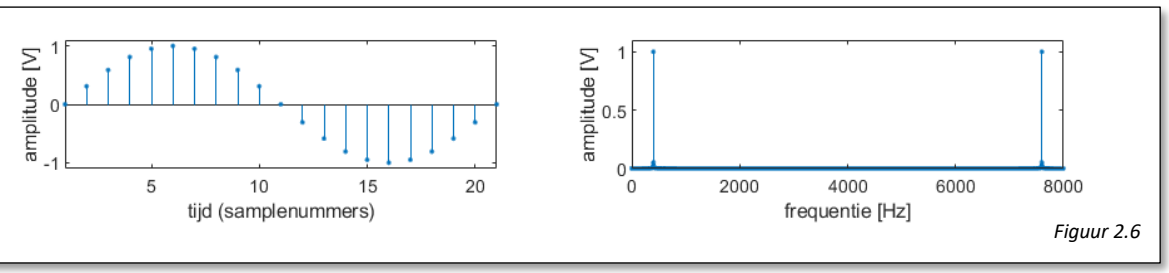


Figuur 2.5

Neem als voorbeeld een muziekfragment. Als bij het afspelen alleen de samplefrequentie wordt verlaagd, dan wordt de muziek langzamer afgespeeld en klinkt alles het lager. Dat is géén downsampling.

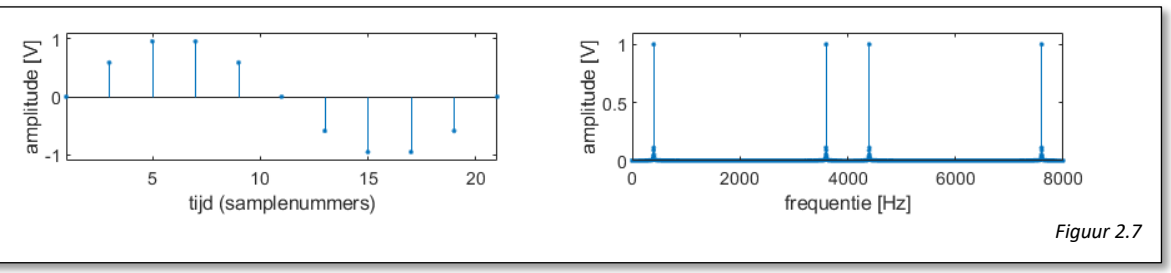
Bij downsampling wordt de samplefrequentie verlaagd én worden samples weggegooid. Gegeven een digitaal signaal met een duur van 1 seconde en een samplefrequentie van 8000 Hz. Het signaal heeft 8000 samples. Stel dat het signaal gedownsampled wordt met een factor 2. De samplefrequentie wordt dan 4000 Hz. Het signaal moet na het downsamplen nog steeds 1 seconde duren en kan dus maar 4000 samples hebben. De helft van de samples moet dus worden weggegooid.

Gegeven een sinusvormig signaal met een frequentie van 400 Hz en een samplefrequentie van 8000 Hz.



Figuur 2.6

Het signaal wordt gedownsampled met een factor 2.

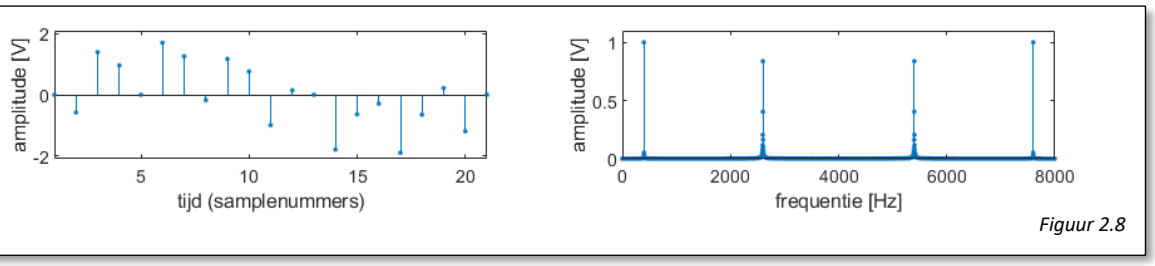


Figuur 2.7

Het gedownsamplede signaal is nog steeds een digitaal signaal en het frequentiespectrum herhaalt zich dus rondom de (nieuwe) samplefrequentie. In bovenstaand voorbeeld is de samplefrequentie na het downsamplen 4000 Hz en in het frequentiespectrum is te zien dat het spectrum zich inderdaad herhaalt rondom die frequentie. Door het downsamplen zijn er frequentie-componenten van 3600 en 4400 Hz ontstaan.

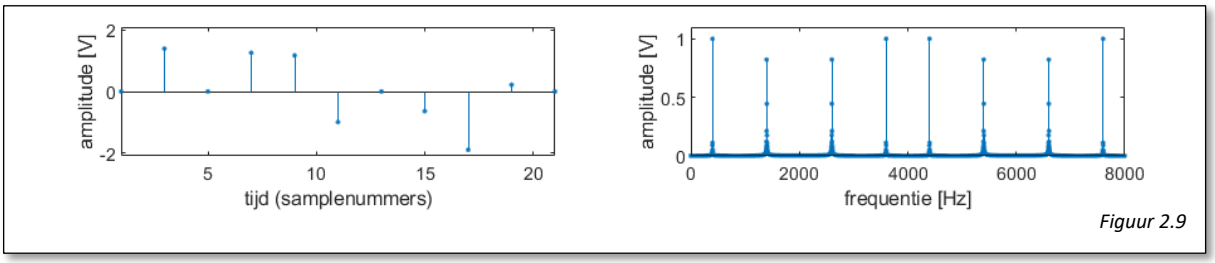
Als het signaal weer analoog gemaakt zou worden, dan wordt het gefilterd met een reconstructiefilter met een kantelfrequentie van 2000 Hz, ofwel de helft van de nieuwe samplefrequentie. Als we het analoge signaal zouden afspelen, dan zou er gewoon een toon van 400 Hz te horen zijn. In dit geval is er dus geen probleem, omdat de nieuwe samplefrequentie nog steeds hoog genoeg is om aliasing te voorkomen.

Stel dat we een signaal hebben met frequentiecomponenten van 400 en 2700 Hz bij een sample-frequentie van 8000 Hz.



Figuur 2.8

In het tijddomein zijn de twee sinusvormige componenten niet meer zo duidelijk te zien, maar in het frequentiedomein wel. Als we dit signaal downsamplen met een factor 2, dan krijgen we het volgende.

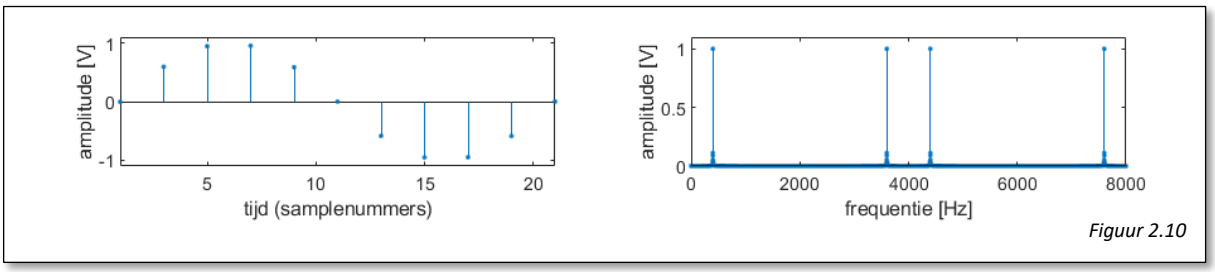


Figuur 2.9

Als dit signaal analoog wordt gemaakt en wordt afgespeeld, dan zijn tonen van 400 en 1300 Hz te horen. De frequentiecomponent van 2700 Hz is niet meer te horen, omdat deze door het reconstructiefilter wordt tegengehouden. De toon van 1300 Hz ontstaat door aliasing van de component van 2700 Hz. Deze wordt gespiegeld rondom de nieuwe samplefrequentie: 4000 – 2700 = 1300 Hz.

Om aliasing bij downsampling te voorkomen, kan het signaal vóórdat het gedownsampled wordt, eerst gefilterd worden. Er kan een digitaal filter worden gebruikt om alle frequenties van 2000 Hz en hoger tegen te houden. Deze combinatie van filteren en downsamplen wordt *decimeren* genoemd.

Als het signaal met componenten van 400 en 2700 Hz gedecimeerd wordt, krijgen we het volgende.



Figuur 2.10

De component van 2700 Hz is vóór het downsamplen al uit het signaal gefilterd, waardoor er geen aliasing ontstaat. Alleen de toon van 400 Hz is nog overgebleven. Het signaal is dus door het decimeren wél veranderd, maar er is geen sprake van aliasing.

Matlab heeft functies voor downsamplen en decimeren (Signal Processing Toolbox).

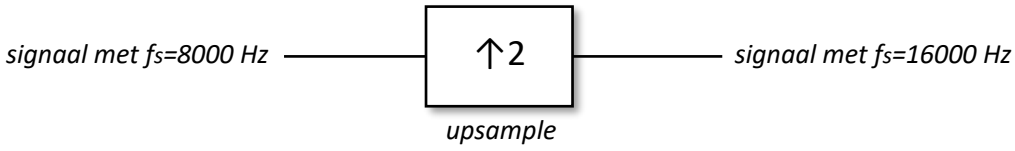
```
x=[1 2 3 4 5 6 7 8 9 10];
xd=downsample(x,3)      % geeft een vector [1 4 7 10]
```

De functie `decimate` heeft een signaal met minimaal 25 samples nodig.

```
xd=decimate(1:25,5)     % geeft een vector [1 6 11 16 21]
```

Upsampling

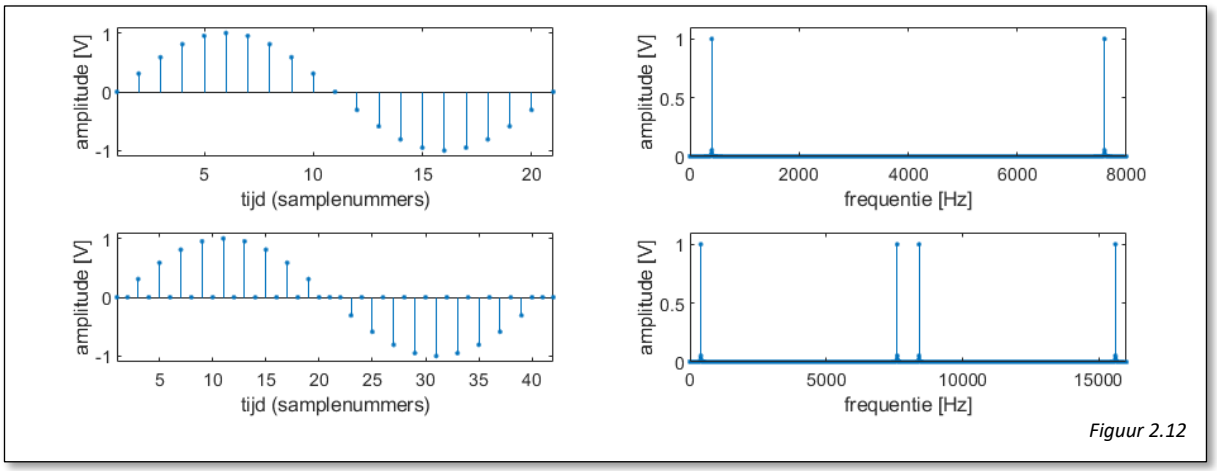
De bedoeling van upsampling is om de samplefrequentie van een digitaal signaal te verhogen, zonder de informatie van het signaal te veranderen. Als het bijvoorbeeld om een audiosignaal gaat, dan is het de bedoeling dat de audio na het upsamplen nog hetzelfde klinkt.



Figuur 2.11

Neem als voorbeeld een muziekfragment. Als bij het afspelen de samplefrequentie wordt verhoogd, dan wordt de muziek sneller afgespeeld en klinkt alles het hoger. Dat is géén upsampling.

Bij upsampling wordt de samplefrequentie verhoogd én worden samples toegevoegd. We nemen als voorbeeld weer een signaal met een frequentie van 400 Hz en een samplefrequentie van 8000 Hz. Als we upsamplen met een factor 2, dan moet het aantal samples twee keer zo groot worden. Er worden samples met waarde 0 toegevoegd tussen de bestaande samples. De samplefrequentie wordt verdubbeld.

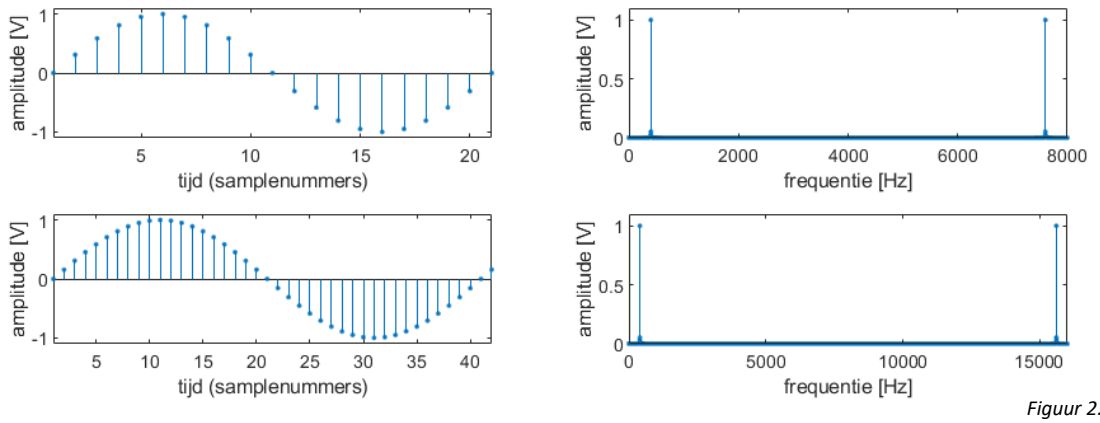


Figuur 2.12

In het originele signaal zit een frequentiecomponent van 400 Hz. Deze is ook gespiegeld aanwezig rondom veelvouden van de samplefrequentie, onder andere bij 7600 en 8400 Hz. Als het signaal analoog gemaakt zou worden en afgespeeld worden, dan zou er behalve de toon van 400 Hz ook een toon van 7600 Hz te horen zijn. Het geüpsamplede signaal klinkt dus anders dan het originele signaal en dat is eigenlijk niet de bedoeling.

Het zou beter zijn om de toegevoegde samples niet de waarde 0 te geven, maar te interpoleren tussen bestaande samples. Dit gebeurt door het geüpsamplede signaal te filteren met een laagdoorlaatfilter met een kantelfrequentie van 4000 Hz. Er worden dus eerst samples met waarde 0 toegevoegd en daarna wordt het signaal gefilterd. Een voorbeeld van het resultaat is te zien in figuur 2.13.

De kantelfrequentie van het *interpolatiefilter* is in het ideale geval de helft van de samplefrequentie van het originele signaal.



Figuur 2.13

Matlab heeft functies voor upsamplen en interpolatie (Signal Processing Toolbox).

```
x=[1 2 3 4 5 4 3 2 1];
xu=upsample(x,2)      % geeft een vector [1 0 2 0 3 0 4 0 3 0 2 0 1 0]
```

De functie `interp` heeft een signaal met minimaal 9 samples nodig.

```
xi=interp(x,2)        % geeft een vector [1.0000  1.4996  2.0000, etc
```

Resampling

Met upsampling en downsampling kan de samplefrequentie alleen met integer factor worden aangepast. Om de samplefrequentie met een niet-integer factor aan te passen, wordt een combinatie van up- en downsampling toegepast. Dat wordt *resampling* genoemd.

Om bijvoorbeeld de samplefrequentie met een factor 1.5 te verhogen, wordt eerst geüpsampled met een factor 3, en daarna gedownsampled met een factor 2.

Bij een factor 1.5 is het gemakkelijk om te bepalen welke factoren gebruikt kunnen worden bij het up- en downsamplen: 3 / 2 = 1.5. Bij andere factoren ligt dat niet zo voor de hand.

Bij audioapparatuur voor de consumentenmarkt wordt vaak een samplefrequentie van 44.1 kHz gebruikt. Bij studioapparatuur worden samplefrequenties van bijvoorbeeld 48 of 96 kHz gebruikt. Stel dat de samplefrequentie van 44.1 kHz aangepast moet worden naar 48 kHz. De verhouding is 48 / 44.1 ≈ 1.09. Welke factoren horen daarbij?

Er kan natuurlijk geüpsampled worden met een factor 48000, maar dat zou betekenen dat er 48000 keer zoveel geheugen nodig zou zijn voor het geüpsamplede signaal. Het zou ook veel rekenwerk kosten om dat signaal te filteren, en vervolgens te downsamplen met een factor 44100.

Er bestaan gelukkig methodes om een reëel getal zo goed mogelijk te benaderen door een breuk. Dat worden wel kettingbreuken genoemd, of in het Engels *rational fraction approximation*.

Matlab heeft functies voor resamplen en rational fraction approximation (Signal Processing Toolbox).

```
xr=resample(x,3,2)     % resamplet een signaal met factor 3/2
[P,Q]=rat(48/44.1)     % geeft P=160 (voor het upsamplen)
                       % en Q=147 (voor het downsamplen)
```

Undersampling

In sommige situaties wordt bewust met een te lage samplefrequentie gesampled. Neem bijvoorbeeld een signaal met frequentiecomponenten tussen 101 en 103 kHz. Door het signaal te samplen worden deze frequentiecomponenten gespiegeld rondom veelvouden van de samplefrequentie. Er ontstaan daardoor dus ook componenten tussen 1 en 3 kHz.

Stel dat als samplefrequentie 10 kHz wordt gekozen. Als het signaal na het samplen door een laagdoorlaatfilter met een kantelfrequentie van 5 kHz wordt geleid, dan blijft een signaal over waar de frequenties tussen 101 en 103 kHz niet meer in zitten, maar alleen de frequenties tussen 1 en 3 kHz. Het lijkt alsof het signaal naar een lagere frequentieband is verplaatst.

Dit soort technieken wordt onder andere toegepast in communicatie- en meetapparatuur. De signalen in de lagere frequentieband zijn gemakkelijker te bewerken en te analyseren dan de signalen in de hogere frequentieband.

Oversampling

In andere situaties wordt juist bewust een te hoge samplefrequentie gebruikt. Dit wordt vooral gedaan om de signaal/ruis-verhouding van het discrete of het gereconstrueerde signaal te verbeteren.

Gegeven een signaal met een bandbreedte van 4 kHz, dat gesampled wordt met een samplefrequentie van 32 kHz. Dat is vier keer hoger dan noodzakelijk volgens Shannon-Nyquist. Er zijn dus vier keer zoveel samples als noodzakelijk. Stel dat er gekwantiseerd wordt met 8 bits. Er zijn dan 2^8 ofwel 256 niveaus, bijvoorbeeld met de waarden 0 t/m 255.

Door nu telkens de gemiddelde waarde te berekenen van vier samples, wordt de resolutie van de kwantisering verhoogd. Laten we zeggen dat de eerste vier samples de waarden 2, 3, 4, 5 hebben. Het gemiddelde daarvan is 3,5. Stel dat de eerste vier samples de waarden 0, 0, 0, 1 hebben. Het gemiddelde is dan 0,25. Er kan nu dus in stapjes van 0,25 gekwantiseerd worden. Er zijn dus eigenlijk 1024 in plaats van 256 niveaus. Je kunt ook zeggen dat er nu met $\log_{10} 1024 = 10$ bits/sample gekwantiseerd wordt.

Bij een groter aantal spanningsniveaus hoeft er tijdens het kwantiseren minder afgerond te worden, waardoor het vermogen van kwantiseringsruis kleiner wordt.

Een andere vorm van oversampling wordt toegepast bij de reconstructie van het analoge signaal. Er wordt dan eerst upsampling met interpolatie toegepast. De totale hoeveelheid kwantisatieruis verandert daardoor niet.

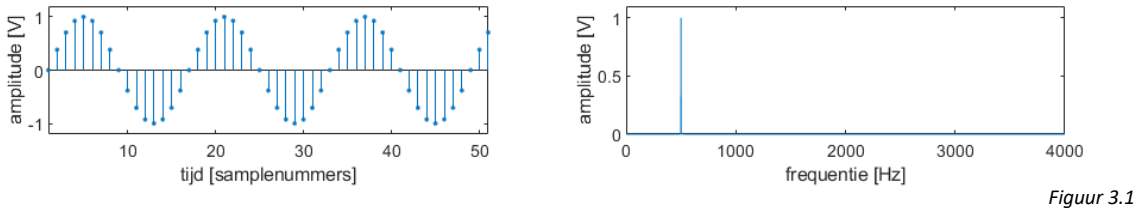
Echter, deze ruis is verspreid over het frequentiebereik van 0 Hz tot de (verhoogde) samplefrequentie van bijvoorbeeld 32 kHz. Het analoge reconstructiefilter moet alles doorlaten tot de helft van de originele samplefrequentie. De kantelfrequentie van het filter is dus 4 kHz. Dat betekent dat de ruis in het gebied van 4 t/m 16 kHz uit het signaal wordt gefilterd. Er blijft maar een kwart van de kwantisatieruis over.

Signalen en systemen

Signalen

In de wereld van de digitale signaalbewerking zien we een signaal als een reeks samplewaarden. Het signaal kan informatie bevatten, bijvoorbeeld over een gemeten grootheid zoals luchtdruk (geluid) of temperatuur. Het kan ook zijn dat het signaal een functie voorstelt, zoals bijvoorbeeld een blokgolf of een sinus.

Sinussen en cosinussen kom je vaak tegen in de signaalbewerking. Vanuit het frequentiespectrum gezien zijn het de meest eenvoudige signaalvormen die bestaan. Elk signaal bestaat uit de som van één of meerdere sinusoïden (sinusvormige signalen), eventueel met verschillende amplitudes, frequenties en fases. Een sinusvormig signaal heeft precies één frequentie en andersom geldt ook dat een signaal met één frequentie alleen een sinusvormig signaal kan zijn.



Figuur 3.1

Als een sinusoïde hoorbaar wordt gemaakt, dan hoor je een toon met precies één frequentie. De amplitude van het signaal bepaalt hoe luid de toon klinkt. De frequentie bepaalt hoe hoog de toon klinkt. Bij één enkele frequentie heeft het weinig zin om over fase te spreken. Deze is eigenlijk niet hoorbaar. Het menselijk oor gebruikt faseverschillen wel om de richting te bepalen vanwaar een geluid komt. Daarvoor wordt gebruik gemaakt van faseverschillen tussen het geluid dat het ene oor waarneemt, ten opzichte van het geluid dat het andere oor waarneemt.

In theorie heeft een sinus of cosinus geen begin of eind. Het signaal is oneindig lang geleden begonnen, en gaat nog oneindig lang door. In de praktijk hebben we te maken met signalen die wel een begin en eind hebben. Als we het over een sinus hebben, dan bedoelen we vaak een sinus die vermenigvuldigd is met een blok. Hierdoor is het signaal aan het begin en aan het eind discontinu. Het is geen echte sinusoïde, dus het signaal bestaat uit meer dan één frequentiecomponent.

In Matlab kan een sinusvormig signaal op verschillende manieren gemaakt worden.

```
fs=8000;                % samplefrequentie van 8000 Hz
t=0:1/fs:1;             % tijdbasis van 1 seconde
x=sin(2*pi*100*t);      % sinusvormig signaal van 100 Hz
```

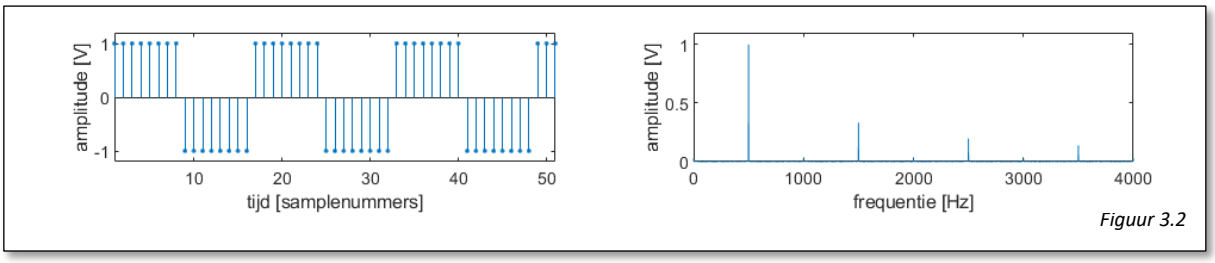
Om een tijdbasis te maken kan ook de functie `linspace` gebruikt worden.

```
t=linspace(0,1,8001);  % tijdbasis van 1 seconde, met 8001 samples
```

Er kan ook voor gekozen worden om, in plaats van een tijdbasis, met samplenummers te werken.

```
fs=8000;                % samplefrequentie van 8000 Hz
n=0:8000;               % samplenummers van 0..8000
x=sin(2*pi*100*n/fs);   % sinusvormig signaal van 100 Hz
```

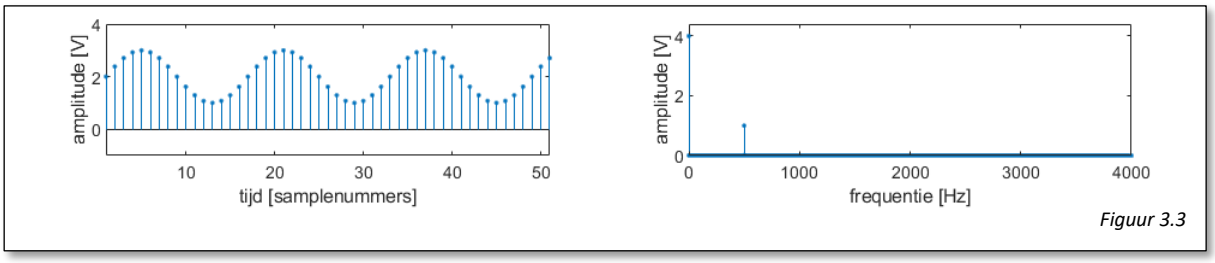
Als een signaal niet sinusvormig is, dan bestaat het uit meerdere sinusoïden. Neem als voorbeeld een blokvormig signaal. Een blokgolf met een frequentie van bijvoorbeeld f = 500 Hz is opgebouwd uit een grondtoon van 500 Hz en een aantal harmonischen. Een *harmonische* (ook wel *boventoon* genoemd) is een component waarvan de frequentie een veelvoud is van de frequentie van de grondtoon. In een blokgolf zitten naast de grondtoon de oneven harmonischen, met frequenties 3·f, 5·f, 7·f, enzovoort.



Figuur 3.2

De gemiddelde waarde van een sinusoïde is nul. Als een signaal een gemiddelde waarde heeft die niet gelijk is aan nul, dan heeft het signaal een *offset*. We spreken ook wel van de *gelijkspanningscomponent* (DC), omdat de offset gezien kan worden als een cosinus met een frequentie van 0 Hz. Als er in het frequentiespectrum een component te zien is bij 0 Hz, dan is dat dus de gemiddelde waarde van het signaal.

In het plaatje hieronder is een sinus met een offset van 2 V te zien. In het frequentiespectrum zit daarom een piek bij 0 Hz. Deze piek lijkt met een amplitude van 4 V veel te groot te zijn, maar dat is vooral een kwestie van de schaalverdeling op de y-as. Het frequentiespectrum wordt berekend van 0 Hz tót de samplefrequentie. Stel dat de samplefrequentie 8000 Hz is. De sinusoïde komt tweemaal in het spectrum voor: bij 500 Hz en bij 7500 Hz. De gelijkspanningscomponent komt maar één keer in het spectrum voor. Bovendien is de offset van de sinus in dit voorbeeld 2 V, terwijl de amplitude van de sinus gelijk is aan 1 V.

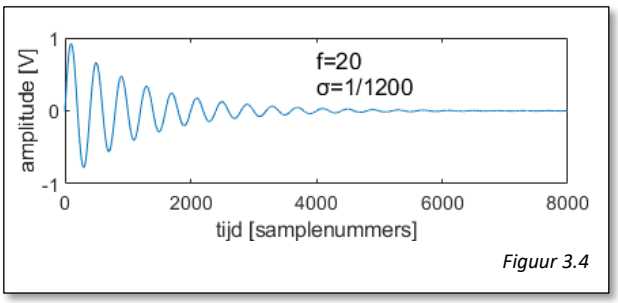


Figuur 3.3

In de praktijk komen veel signalen voor die in de tijd gedempt of juist versterkt worden. De amplitude van een gedempt signaal wordt exponentieel kleiner. De formule van een exponentieel gedempte sinus ziet er bijvoorbeeld als volgt uit.

$$x[n] = e^{-\sigma \cdot n} \cdot \sin(2\pi \cdot f/f_s \cdot n)$$

De constante σ bepaalt de mate van demping.



Figuur 3.4

Om in Matlab twee signalen met elkaar te vermenigvuldigen kun je de operator `.*` gebruiken (dus een punt met daarachter een sterretje). De signalen, die in vectoren van dezelfde afmetingen moeten staan, worden dan sample voor sample met elkaar vermenigvuldigd. Om het signaal in het figuur met de gedempte sinus te maken, is het volgende Matlab-commando gebruikt.

```
x=exp(-n/1200).*sin(2*pi*20*n/fs);
```

De operator `*` wordt in Matlab ook gebruikt voor vermenigvuldiging. Met deze operator kan elk element in een matrix met een scalar vermenigvuldigd worden. Bij vectoren of matrices wordt met deze operator een matrixvermenigvuldiging gedaan.

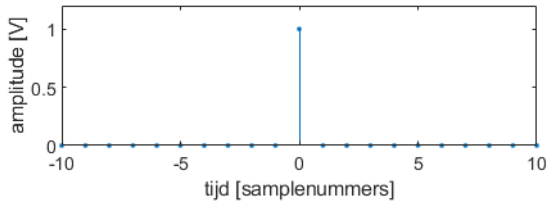
```
5*[1 2 3]              % geeft 5 10 15
[1 2 3]*[1; 2; 3]      % geeft 14 (ofwel 1·1+2·2+3·3)
```

Delta- en stapfunctie

In de wereld van de digitale signaalbewerking is de *deltafunctie δ[n]* een digitaal signaal dat de waarde 1 heeft op tijdstip n = 0 en op alle andere tijdstippen de waarde 0.

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

De deltafunctie wordt ook wel *impuls* genoemd, of *eenheidsfunctie*. De tegenhanger van deze functie in het continue domein is de *dirac-functie*.

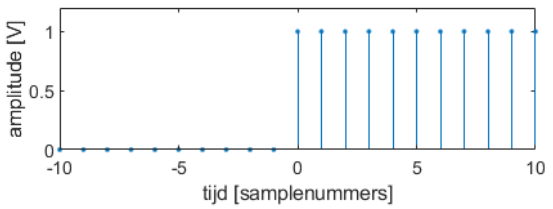


Figuur 3.5

De *stapfunctie u[n]* is een digitaal signaal dat op tijdstippen n ≥ 0 de waarde 1 heeft en op alle andere tijdstippen de waarde 0.

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

De tegenhanger van deze functie in het continue domein is de *heaviside-functie*, die meestal gewoon stapfunctie genoemd wordt.



Figuur 3.6

In de regeltechniek wordt veelal de stapfunctie gebruikt bij het beschrijven van systemen, terwijl in de signaalbewerking meer gebruik wordt gemaakt van de deltafunctie.

Om in Matlab een deltafunctie te maken, kan eenvoudigweg een vector gemaakt worden met een 1, gevolgd door een aantal nullen.

```
d=[1 0 0 0 0];
```

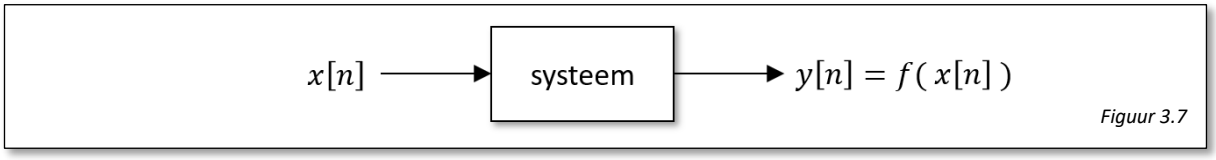
Een langere deltafunctie kan als volgt gemaakt worden.

```
d=[1 zeros(1,1000)];
```

Om signalen in het tijddomein te beschrijven wordt altijd een kleine letter gebruikt (`x`, `y`, `d`, etc).

LTI-Systemen

Een systeem kunnen we zien als een functie. Het ingangssignaal wordt meestal *x[n]* genoemd en het uitgangssignaal wordt dan *y[n]* genoemd. In de wereld van de digitale signaalbewerking wordt een systeem ook vaak *filter* genoemd. Systemen kunnen bepaalde eigenschappen hebben, waarvan er hier een aantal besproken worden.

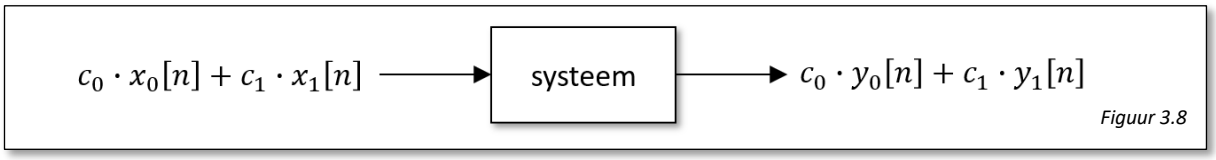


Figuur 3.7

Lineariteit

Een belangrijke eigenschap van een systeem is *lineariteit*. Als het ingangssignaal van een lineair systeem bijvoorbeeld *c* keer zo groot wordt gemaakt, dan wordt ook het uitgangssignaal *c* keer zo groot.

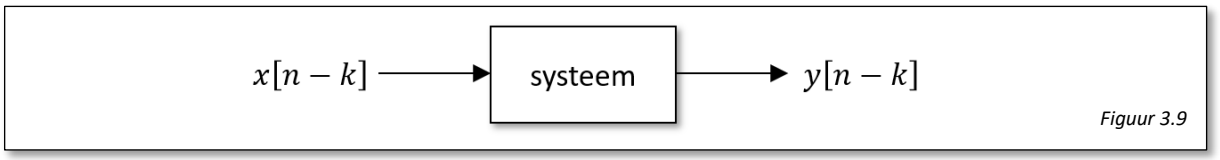
Stel dat er twee signalen $x_1[n]$ en $x_2[n]$ zijn, waarbij het uitgangssignaal van het systeem respectievelijk $y_1[n]$ en $y_2[n]$ is. Als de som van $x_1[n]$ en $x_2[n]$ aan een lineair systeem wordt aangeboden, dan is het uitgangssignaal gelijk aan de som van $y_1[n]$ en $y_2[n]$. Voor een lineair systeem geldt de volgende relatie.



Figuur 3.8

Tijdinvariantie

Een andere eigenschap die een systeem kan hebben is *tijdinvariantie*. Dat wil zeggen dat het niet uitmaakt wanneer een bepaald ingangssignaal aan het systeem wordt aangeboden, het zal altijd hetzelfde uitgangssignaal opleveren. Als een ingangssignaal *k* samples in de tijd wordt vertraagd, dan zal het bijbehorende uitgangssignaal ook *k* samples in de tijd vertraagd zijn.



Figuur 3.9

Systemen die zowel lineair als tijdinvariant zijn, worden *lineaire tijdinvariante systemen*, of *LTI-systemen* genoemd. Als er in dit document over een systeem of een filter wordt gesproken, kun je ervan uitgaan dat daarmee altijd een LTI-systeem wordt bedoeld.

Causaliteit

Een derde eigenschap die een systeem kan hebben, heeft te maken met *causaliteit*. Bij een causaal systeem is er een oorzakelijk verband tussen het ingangssignaal en het uitgangssignaal.

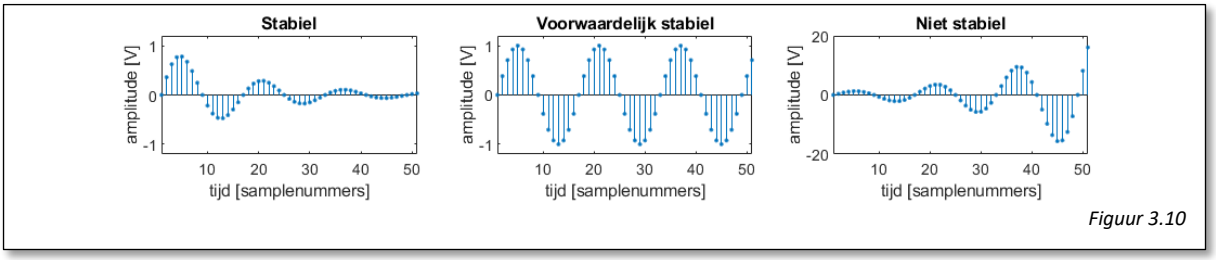
Een voorbeeld van een niet-causaal systeem is een systeem dat een ruissignaal produceert, waarbij er geen enkele relatie is tussen het ingangs- en het uitgangssignaal. Een ander voorbeeld van een niet-causaal systeem is een systeem dat het sample *x[n+1]* nodig heeft om het uitgangssample *y[n]* te berekenen. Omdat er een sample uit de toekomst nodig is, is er geen oorzakelijk verband tussen in- en uitgang.

Het lijkt misschien dat niet-causale systemen niet gebruikt worden of ongewenst zijn. Dat is niet het geval. Een systeem dat samples uit de toekomst nodig heeft, kan niet *real-time* gebruikt worden, maar wel *offline*. Bij een real-time systeem wordt een ingangssignaal aangeboden en wordt binnen een korte, vastgestelde tijd het bijbehorende uitgangssample berekend. Er kunnen dan geen samples uit de toekomst gebruikt worden.

Bij een *offline* systeem kan het zijn dat alle ingangssamples bekend zijn als de signaalbewerking plaatsvindt. Stel je meet gedurende een jaar lang, elke dag de buitentemperatuur. Aan het eind van het jaar kan je dan voor bijna alle dagen het gemiddelde berekenen, bijvoorbeeld van de temperatuur van een bepaalde dag, de dag ervoor en de dag erna. Aan het eind van het jaar weet je voor bijna alle dagen de temperatuur van 'de dag erna', ofwel een dag in de toekomst.

Stabiliteit

Een belangrijke eigenschap van een systeem is de *stabiliteit*. Zolang het ingangssignaal van een stabiel systeem begrensd is (en dus geen oneindig grote amplitude heeft), dan zal het uitgangssignaal ook begrensd zijn. Er wordt wel gesproken van *BIBO-stability*, waarbij *BIBO* staat voor *Bounded Input Bounded Output*. Een systeem dat op de grens tussen stabiel en niet-stabiel zit, wordt wel *voorwaardelijk stabiel* genoemd. In onderstaand figuur zijn een paar voorbeelden te zien van uitgangssignalen van verschillende systemen.



Figuur 3.10

Terugkoppeling

De laatste eigenschap die we bekijken is terugkoppeling. Bij een systeem met terugkoppeling worden oude samples van het uitgangssignaal gebruikt om een nieuw uitgangssample te berekenen.

We zullen nog zien dat systemen met terugkoppeling bepaalde voor- en nadelen hebben. Eén van de voordelen is dat er minder coëfficiënten nodig zijn om een signaal te filteren, dan wanneer er geen terugkoppeling is. Daardoor kan een filter met terugkoppeling sneller zijn werk doen. Een nadeel van een systeem met terugkoppeling is dat het instabiel kan zijn, terwijl een systeem zonder terugkoppeling altijd stabiel is.

Systeembeschrijvingen

Er zijn verschillende manieren waarop een systeem beschreven kan worden. We bekijken een paar van deze manieren om een eenvoudig *moving average* filter te beschrijven. Een moving average filter berekent het gemiddelde van de laatste *k* samples van het ingangssignaal.

Differentievergelijking

Stel dat we een MA-filter bekijken met $k = 3$. Het is een LTI-systeem zonder terugkoppeling, dat met de volgende *differentievergelijking* beschreven kan worden:

$$y[n] = \frac{1}{3} \cdot x[n] + \frac{1}{3} \cdot x[n-1] + \frac{1}{3} \cdot x[n-2]$$

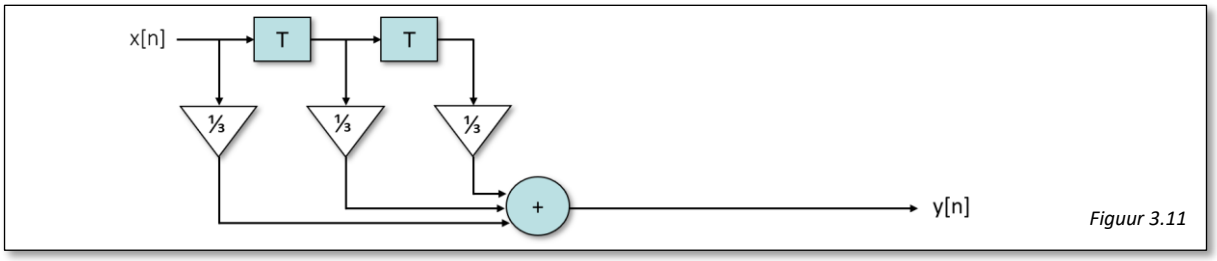
De vergelijking beschrijft de relatie tussen het ingangssignaal en het uitgangssignaal, waarbij *x[n]* het huidige ingangssample is, *x[n-1]* het vorige sample en *x[n-2]* het sample daarvoor. De waarden $\frac{1}{3}$, $\frac{1}{3}$ en $\frac{1}{3}$ zijn de *coëfficiënten* van het filter.

De coëfficiënten worden vaak met de letter *b* aangeduid en het aantal coëfficiënten met *N*. De orde van het filter is dan *N-1*. Een filter met drie coëfficiënten is dus een tweede-orde filter. De algemene vorm van de differentievergelijking van een LTI-filter zonder terugkoppeling ziet er als volgt uit:

$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k]$$

Signal flow diagram

Een andere manier om een systeem te beschrijven is met een *signal flow diagram*, ook wel blokschema genoemd. In onderstaande figuur is het diagram van het MA-filter (*k* = 3) weergegeven.



Figuur 3.11

De blokken met een *T* zijn delays. In plaats een *T* wordt ook wel een *D* of z^{-1} gebruikt. De rij met delayblokken kun je zien als een soort schuifregister. Telkens als er een nieuw sample binnenkomt, worden de oude ingangssamples één plaats verder geschoven. Er wordt wel gesproken van de *delay line* of *tap delay line*. Als de ingang van een delayblok de waarde *x[n]* heeft, dan heeft de uitgang de waarde *x[n-1]*.

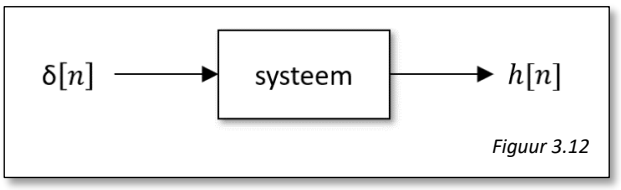
In het signal flow diagram hierboven is de delay line horizontaal getekend, maar deze wordt ook vaak verticaal getekend.

Impulsresponsie

Weer een andere manier om een systeem te beschrijven is door middel van de *impulsresponsie*. Dit is het signaal dat het systeem produceert als er een deltafunctie op de ingang wordt aangeboden. De impuls-responsie wordt meestal aangeduid met de letter *h*.

In het geval van het MA-filter (*k* = 3) ziet de impulsresponsie er als volgt uit:

$$h = \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$$



Figuur 3.12

Omdat de impulsresponsie een eindig aantal waarden bevat, spreken we over een *Finite Impuls Response* filter, ofwel een *FIR-filter*. Bij een LTI-systeem zonder terugkoppeling is de impulsresponsie gelijk aan de waarden van de coëfficiënten.

De impulsresponsie bevat alle informatie over het systeem. Als je dus de impulsresponsie weet, dan weet je alles over het gedrag van het systeem. De impulsresponsie kan beschreven worden met een vergelijking en deze is heel gemakkelijk om te zetten naar de differentievergelijking van het systeem.

$$h[n] = \frac{1}{3}\cdot\delta[n] + \frac{1}{3}\cdot\delta[n-1] + \frac{1}{3}\cdot\delta[n-2] \quad \rightarrow \quad y[n] = \frac{1}{3}\cdot x[n] + \frac{1}{3}\cdot x[n-1] + \frac{1}{3}\cdot x[n-2]$$

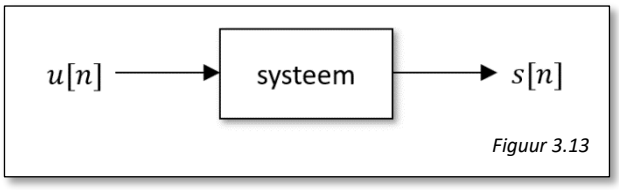
Dat de impulsresponsie gemakkelijk omgezet kan worden naar de differentievergelijking van een filter, geldt alleen voor FIR-systemen. Er bestaan ook systemen met een oneindig lange impulsresponsie: de zogenaamde *Infinite Impulse Response* filters, of *IIR-filters*. Bij deze filters kan de impulsresponsie ook gebruikt worden om het gedrag van het filter te beschrijven, maar is het veel lastiger om aan de hand van de impulsresponsie bijvoorbeeld de coëfficiënten te achterhalen.

Stapresponsie

In de digitale signaalbewerking wordt meestal met de impulsresponsie gewerkt, maar een systeem kan ook beschreven worden door middel van de *stapresponsie*. Dit is het signaal dat het systeem produceert als een stapfunctie op de ingang wordt aangeboden. De stapresponsie wordt meestal aangeduid met de letter *s*.

In het geval van het MA-filter (*k* = 3) bestaat de stapresponsie uit de volgende waarden:

$$s = \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1, 1, 1, 1 \ldots$$



Figuur 3.13

Net zoals bij de impulsresponsie, bevat de stapresponsie alle informatie van een LTI-systeem. Als je dus de stapresponsie weet, dan weet je alles over het gedrag van het systeem. Als de stapresponsie als een functie van de stapfunctie *u* beschreven kan worden, dan kan weer eenvoudig de differentievergelijking bepaald worden.

$$s[n] = \frac{1}{3}\cdot u[n] + \frac{1}{3}\cdot u[n-1] + \frac{1}{3}\cdot u[n-2] \quad \rightarrow \quad y[n] = \frac{1}{3}\cdot x[n] + \frac{1}{3}\cdot x[n-1] + \frac{1}{3}\cdot x[n-2]$$

Een systeem kan beschreven worden door middel van de differentievergelijking, het signal flow diagram, de impulsresponsie en de stapresponsie. Dit zijn allemaal beschrijvingen in het tijddomein. Er bestaan ook beschrijvingen in het frequentiedomein, waar we later nog naar zullen kijken.

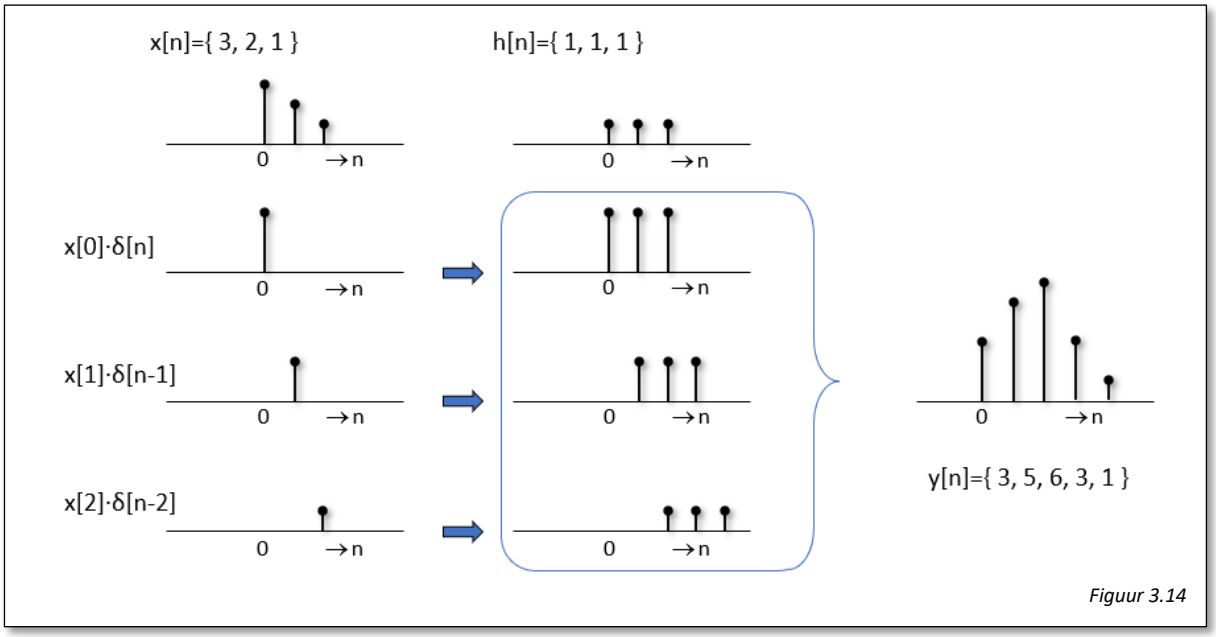
Convolutie

De bewerking die een FIR-filter uitvoert, wordt *convolutie* genoemd. De samples in de delay line worden vermenigvuldigd met de coëfficiënten en de resultaten daarvan worden bij elkaar opgeteld om zo het nieuwe uitgangssample te vormen.

Stel we hebben een systeem met impulsresponsie *h[n] = { 1, 1, 1 }*. Het is een eindige impulsresponsie, dus het betreft een FIR-filter. De coëfficiënten zijn hetzelfde als de waarden van de impulsresponsie. Stel dat we een ingangssignaal *x[n] = { 3, 2, 1 }* aanbieden. Dit signaal kunnen we beschouwen als drie deltafuncties, met amplitudes 3, 2 en 1, die respectievelijk 0, 1 en 2 sampleperioden in de tijd verschoven zijn.

Omdat het een lineair systeem is, kunnen we het uitgangssignaal bepalen door de responsies van het systeem op elk van de samples van het ingangssignaal bij elkaar op te tellen.

Het eerste sample geeft een responsie met waarden *{ 3, 3, 3 }*, ofwel *x[0]·h[n]*. Het tweede sample is één periode in de tijd verschoven. Omdat het systeem tijdinvariant is, kunnen we zeggen dat de responsie ook één periode verschoven is. Het tweede sample geeft dus een responsie met waarden *{ 0, 2, 2, 2 }*, ofwel *x[1]·h[n-1]*. Het laatste sample geeft een responsie met waarden *{ 0, 0, 1, 1, 1 }*, ofwel *x[2]·h[n-2]*.



Figuur 3.14

De responsies op de verschillende samples kunnen we bij elkaar optellen, hetgeen een uitgangssignaal met de waarden *{ 3, 5, 6, 3, 1 }* oplevert. In het algemeen kunnen we zeggen dat het uitgangssignaal er als volgt uitziet.

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Deze vergelijking wordt de *convolutiesom* genoemd. De indexvariabele *k* loopt van $-\infty$ tot $+\infty$. In de praktijk is het voldoende om met deze variabele het ingangssignaal te doorlopen. In bovenstaand voorbeeld kan *k* van 0…2 lopen.

Uiteraard heeft Matlab een functie om de convolutie uit te rekenen.

```
conv([3 2 1],[1 1 1])        % geeft als resultaat [3 5 6 3 1]
```

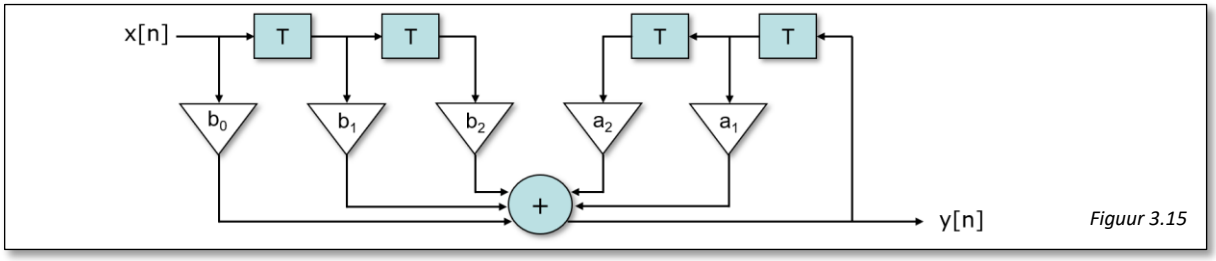
IIR-filters

Zoals eerder vermeld, bestaan er naast FIR-filters ook filters met een oneindig lange impulsresponsie: de *IIR-filters*. De algemene differentievergelijking van een IIR-filter ziet er als volgt uit.

$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k] + \sum_{k=1}^{M} a_k \cdot y[n-k]$$

In deze vergelijking is te zien dat er oude samples *y[n - k]* van het uitgangssignaal gebruikt worden om een nieuw uitgangssample *y[n]* te berekenen. Er is bij IIR-filters dus altijd sprake van terugkoppeling, waardoor het voor kan komen dat een IIR-filter instabiel is. Bij filters met terugkoppeling wordt ook wel van *recursieve filters* gesproken. Merk op dat een filter met terugkoppeling niet altijd een oneindig lange impulsresponsie hoeft te hebben.

Een voorbeeld van een signal flow diagram van een IIR-filter ziet er als volgt uit.



Figuur 3.15

Merk op dat zowel in de differentievergelijking als in het signal flow diagram de *b*-coëfficiënten genummerd zijn vanaf 0, en de *a*-coëfficiënten vanaf 1. De coëfficiënt a_0 kan gebruikt worden als algemene verzwakkingsfactor (die dan voor de term *y[n]* gezet wordt), maar in de meeste geval heeft a_0 de waarde 1, en wordt dan weggelaten uit de differentievergelijking en het signal flow diagram.

Zoals de naam al aangeeft, heeft een IIR-filter een oneindig lange impulsresponsie. Anders dan bij een FIR-filter, kunnen we daarom niet zeggen dat de coëfficiënten dezelfde waarden hebben als de samples van de impulsresponsie. Het filter zou dan oneindig veel coëfficiënten moeten hebben. Het bepalen van de waarden voor de coëfficiënten van een IIR-filter is over het algemeen wat lastiger dan het bepalen van de coëfficiënten van een FIR-filter.

Je zou kunnen zeggen dat een FIR-filter een subset van een IIR-filter is. Als de coëfficiënten $a_1 \ldots a_M$ allemaal de waarde nul hebben, dan vervalt daarmee het recursieve deel van het filter en blijft er een FIR-filter over.

De meeste boeken en websites over DSP gebruiken de letter *b* om de coëfficiënten van het niet-recursieve deel van het filter te benoemen, en de letter *a* voor de coëfficiënten van het recursieve deel. Helaas zijn er ook boeken en website waarbij dit is omgedraaid. Zorg er dus voor dat als je een tekst over filters leest, dat je weet welke conventie gebruikt wordt.

Er zijn ook boeken en websites die een opbouw van IIR-filters gebruiken waarbij de berekeningen van het recursieve deel niet worden opgeteld, maar worden afgetrokken van de berekeningen van het niet-recursieve deel. In de differentievergelijking staat dan een minteken voor het recursieve deel. De functie *filter* van Matlab werkt op die manier. Er zijn goede redenen om dit doen, net zoals er goede redenen zijn om alles gewoon bij elkaar op te tellen zoals we hier doen. Het is een kwestie van afspraken, en beide manieren zijn goed. Uiteraard moet je bij berekeningen weten welke manier gebruikt wordt.

De Matlab-functie om IIR-filters door te rekenen heet `filter`. Het is het beste om de coëfficiënten in afzonderlijke vectoren aan te bieden. Met dit voorbeeld wordt het signaal in vector `x` gefilterd door een hoogdoorlaatfilter.

```
b=[1 -2 1];             % er zijn drie b- en drie a-coëfficiënten,
a=[1 0 0.17];           % het is dus een tweede-orde filter
y=filter(b,a,x);        % het ingangssignaal x wordt gefilterd
```

De functie kan ook gebruikt worden om een FIR-filter door te rekenen (en dus een convolutie te doen). In dat geval moeten de a-coëfficiënten nul zijn, behalve a_0, die de waarde één moet hebben. Stel dat het ingangssignaal in vector `x` zit, dan kan het uitgangssignaal op de volgende manier berekend worden.

```
b=[1 1 1]/3;            % de b-coëfficiënten zijn allemaal 1/3
a=[1];                  % a0 is 1 en overige a-coëfficiënten zijn 0
y=filter(b,a,x);        % het uitgangssignaal wordt in y gezet
```

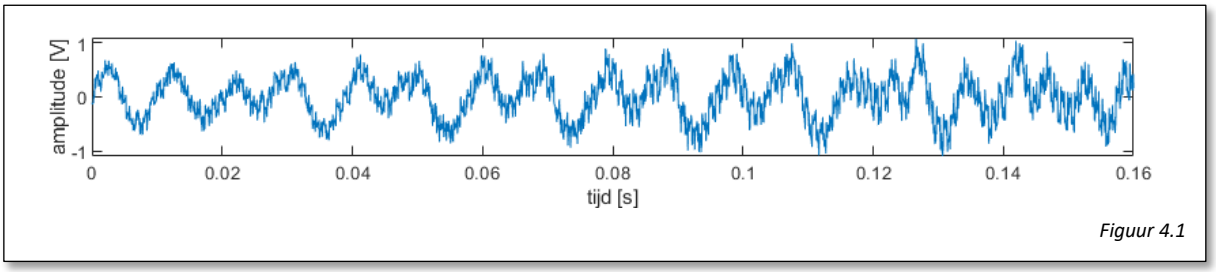
Fourier transformatie

Inleiding

Een signaal kunnen we zien als een reeks samplewaarden, die bijvoorbeeld het verloop van de amplitude van het signaal in de tijd weergeven. Stel dat we een filter willen ontwerpen om bepaalde frequenties uit het signaal te filteren. Het is dan niet meer genoeg om alleen naar het signaal te kijken in het tijddomein, maar we hebben een hulpmiddel nodig om ook in het frequentiedomein met het signaal te werken.

Elke signaal is opgebouwd uit reeks sinusoïden met verschillende amplitudes, frequenties en fases. Als we een signaal in het frequentiedomein willen bekijken, dan moeten we er dus achter komen welke frequentiecomponenten er in het signaal zitten en welke amplitudes die componenten hebben.

Stel dat we een deel van een muzieksignaal in de tijd hebben, zoals in onderstaand figuur.



Figuur 4.1

Hoe zou je erachter kunnen komen of hier een frequentiecomponent van bijvoorbeeld 1000 Hz in zit? En welke amplitude zou die component hebben?

Eén manier om hier achter te komen is door het signaal te vermenigvuldigen met een cosinus van 1 kHz. Als sinusoïden met elkaar vermenigvuldigd worden, ontstaan er som- en verschilfrequenties.

$$\cos\alpha \cdot \cos\beta = \frac{1}{2} \cdot \cos(\alpha - \beta) + \frac{1}{2} \cdot \cos(\alpha + \beta)$$

Bij alle frequentiecomponenten van het muzieksignaal levert de vermenigvuldiging som- en verschil-frequenties op. Stel dat er een frequentiecomponent van bijvoorbeeld 200 Hz in het signaal zit, dan levert de vermenigvuldiging met het signaal van 1000 Hz dus cosinussen van 800 en 1200 Hz op. Deze cosinussen hebben een gemiddelde waarde van 0 en de som van de cosinussen heeft dan ook een gemiddelde waarde van 0.

Stel dat er inderdaad een frequentiecomponent van 1 kHz in het muzieksignaal zit. De vermenigvuldiging levert dan een somfrequentie van 2000 Hz op, maar ook een verschilfrequentie van 0 Hz. Een cosinus van 0 Hz heeft altijd de waarde 1 en is dus een gelijkspanning. De gemiddelde waarde van het resulterende signaal is groter dan 0, en daaruit kan geconcludeerd worden dat de betreffende frequentie van 1000 Hz inderdaad in het signaal zit.

De gemiddelde waarde is een maat voor de amplitude van de betreffende frequentiecomponent. Als de gemiddelde waarde bij een bepaalde frequentie nul is, dan zit die frequentie niet in het signaal. De gemiddelde waarde van een signaal kan gevonden worden door het signaal te integreren over een bepaalde tijd, en vervolgens door die tijd te delen.

Als we van een signaal kunnen bepalen of er één specifieke frequentiecomponent in zit (bijvoorbeeld van 1000 Hz), dan kunnen we dat ook voor alle andere frequenties doen.

We kunnen van een signaal *x(t)* dus bepalen welke cosinussen er in zitten, door het signaal te vermenigvuldigen met een cosinus en vervolgens de gemiddelde waarde te bepalen.

$$X(\omega) = \int_{-\infty}^{+\infty} x(t) \cdot \cos \omega t \cdot dt$$

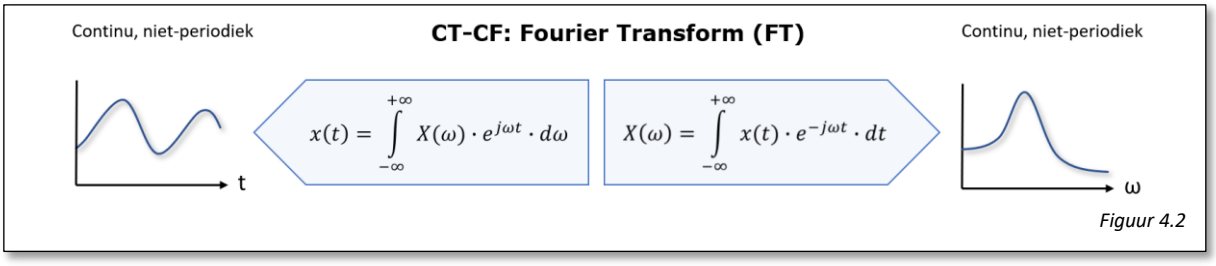
In deze formule wordt alleen met een cosinus vermenigvuldigd, en daardoor kunnen alleen cosinussen in het signaal gevonden worden. Door met een sinus te vermenigvuldigen kunnen we erachter komen welke sinussen in het signaal zitten. Door met een combinatie van een reële cosinus en een imaginaire sinus te vermenigvuldigen, kunnen we van elke frequentiecomponent de amplitude bepalen. Overigens kan uit het resultaat behalve de amplitude ook de fase van de betreffende frequentiecomponent gevonden worden. De cosinus en de sinus kunnen volgens Euler als een complexe e-macht geschreven worden. De formule ziet er dan als volgt uit.

$$X(\omega) = \int_{-\infty}^{+\infty} x(t) \cdot (\cos \omega t - j \cdot \sin \omega t) \cdot dt = \int_{-\infty}^{+\infty} x(t) \cdot e^{-j\omega t} \cdot dt$$

Dit is de formule van de *Fourier transformatie*. Het signaal in het tijddomein en het resultaat in het frequentiedomein zijn beide continue signalen. In de formule staat een minteken voor het imaginaire gedeelte. Er is een goede reden[1] dat deze keuze gemaakt is, maar het is niet heel belangrijk om te weten wat die reden is.

De Fourier transformatie wordt vaak als volgt genoteerd: $X(\omega) = \mathcal{F}(\, x(t)\,)$. Merk op dat signalen in de tijd met een kleine letter worden aangeduid, zoals *x(t)*. Signalen in het frequentiedomein worden met hoofdletters aangeduid, zoals *X(ω)*.

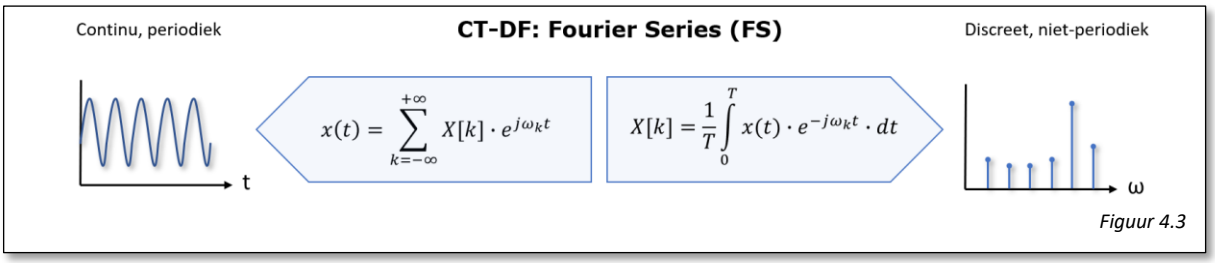
Er bestaat ook een formule voor de *inverse Fourier transformatie*, waarmee van het frequentiedomein naar het tijddomein kan worden getransformeerd. Die formule lijkt heel veel op de formule hierboven, alleen het minteken ontbreekt. De Fourier transformatie zoals die hier is besproken, geldt voor continue signalen (CT-CF, ofwel continu in tijd en continu in frequentie).



Figuur 4.2

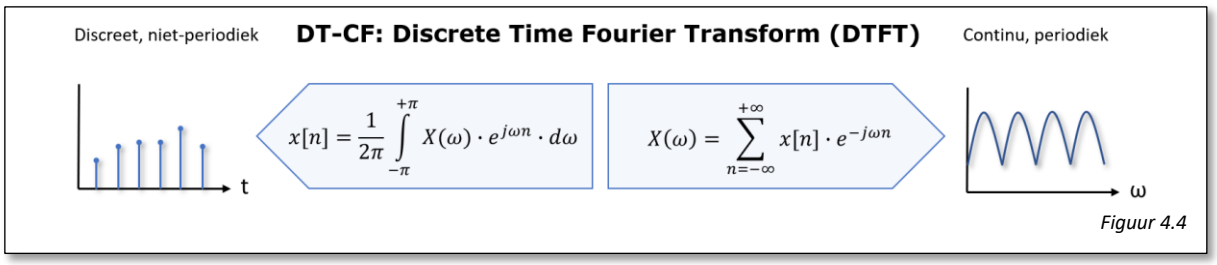
Als het signaal in de tijd periodiek is, zoals in het volgende figuur, dan blijkt dat het frequentiespectrum discreet is. In het spectrum zie je dan een component bij de grondtoon van het periodieke signaal en bij de harmonischen. Omdat het spectrum discreet is, is bij de inverse transformatie geen integraal meer nodig, maar kan volstaan worden met een eenvoudige sommatie.

[1] Het resultaat van de transformatie is een complex getal, waar dus de fase van de betreffende frequentie-component van kan worden afgeleid. Er wordt bij frequentiecomponenten uitgegaan van cosinussen i.p.v. sinussen. Een sinus kan geschreven worden als $\cos(\omega t + \varphi)$, waarbij φ de waarde $-\pi/2$ heeft. Het minteken in de formule zorgt ervoor dat de juiste fase wordt berekend.



Figuur 4.3

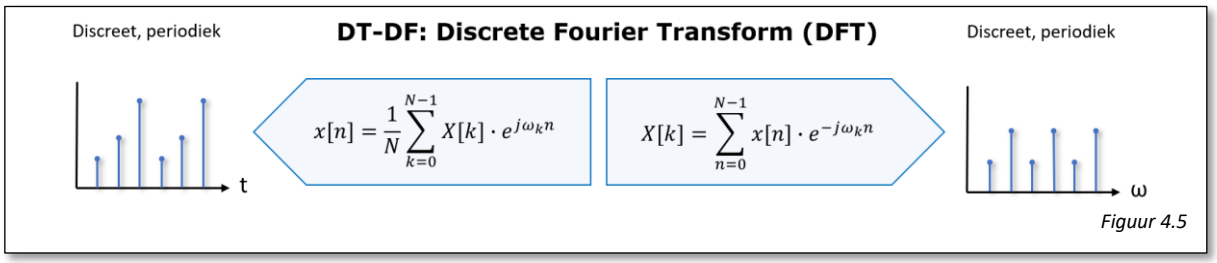
Uiteraard kan ook voor discrete signalen het frequentiespectrum worden bepaald, zoals in onderstaand figuur te zien is.



Figuur 4.4

Merk op dat als een signaal in het ene domein discreet is, dat het dan in het andere domein periodiek is. In DSP-systemen werken we met discrete waarden. Als in een DSP-systeem het frequentiespectrum wordt berekend van een discreet signaal in de tijd, dan is het frequentiespectrum dus periodiek. Andersom geldt ook, dat als het frequentiespectrum discreet is, het signaal in de tijd periodiek moet zijn. Als je een Fourier transformatie uitvoert op een muzieksignaal, dan gaat de transformatie er dus vanuit dat het muzieksignaal periodiek is.

In de DSP-wereld wordt de *Discrete Fourier Transform (DFT)* het meest gebruikt. Zowel het signaal in de tijd- als het signaal in het frequentiedomein zijn dan discreet én periodiek.



Figuur 4.5

Als de DFT wordt uitgevoerd met *N* samples in het tijddomein, dan is het resultaat een vector met *N* elementen, die het frequentiedomein beschrijven van frequentie 0 tót de samplefrequentie.

In de formule van de DFT wordt de indexvariabele *n* gebruikt om door *N* samples van het ingangssignaal te lopen. De variabele *k* wordt gebruikt om de *N* frequentiecomponenten te nummeren. De frequentie die hoort bij component *k* kan gevonden worden middels de volgende vergelijking.

$$\omega_k = 2\pi \cdot \frac{k}{N}$$

In sommige van de bovenstaande formules staat een factor *1/T*, *1/2π* of *1/N*. Dit zijn telkens constante waarden, die gebruikt worden om het gemiddelde van het resultaat van de integraal of de sommatie te berekenen. De feitelijke waarde van deze factoren is niet zo van belang. Ze zijn in de formule opgenomen om ervoor te zorgen dat als je eerst de Fourier transformatie doet, en daarna op het resultaat de inverse Fourier transformatie, er weer hetzelfde signaal uitkomt. Het maakt dan ook niet zoveel uit in welke van de twee formules de constante staat.

Om een Fourier transformatie uit te voeren, zijn veel berekeningen nodig. Stel dat het signaal *x[n]* bestaat uit 8000 samples, dan zijn er voor het berekenen van één component van het frequentiespectrum 8000 vermenigvuldigingen met een complexe e-macht nodig. Om het hele spectrum, met 8000 componenten te bepalen, zijn dan 64 miljoen vermenigvuldigingen nodig.

Gelukkig is er een slimme manier bedacht om de transformatie uit te voeren, waarbij het signaal *x[n]* in kleine stukken wordt verdeeld. Van elke klein stuk wordt het spectrum bepaald en de resultaten worden samengevoegd om het hele spectrum te beschrijven. Deze methode wordt *Fast Fourier Transform (FFT)* genoemd en levert een hele grote besparing van het aantal berekeningen op.

Matlab heeft uiteraard een functie `fft()` om de DFT uit te voeren (DT-DF) met het FFT-algoritme.

```
[x fs]=audioread('chime.wav');    % lees een audiobestand in
X=fft(x);         % transformeer naar het frequentiedomein
                  % let op het gebruik van hoofd- en kleine letters
```

De Fourier transformatie retourneert een vector met complexe getallen, dat het frequentiegebied van 0 Hz tót de samplefrequentie beschrijft. Het resultaat van de transformatie geeft informatie over amplitude en fase van het signaal, en over de overeenkomsten met cosinussen en sinussen.

```
stem(abs(X));     % plot de amplitude van het spectrum
stem(angle(X));   % plot de fase van het spectrum

stem(real(X));    % plot de overeenkomsten met cosinus
stem(imag(X));    % plot de overeenkomsten met sinus
```

Discrete Fourier Transform (DFT)

De DFT is de transformatie van het discrete tijddomein naar het discrete frequentiedomein. Het aantal frequentiecomponenten waarmee het spectrum wordt beschreven (van 0 Hz tót de samplefrequentie) is gelijk aan het aantal samples van het tijdsignaal dat aan de DFT wordt aangeboden. Deze componenten worden ook wel *frequency bins* genoemd.

Gegeven een signaal dat beschreven wordt met vier samples, bijvoorbeeld *{ 0, 1, 2, 3 }*. Dit signaal wordt door de DFT beschouwd als een periodiek signaal, dat er uitziet als een zaagtand. Als de DFT van het signaal wordt berekend, dan is het resultaat een vector met vier complexe getallen: *{ 6, -2+2j, -2, -2-2j }*, waarmee het spectrum tót de samplefrequentie wordt beschreven. De amplitude van het spectrum kan gevonden worden door de absolute waarden van de complexe getallen te nemen: *{ 6, 2.83, 2, 2.83 }*.

Het frequentiespectrum is gespiegeld rondom de halve samplefrequentie. Daarom hebben in dit geval het tweede en vierde element van de amplitudevector dezelfde waarde. Stel dat de samplefrequentie gelijk is aan 8000 Hz. We weten dan dat de amplitude bij 0 Hz gelijk is aan 6, bij 2000 Hz is deze 2.83 en bij de halve samplefrequentie (4000 Hz) is deze gelijk aan 2.

Maar stel dat we het frequentiespectrum willen weten van een niet-periodiek signaal? Bijvoorbeeld het signaal met dezelfde waarden *{ 0, 1, 2, 3 }*, dat voorafgegaan en gevolgd wordt door oneindig veel samples met de waarde 0. Het is een oneindig lang signaal, dat niet periodiek is. Als we van dit signaal de DFT zouden berekenen, dan is het resultaat een beschrijving van het spectrum met oneindig veel frequentie-componenten. We kunnen dan van elke frequentiecomponent de amplitude en fase vinden.

In de praktijk is het niet handig om met een oneindig lange vector te werken. We kunnen aan het tijdsignaal wel een aantal nullen toevoegen, waardoor het signaal ‘minder periodiek’ wordt. Het resultaat van de DFT is dan ‘minder discreet’ of ‘meer continu’, omdat het door meer waarden beschreven wordt. Het toevoegen van nullen aan een vector wordt *zero padding* genoemd.

In Matlab kunnen gemakkelijk nullen aan een vector worden toegevoegd. Merk op dat de vector *x* in dit voorbeeld niet gewijzigd wordt. Er worden alleen nullen meegegeven bij de aanroep van de *fft()*.

```
x=[0 1 2 3];

X=fft(x);                    % geeft het spectrum in 4 waarden
X=fft([x zeros(1,996)]);     % geeft het spectrum in 1000 waarden
```

Omdat de DFT (en dus ook de FFT) het tijdsignaal als periodiek beschouwt, maakt het niet uit of de nullen voor of achter het signaal geplaatst worden.

Eigenschappen de Fourier transformatie

De Fourier transformatie heeft een aantal eigenschappen, waarvan er hier drie genoemd worden. De eigenschappen worden hieronder beschreven voor de DFT, maar gelden ook voor de andere varianten van de transformatie.

De Fourier transformatie heeft een *lineariteitseigenschap*. Het is een lineaire operatie, omdat deze voldoet aan de definitie van lineariteit zoals we die bij LTI-systemen hebben gezien.

$$a \cdot x[n] + b \cdot y[n] \leftrightarrow a \cdot X[k] + b \cdot Y[k]$$

De Fourier transformatie heeft een *tijdverschuivingseigenschap*. Als een signaal in de tijd wordt vertraagd, dan is dat in de Fourier-getransformeerde van het signaal terug te zien als een vermenigvuldiging met een complexe e-macht.

$$x[n-m] \leftrightarrow X[\omega] \cdot e^{-j\omega m}$$

De Fourier transformatie heeft een *convolutie-eigenschap*. Een convolutie in het tijddomein is hetzelfde als een vermenigvuldiging in het frequentiedomein.

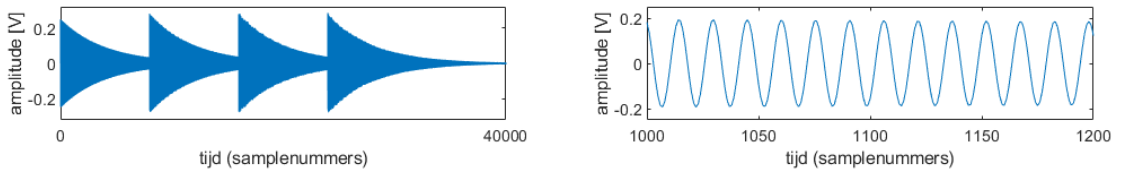
$$DFT\{x_1[n] * x_2[n]\} = X_1[\omega] \cdot X_2[\omega]$$

Deze eigenschappen worden gebruikt bij het ontwerpen van FIR-filters. Het bewijs voor elk van deze eigenschappen is te vinden in bijlage 1 van dit document.

Analyseren van signalen

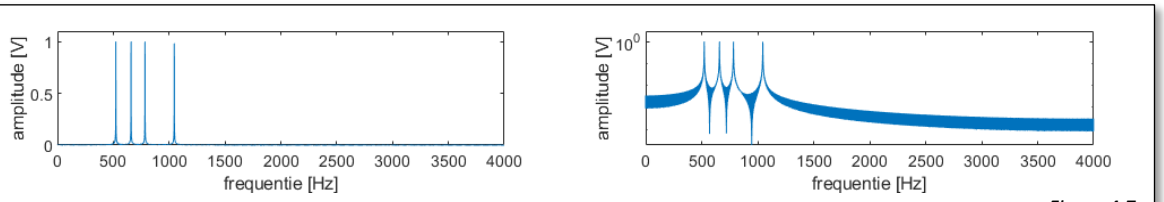
Gegeven een audiosignaal, met het geluid van vier ‘gong’-geluiden, in oplopende toonhoogte (feitelijk de tonen van een C-akkoord: C, E, G en C.

Als we een signaal in het tijddomein bekijken, dan zien we alleen de samplewaarden, ofwel de amplitude van het signaal. Op vier momenten in de tijd krijgt de amplitude een waarde, die telkens exponentieel afneemt. Door in te zoomen, kun je zien dat het om een sinusvormig signaal gaat. Vanuit dit plaatje is het lastig om de frequenties van de gong-tonen te bepalen.



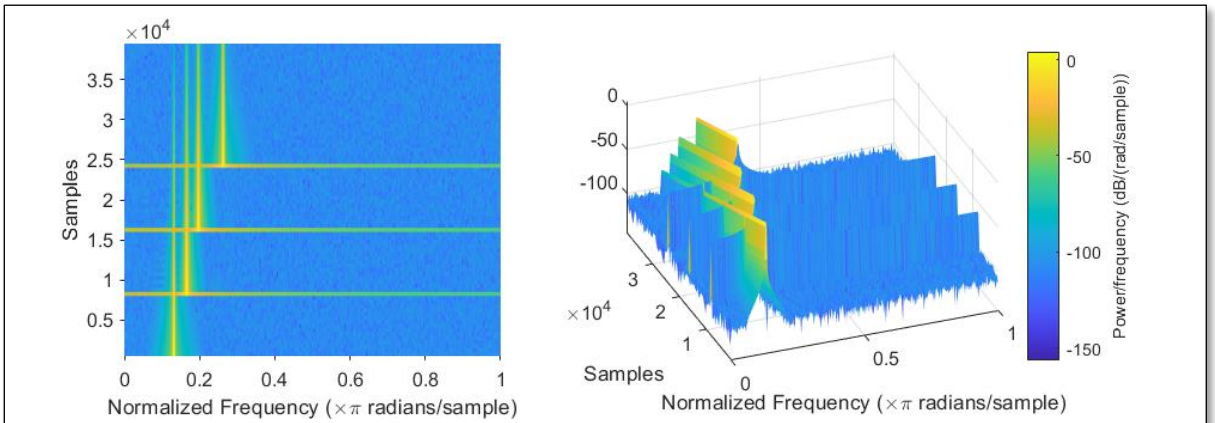
Figuur 4.6

Met behulp van de Fourier transformatie kan het signaal ook in het frequentiespectrum bekeken worden. De amplitude van het signaal kan lineair of logaritmisch worden uitgezet. In beide gevallen is duidelijk te zien dat er vier belangrijke frequenties in het signaal zitten. Bij de logaritmische weergave is ook te zien dat er componenten met andere frequenties in het signaal zitten (wel met kleinere amplitudes). Dit moet ook wel, omdat het signaal nu eenmaal niet bestaat uit vier 'pure' oneindig lange sinusoïden, maar uit sinusoïden die variëren in amplitude en in een bepaalde volgorde in het signaal zitten.



Figuur 4.7

Een andere manier om het signaal te bekijken, is in de vorm van een *spectrogram*. In het figuur hieronder is een spectrogram weergegeven. Het is een 3D-plaatje, met een tijd- én een frequentie-as. De amplitude wordt met een kleur weergegeven.



Figuur 4.8

In bovenstaand figuur zijn de vier gong-tonen te zien, als de verticale gele lijnen. Op de x-as kan de frequentie van elk van de tonen afgelezen worden. Het is een genormaliseerde frequentie-as, die loopt van 0 … π [rad/sample], ofwel van 0 Hz tot de halve samplefrequentie.

De y-as is de tijd-as, met als eenheid het aantal samples. Het is duidelijk te zien dat de gele lijnen vager worden, naarmate de tijd verstrijkt. De amplitude van elke toon neemt exponentieel af. In het rechter-plaatje lijkt het alsof de amplitude van de tonen lineair afneemt, maar de amplitudeschaal is logaritmisch, en een e-macht wordt op een logaritmische schaal als een rechte lijn weergegeven.

In het spectrogram is ook te zien dat een toon niet stopt als de volgende toon begint, maar dat de tonen gelijktijdig hoorbaar zijn. Dat soort informatie is heel lastig om uit de andere weergaven te halen.

Om een spectrogram te maken, wordt het signaal eerst in kleine stukjes verdeeld, bijvoorbeeld van een paar milliseconden. Van elk stukje wordt de Fourier transformatie berekend en het resultaat daarvan wordt weergegeven op een horizontale regel in het spectrogram. Deze techniek wordt *Short Time Fourier Transform (STFT)* genoemd. We komen hier later nog op terug.

Analyse en ontwerp van FIR-filters

Inleiding

Een belangrijk doel bij het analyseren van filters is het bepalen van de frequentiekarakteristiek. Om deze karakteristiek te bepalen, wordt een signaal op de ingang van een filter aangeboden waarin alle frequenties voorkomen. In het uitgangssignaal is dan te zien in welke mate frequentiecomponenten versterkt of verzwakt worden.

Bij het ontwerpen van filters gebeurt het omgekeerde. Eerst wordt de gewenste frequentiekarakteristiek vastgesteld en van daaruit worden dan de filtercoëfficiënten bepaald.

Frequentiespectrum van de deltafunctie

De impulsresponsie van een FIR-filter heeft dezelfde waarden als de coëfficiënten van het filter, en bevat alle informatie over het filter. Zowel bij de analyse van filters als bij het ontwerp ervan speelt de deltafunctie een belangrijke rol. Het frequentiespectrum van de deltafunctie is eenvoudig te bepalen met de DFT.

$$\Delta[k] = \sum_{n=0}^{N-1} \delta[n] \cdot e^{-j\omega_k n} = 1 \cdot e^{-j\omega_k 0} + 0 + 0 + 0 + 0 \ldots = 1$$

De deltafunctie heeft alleen een waarde bij *n = 0*, dus de meeste termen van het resultaat van de DFT hebben de waarde nul. Het frequentiespectrum $\Delta[k]$ heeft een constante waarde 1, die dus onafhankelijk is van de frequentie. Alle frequentiecomponenten hebben de waarde 1, dus de deltafunctie bevat alle frequenties.

Het spectrum van een verschoven deltafunctie ziet er als volgt uit.

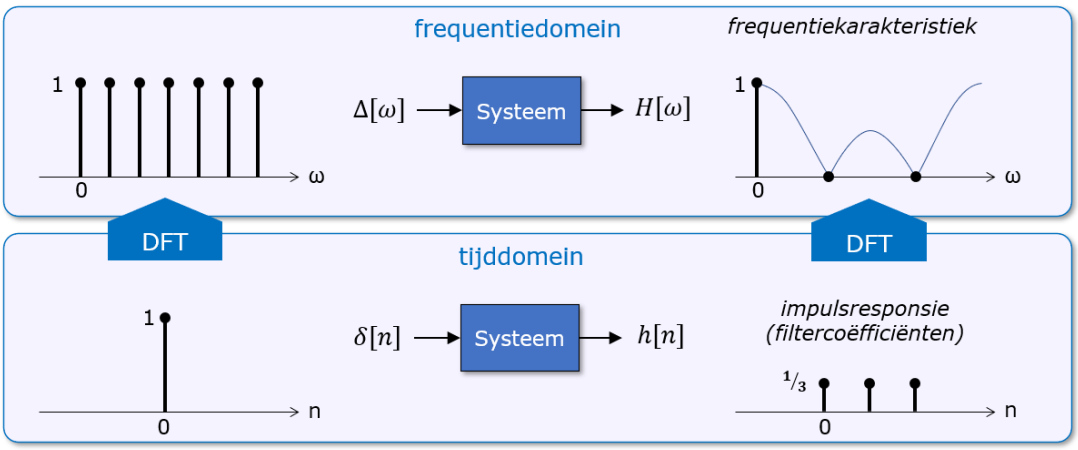
$$\Delta[k] = \sum_{n=0}^{N-1} \delta[n-1] \cdot e^{-j\omega_k n} = 0 + 1 \cdot e^{-j\omega_k 1} + 0 + 0 + 0 \ldots = e^{-j\omega_k}$$

De amplitude van $e^{-j\omega_k}$ is 1, en is onafhankelijk van de frequentie. In de verschoven deltafunctie zitten dus ook alle frequenties. De fase van $e^{-j\omega_k}$ is $-\omega_k$, en die is recht evenredig met de frequentie. De fase-karakteristiek van de verschoven deltafunctie is een rechte lijn, hetgeen overeenkomt met een vaste tijdvertraging. Dit is precies wat verwacht kon worden, omdat de deltafunctie verschoven is in de tijd. Overigens komt het antwoord ook overeen met hetgeen we weten van de tijdverschuivingseigenschap van de Fourier transformatie.

De deltafunctie bevat alle frequenties. Het is dan ook niet verwonderlijk dat de impulsresponsie alle informatie van het filter bevat. Het is namelijk de responsie op een ingangssignaal dat alle frequenties bevat.

Frequentiespectrum van een FIR-filter

De werking van een FIR-filter kan op twee manieren bekeken worden. Ten eerste vanuit het tijddomein: als een deltafunctie op de ingang van het filter wordt aangeboden, dan zien we op de uitgang de impulsresponsie, ofwel de filtercoëfficiënten. We kunnen de werking van het filter ook vanuit het frequentiedomein bekijken: als we op de ingang van het filter een signaal aanbieden waarin alle frequenties aanwezig zijn (zoals een deltafunctie), dan zien we op de uitgang van het filter in welke mate frequentiecomponenten versterkt of verzwakt worden, ofwel de frequentiekarakteristiek.



Figuur 5.1

Om van het tijddomein naar het frequentiedomein te transformeren, wordt de DFT gebruikt. De frequentiekarakteristiek van het filter kan dus bepaald worden door de DFT van de filtercoëfficiënten te nemen.

Als er maar weinig coëfficiënten zijn, dan wordt de karakteristiek van het filter ook met weinig componenten (*frequency bins*) beschreven. Bij een filter met drie coëfficiënten, bijvoorbeeld { ⅓, ⅓, ⅓ }, wordt het hele spectrum (van 0 Hz tót de samplefrequentie) beschreven met slechts drie waarden. De frequency bins zijn dan relatief breed, waardoor de karakteristiek van het filter niet duidelijk is. Er wordt daarom bij het analyseren van filters vaak *zero padding* toegepast, zodat de karakteristiek met veel meer frequency bins beschreven wordt.

Dit is een voorbeeld van het bepalen van de frequentiekarakteristiek van een filter. Merk op dat de vector *h* in dit voorbeeld niet gewijzigd wordt. Deze vector bevat namelijk de coëfficiënten van het filter. Er worden alleen nullen meegegeven bij de aanroep van de *fft()*-functie.

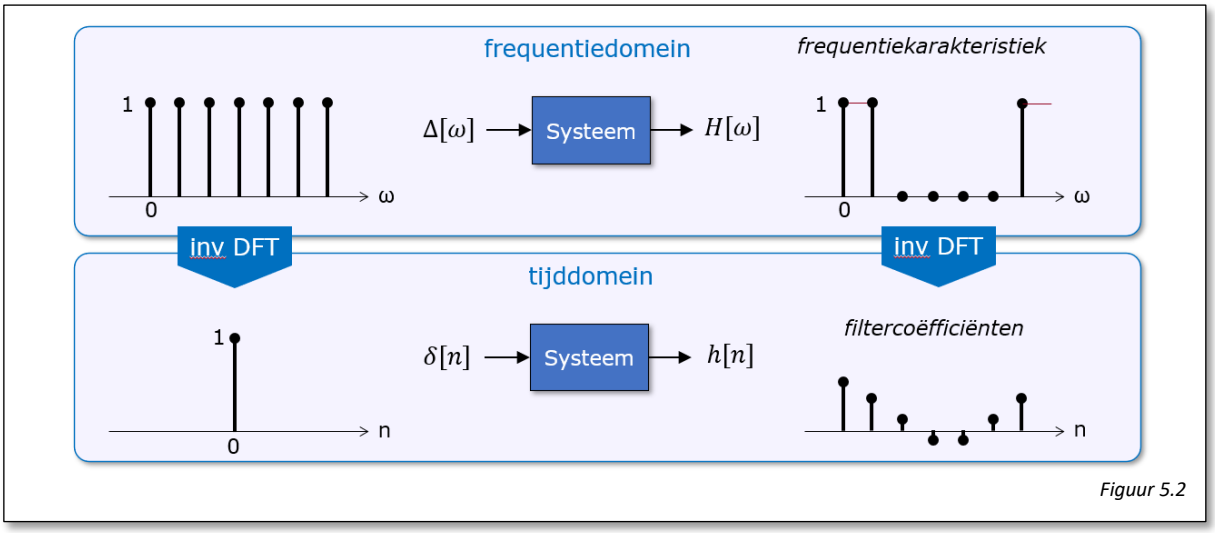
```
h=[1 1 1]/3;
H=fft([h zeros(1,1000)]);    % het getal 1000 is arbitrair
plot(abs(H));                % plot de amplitudekarakteristiek
                             % van 0 Hz tót de samplefrequentie
```

Let weer op het gebruik van kleine letters (tijddomein) en hoofdletters (frequentiedomein).

De frequentiekarakteristiek bestaat uit informatie over de amplitude (*amplitudekarakteristiek*) en over de fase (*fasekarakteristiek*). Meestal wordt de amplitudekarakteristiek als de belangrijkste eigenschap van een filter gezien. Informatie over de fase kan gebruikt worden als de tijdvertraging van het filter van belang is.

Ontwerp met de Fouriertransformatiemethode

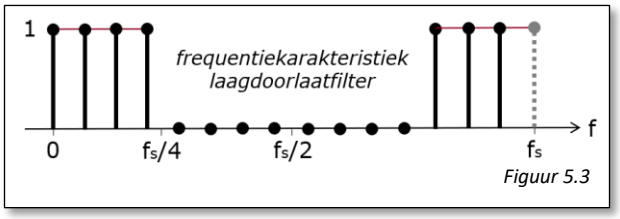
Bij analyse van een FIR-filter hebben we gezien dat de frequentiekarakteristiek bepaald kan worden door de DFT van de filtercoëfficiënten te nemen. Bij het ontwerpen van een FIR-filter stellen we eerst de gewenste frequentiekarakteristiek vast en berekenen vervolgens de inverse DFT (IDFT) om de filter-coëfficiënten te vinden.



Figuur 5.2

De gewenste frequentiekarakteristiek moet daarbij beschreven worden van 0 Hz tót de samplefrequentie. De karakteristiek van een laagdoorlaatfilter kan er bijvoorbeeld als volgt uitzien.

$$H = [1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1]$$

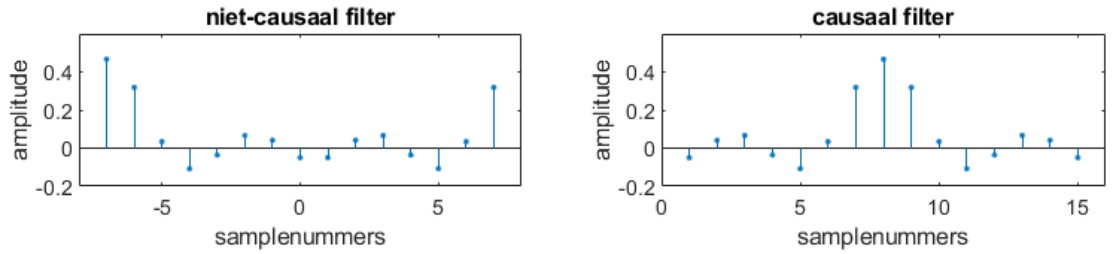


Figuur 5.3

De eerste vier enen geven aan dat de frequenties tot een kwart van de samplefrequentie doorgelaten moeten worden (ofwel met één vermenigvuldigd moeten worden). De laatste drie enen geven hetzelfde gebied aan, maar dan gespiegeld rondom de halve samplefrequentie. Dat er maar drie enen worden gebruikt, komt omdat het spectrum tót de samplefrequentie moet worden beschreven. Het spectrum is periodiek en herhaalt zich dus rondom veelvouden van de samplefrequentie. De versterking bij de samplefrequentie is hetzelfde als de DC-versterking, en deze is al beschreven met de eerste één.

Om de filtercoëfficiënten te bepalen kan de IDFT van de frequentiekarakteristiek genomen worden. Als de karakteristiek, zoals hierboven, met 15 frequency bins is beschreven, dan levert de IDFT een beschrijving van de impulsresponsie met 15 samples. Deze coëfficiënten zijn echter gecentreerd rondom tijdstip 0. De samples zijn genummerd van -7 tot en met 7. Het is dus eigenlijk een niet-causaal filter.

De DFT en de IDFT werken zowel in het tijddomein als in het frequentiedomein met discrete waarden, en zijn dus in beide domeinen ook periodiek. Om het gevonden filter causaal te maken, moeten de coëfficiënten verschoven worden, zodanig dat de eerste coëfficiënt op tijdstip 0 terecht komt. In het voorbeeld hierboven moeten de coëfficiënten 7 plaatsen verschoven worden.



Figuur 5.4

Om in Matlab een eenvoudig 14e-orde FIR-filter met een laagdoorlaatkarakteristiek te maken, kan de volgende code gebruikt worden.

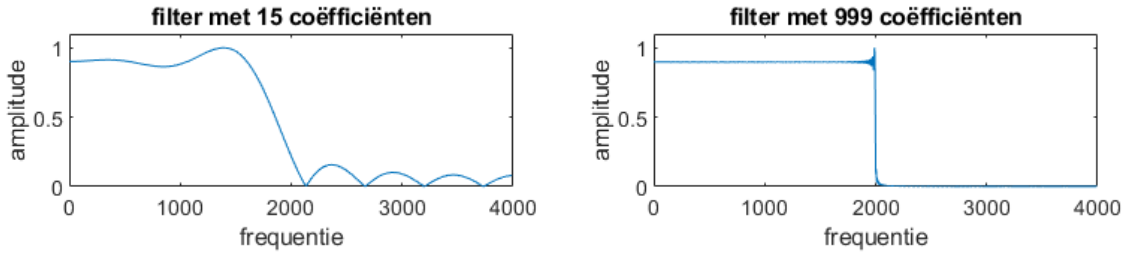
```
H=[1 1 1 1 0 0 0 0 0 0 0 0 1 1 1];  % frequentiekarakteristiek
h=ifft(H);                          % inverse DFT
h=fftshift(h);                      % verschuiven van coëfficiënten
```

De coëfficiënten van het filter staan nu in *h*. Let weer op het gebruik van kleine letters (tijddomein) en hoofdletters (frequentiedomein).

Om het filter te gebruiken kan de functie *conv()*gebruikt worden.

```
[x fs]=audioread('music.mp3');      % lees muziekbestand in
y=conv(x,h);                        % voer convolutie uit
sound(y,fs);                        % speel resultaat af
```

Het filter kan geanalyseerd worden door de DFT te nemen van de gevonden coëfficiënten. Als zero padding wordt toegepast, levert dat een frequentiekarakteristiek op zoals te zien in het linker diagram van onderstaand figuur.



Figuur 5.5

Door een filter met meer coëfficiënten te maken, lijkt de karakteristiek al veel meer op die van een ideaal filter, zoals te zien is in het rechter diagram in bovenstaand figuur.

Een frequentiekarakteristiek kan in Matlab ook op de volgende manier beschreven worden.

```
H=[ones(1,250) zeros(1,250)];       % karakteristiek(tot fs/2)
H=[H fliplr(H)];                    % gespiegeld rondom fs/2
H=H(1:length(H)-1);                 % laatste bin weghalen

h=ifft(H);                          % inverse DFT
h=fftshift(h);                      % verschuiven van coëfficiënten
```

Het filter heeft (250 + 250)·2 – 1 ofwel 999 coëfficiënten.

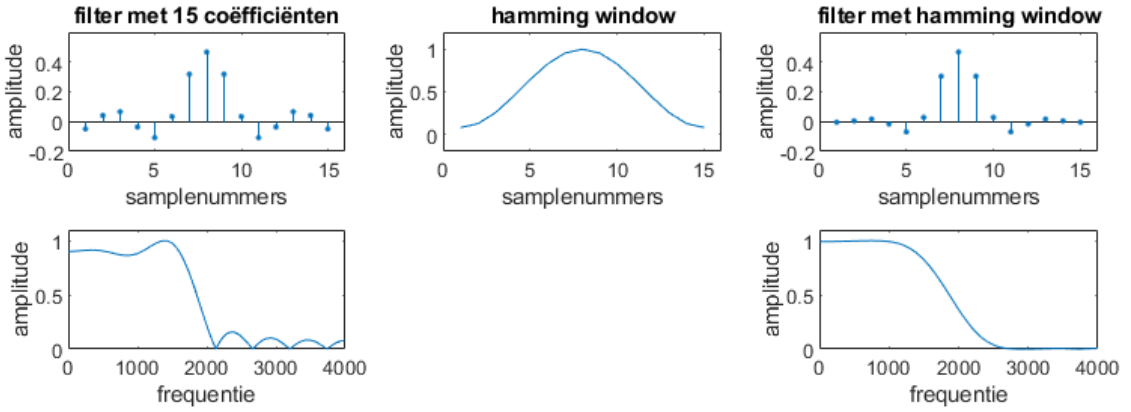
Windows

Als filtercoëfficiënten berekend worden met de *Fouriertransformatiemethode*, zoals hiervoor beschreven, dan zit er nog een 'rimpel' in de frequentiekarakteristiek van het filter. Als er meer coëfficiënten worden gebruikt, dan beperkt de rimpel zich tot een kleiner frequentiegebied, maar de amplitude van de rimpel blijft even groot. Dit wordt wel het *Gibbs-fenomeen* genoemd.

Een ideaal filter heeft in het frequentiedomein een oneindig steile overgang tussen de passband en de stopband. Om een dergelijk filter te maken zijn oneindig veel coëfficiënten nodig. We gebruiken slechts een beperkt aantal van deze coëfficiënten. Daardoor lijkt het alsof de coëfficiëntenreeks met een blokvormig signaal vermenigvuldigd is. De frequentiekarakteristiek van het filter wordt dan ook niet alleen bepaald door de (in theorie niet-periodieke, dus oneindig lange) impulsresponsie, maar ook door de grootte van het blok waarmee de impulsresponsie vermenigvuldigd is. Door de vermenigvuldiging lijkt het alsof de impulsresponsie wordt afgekapt, en een begin en een eind heeft.

De oplossing is conceptueel heel eenvoudig. In plaats van een vermenigvuldiging met een blok, kunnen we een vermenigvuldiging gebruiken met een vorm die minder steile flanken heeft. Deze vorm wordt *window* genoemd. Er zijn in het verleden verschillende windows ontworpen en deze zijn genoemd naar de ontwikkelaars. Een paar bekende windows zijn *hamming*, *hann*, *blackman* en *kaiser*.

In onderstaand figuur is het resultaat te zien van de vermenigvuldiging van de filtercoëfficiënten met een hamming window. Het is duidelijk te zien dat de eerste en laatste coëfficiënten wat kleiner zijn geworden, waardoor er een minder steil begin en eind van de coëfficiëntenreeks is ontstaan. In de frequentie-karakteristiek is duidelijk te zien dat de rimpel weg is, maar dat de transitieband iets minder steil is geworden.



Figuur 5.6

Verschillende windows leveren iets andere karakteristieken op. Bij een filter met veel coëfficiënten levert een hamming window een karakteristiek op met een vrijwel rechte passband (zonder rimpel).

Bij gebruik van een window worden de coëfficiënten tijdens het ontwerp eenmalig aangepast. De tijdvertraging van het filter blijft gelijk.

Dit is een voorbeeld van een hoogdoorlaatfilter met hamming window.

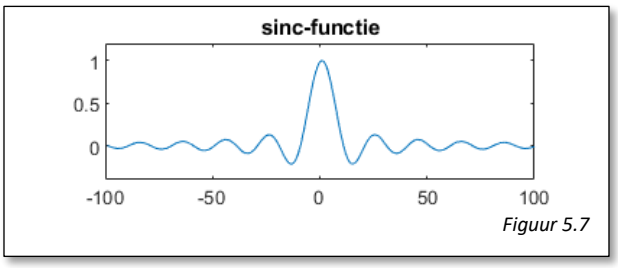
```
H=[0 0 0 0 1 1 1 1 1 1 1 1 0 0 0];  % hoogdoorlaatfilter

h=ifft(H);                          % inverse DFT
h=fftshift(h);                      % verschuiven van coëfficiënten
h=h.*hamming(length(h))';           % vermenigvuldiging met window
                                    % let op het quootje om de window
                                    % vector 90 graden te draaien
```

Ontwerp met windowed sinc methode

Een ideaal laagdoorlaatfilter heeft een rechthoekige frequentiekarakteristiek. De filtercoëfficiënten van zo'n filter hebben de vorm van een *sinc*-functie.

$$\operatorname{sinc} x = \frac{\sin \pi x}{\pi x}$$

Deze functie wordt steeds kleiner naarmate de waarde van *x* groter wordt (of negatiever). Voor $x = 0$ levert de formule $0/0$ op. De waarde van de limiet van $x \to 0$ is gelijk aan 1.



Figuur 5.7

Een rechthoek in het frequentiedomein geeft filtercoëfficiënten in de vorm van een *sinc*-functie in het tijddomein. Er bestaat een bijzondere relatie tussen de rechthoek en de *sinc*. Andersom geldt namelijk ook dat een rechthoek in het tijddomein een *sinc*-functie in het frequentiedomein geeft. De afleiding van deze relatie is te vinden in bijlage 2 van dit document.

Als bekend is dat een rechthoek in het frequentiedomein altijd een sinc-functie in het tijddomein oplevert, dan kunnen we een laagdoorlaatfilter maken zonder de (inverse) Fourier transformatie uit te voeren. De filtercoëfficiënten kunnen namelijk direct door een *sinc*-functie worden gevonden.

Het aantal samples dat gebruikt wordt om de *sinc* te beschrijven is gelijk aan het aantal coëfficiënten. Hoe meer coëfficiënten het filter heeft, hoe steiler de overgang van passband naar stopband gemaakt kan worden. De frequentie van de *sinc* bepaalt de breedte van de passband. Hoe hoger de frequentie van de *sinc*, hoe breder de passband.

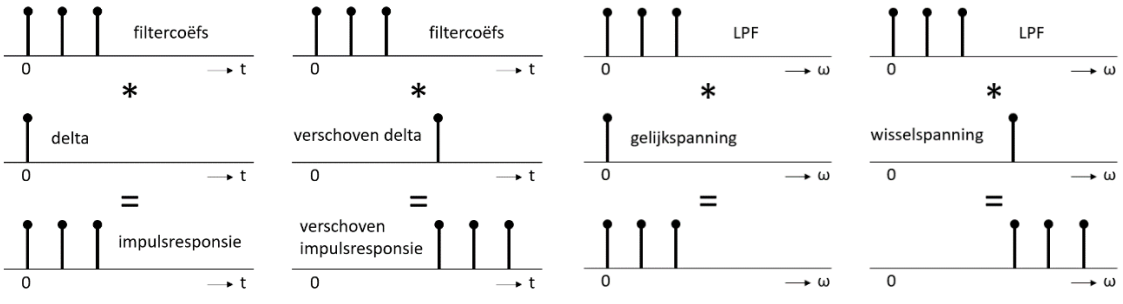
Dit is een voorbeeld van het berekenen van filtercoëfficiënten met de *sinc*-functie.

```
n=-pi:0.1:pi;                 % 'tijd'-basis voor sinc
x=sinc(n*5);                  % sinc-functie met 'frequentie' 5 (0…10)
x=x.*hamming(length(x))';     % vermenigvuldiging met window
```

Convolutie in tijddomein en in frequentiedomein

Stel dat een deltafunctie aan een filter wordt aangeboden. Het filter doet een convolutie en het resultaat is de impulsresponsie. Als een in de tijd verschoven deltafunctie wordt aangeboden, dan zal ook de impulsresponsie in de tijd verschuiven.

Hetzelfde effect zien we in het frequentiedomein. Als we in het frequentiedomein een convolutie uitvoeren van een frequentieband met een gelijkspanning (een constante waarde), dan zal de frequentieband niet verschuiven. Als we een convolutie uitvoeren van een frequentieband met een wisselspanning, dan is het resultaat een verschoven frequentieband.



Convolutie in tijddomein

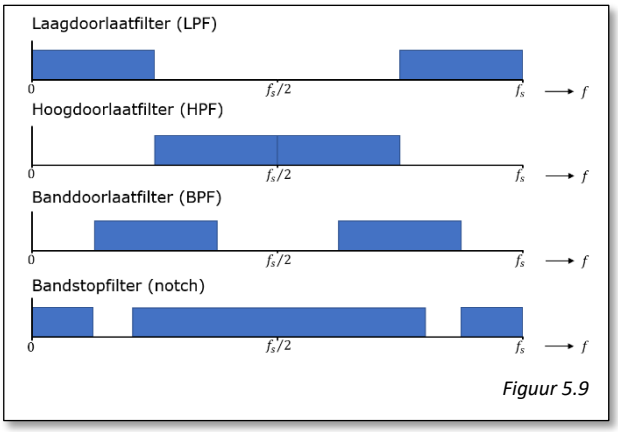
Convolutie in frequentiedomein

Figuur 5.8

Eén van de eigenschappen van de Fourier transformatie is de convolutie-eigenschap: een convolutie in het tijddomein is hetzelfde als een vermenigvuldiging in het frequentiedomein. Het omgekeerde is ook het geval. Een convolutie in het frequentiedomein is hetzelfde als een vermenigvuldiging in het tijddomein. Het bewijs hiervoor is te vinden in bijlage 1 van dit document.

Door een convolutie in het frequentiedomein te doen met een sinusoïde (een wisselspanning) kan een frequentieband worden verschoven. Het doorlaatgebied van een filter kan dus verschoven worden door de filtercoëfficiënten te vermenigvuldigen met een sinusoïde.

Zoals we eerder hebben gezien zijn er vier filter-karakteristieken die veel gebruikt worden. Deze bestaan allemaal uit rechthoeken en kunnen dus met de windowed-sinc methode ontworpen worden.



Figuur 5.9

Een hoogdoorlaatfilter of een banddoorlaatfilter kan worden gemaakt door de coëfficiënten van een laagdoorlaatfilter te vermenigvuldigen met een sinusoïde.

Een banddoorlaatfilter of een notch filter kan worden gemaakt door een laag- en een hoog-doorlaatfilter te combineren. De coëfficiënten van deze filters kunnen bij elkaar opgeteld worden om het coëfficiënten van het banddoorlaatfilter te verkrijgen (distributieve eigenschap van de convolutie). De versterking in de passband van beide filters is bij deze methode niet hetzelfde en moet dus nog aangepast worden.

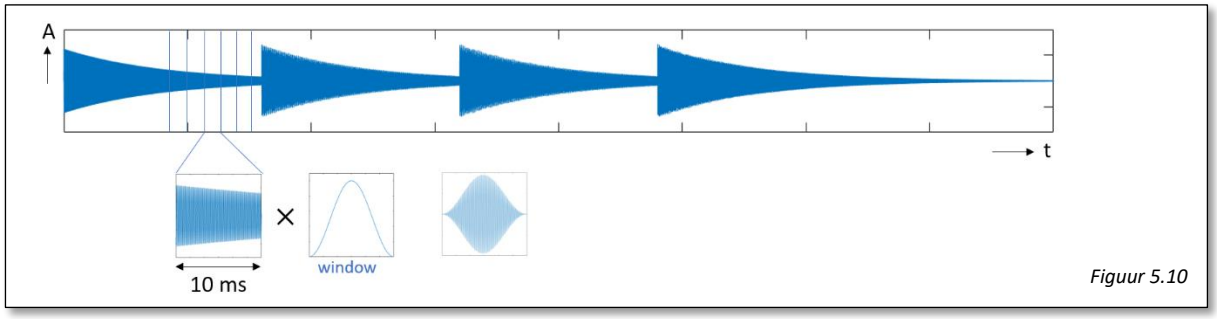
Om een filter te ontwerpen kan het best de Fouriertransformatiemethode worden gebruikt. Hiermee kan een filter met elke gewenste amplitudekarakteristiek gemaakt worden. De hier beschreven sinc-methode vergt wat minder berekeningen, maar dat is bij het eenmalig ontwerp van een filter niet zo belangrijk. Het kan wel belangrijk zijn bij een filter waarvan de filtercoëfficiënten real-time aangepast moeten worden.

Short-time Fourier Transform

Signalen en systemen kunnen op verschillende manieren geanalyseerd worden. We kunnen bijvoorbeeld signalen of impulsresponsies bekijken in het tijddomein. Door een Fourier transformatie uit te voeren kunnen we het spectrum van signalen en systemen bekijken in het frequentiedomein. Soms is het nodig om een signaal of systeem in zowel het tijd- als in het frequentiedomein te bekijken. We hebben al eerder gezien dat we daar een *spectrogram* voor kunnen gebruiken. Bij een spectrogram staat de frequentie op de x-as en de tijd op de y-as. De amplitude (of eventueel de fase) kan op de z-as of door middel van kleur worden weergegeven.

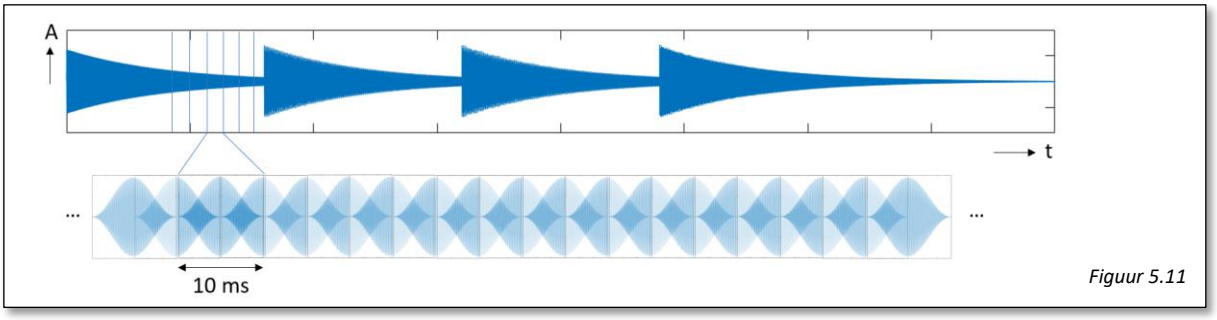
Om een spectrogram te maken wordt een signaal (in de tijd) verdeeld in kleine stukjes. Bij een muziek-signaal kunnen bijvoorbeeld stukjes van 10 of 20 ms worden gebruikt. Van elk stukje signaal wordt de DFT berekend, en het resultaat (bijvoorbeeld de amplitude) wordt gebruikt om telkens één regel van het spectrogram te tekenen.

De DFT beschouwt het kleine signaalstukje als een periodiek signaal, dat zichzelf dus continu herhaalt. In de meeste gevallen zullen het begin en het eind van het signaalstukje niet op elkaar aansluiten. Er zitten dus ongewenste discontinuïteiten in het signaal, die ook in het frequentiedomein zichtbaar zijn. Door het signaalstukje te vermenigvuldigen met een window, worden de amplitude van het begin en van het eind van het signaalstukje kleiner, waardoor de discontinuïteiten minder prominent in het signaal aanwezig zijn.



Figuur 5.10

Stel dat er signaalstukjes van 10 ms worden gebruikt. Na de toepassing van het window, zal er een duidelijke frequentiecomponent van 1 / 10 ms ofwel 100 Hz in het spectrum zichtbaar zijn, terwijl deze frequentie wellicht niet in het originele signaal voorkomt. Dit komt doordat het window de amplitude van het begin en van het eind van het signaalstukje verzwakt. Door overlappende signaalstukjes te gebruiken wordt dit effect verminderd.

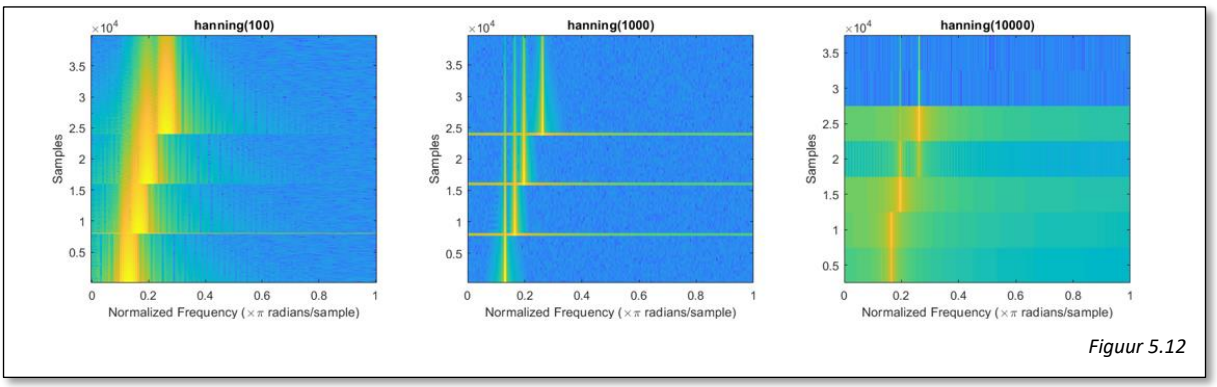


Figuur 5.11

Door telkens de DFT te nemen van overlappende signaalstukjes (met window) kan het spectrogram worden opgebouwd. Deze techniek wordt Short-time Fourier Transform (STFT) genoemd.

Als er langere signaalstukjes worden gebruikt, dan levert de DFT meer informatie op over het frequentie-spectrum van het betreffende stukje. De frequentie kan dan nauwkeurig bepaald worden. Aan de andere kant kan bij langere signaalstukjes niet nauwkeurig bepaald worden op welke moment een bepaalde frequentiecomponent in het signaal begint of eindigt. Dat soort informatie kan juist nauwkeurig bepaald worden bij gebruik van kleine signaalstukjes.

In onderstaand figuur zijn spectrogrammen te zien van een signaal met vier tonen, die elk op een verschillend moment starten. In het plaatje links zijn signaalstukjes met 100 samples gebruikt (vermenigvuldigd met een hanning window). Het startmoment van de tonen is nauwkeurig te bepalen, maar de gebruikte frequentie niet. In het plaatje rechts zijn signaalstukjes van 10000 samples gebruikt. Daar is de frequentie nauwkeurig te bepalen, maar het startmoment van de tonen niet.



Figuur 5.12

Realisatie van FIR-filters

Inleiding

Digitale filters zoals FIR-filters kunnen geïmplementeerd worden met behulp van een programmeertaal zoals C of C++. Ze kunnen dan draaien op een gewone processor of microcontroller, of op een DSP. Het is ook mogelijk om digitale filters in een FPGA te realiseren.

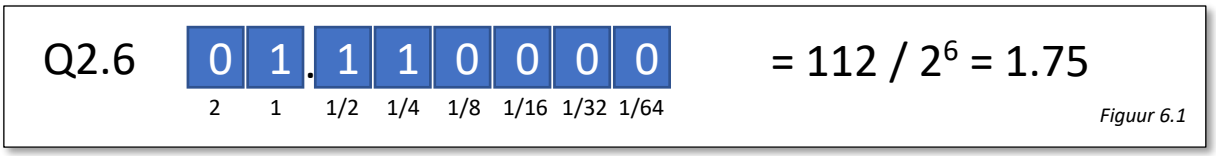
Fixed point

De berekeningen die door een filter uitgevoerd worden, zijn vooral vermenigvuldigingen (van de samples in de delay line met de filtercoëfficiënten) en optellingen. Als de coëfficiënten in floating point format staan, dan is het efficiënt als de processor de berekeningen hardwarematig kan uitvoeren. Daar is een floating point unit (FPU) in de processor voor nodig, maar niet alle processoren hebben zo'n unit. Voor een processor zonder FPU is minder hardware (silicium) nodig, en deze kan daardoor een stuk goedkoper zijn. Floating point berekeningen moeten in dat geval softwarematig uitgevoerd worden en dat is veel trager dan een berekening door een hardwarematige FPU. Er bestaat ook de mogelijkheid om fixed point berekeningen te gebruiken. Feitelijk wordt er dan met integers gewerkt en dit wordt door alle processoren ondersteund.

Er zijn dus drie mogelijke situaties:

- Floating point berekeningen met processor mét FPU (duur, maar snel)
- Floating point berekeningen met een processor zónder FPU (goedkoop, maar traag)
- Fixed point berekeningen (kan met alle processoren, goedkoop, snel)

Bij het fixed point format wordt er gerekend met integers, waarbij er een 'denkbeeldige' decimale komma wordt gebruikt. Een voorbeeld van een 8-bits variabele met twee bits vóór en zes bits achter de komma ziet er bijvoorbeeld als volgt uit.



Figuur 6.1

Dit format wordt Q2.6 genoemd, naar het aantal bits voor en achter de decimale komma. Door deze denkbeeldige decimale komma te gebruiken, worden de weegfactoren van de bits veranderd. Bij een normaal binair getal worden integer weegfactoren gebruikt. De decimale komma staat dan eigenlijk helemaal rechts van het getal. In dit voorbeeld in de decimale komma zes plaatsen naar links geschoven, waardoor de weegfactoren allemaal een factor 2^6 kleiner zijn geworden.

Door met fixed point variabelen te werken kunnen niet-integer getallen worden gebruikt, terwijl de processor alleen met integers hoeft te rekenen. Er zijn allerlei verschillende formats denkbaar. Een veelgebruikt format is Q15, ofwel Q1.15. Het bit voor de komma is een sign bit. In een Q15-getal kunnen in een *short int* waarden tussen -1 en 1 worden opgeslagen, met een resolutie van $1/2^{15}$ ofwel ~0.00003. Het bereik van een Q15-waarde is beperkt, maar de resolutie is erg goed. Als het Q31-format wordt gebruikt (een *long int*) dan wordt de resolutie beter dan $5 \cdot 10^{-10}$.

Het bereik van een fixed point variabele wordt vooral bepaald door het aantal bits vóór de decimale komma. Bij een signed variabele in Q2.6 kan met de bits voor de komma een waarde worden ingesteld van [-1, 1]. Samen met de bits achter de komma is het bereik [-2, 2>, ofwel [-2, 1.984375].

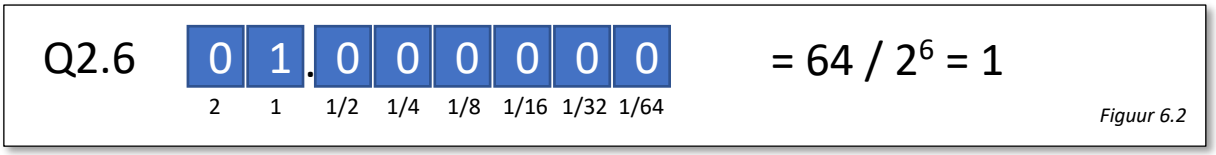
De resolutie, of stapgrootte, van een fixed point waarde wordt vooral bepaald door het aantal bits ná de decimale komma. Bij een variabele in Q2.6 is de kleinste stap 2^{-6}, ofwel 0.015625.

Het format van fixed point variabelen wordt ook wel op een andere manier aangeduid. Een format zoals Q15 of Q1.15 kan ook aangeduid worden met S16Q15. De S betekent dat het om een signed waarde gaat, en het getal 16 is het totaal aantal bits dat door de variabele wordt gebruikt.

Binnen een DSP-systeem kunnen variabelen met verschillende Q-formaten worden gebruikt. Bij het rekenen met fixed point variabelen moet dan natuurlijk wel rekening gehouden worden met de plaats van de decimale komma. Overigens gaat het om een denkbeeldige komma, waar de processor niets van weet.

Het optellen van fixed point getallen (in hetzelfde Q-format) is exact hetzelfde als het optellen van integers. Als de getallen in een verschillend Q-format staan, dan moeten de bits opgeschoven worden, zodat de decimale komma van beide getallen op dezelfde plaats staan.

Bij het vermenigvuldigen van fixed point getallen is altijd een schaling nodig. Stel dat we twee variabelen in Q2.6-format hebben, elk met de waarde 1.



Figuur 6.2

De processor ziet een getal met bits 0100 0000 als de waarde 64. Als de twee variabelen met elkaar vermenigvuldigd worden, dan is het resultaat 64 · 64 = 4096. Echter, de fixed point waarde van beide getallen is 1, en de verwachte uitkomst van de vermenigvuldiging is dan ook 1. Het resultaat van de vermenigvuldiging staat nu eigenlijk in een format met 12 bits achter de komma. Als het resultaat in Q2.6-format moet staan, dan moet het resultaat gedeeld worden door 2^6 ofwel 64. Na elke vermenigvuldiging met fixed point getallen moet er geschaald worden, zodat het resultaat in het juiste format komt te staan.

Matlab heeft de mogelijkheid om met fixed point variabelen te werken, bijvoorbeeld met *fi()*.

```
a=fi(4)     % variabele a krijgt de waarde 4 in fixed point format
```

Het is natuurlijk mogelijk om te specificeren hoeveel bits er achter de komma gebruikt moeten worden (fractional length), maar als dat niet gebeurt dan kiest Matlab zelf een passend format. Bij bovenstaand commando zegt Matlab het volgende:

```
          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 16
        FractionLength: 12
```

Blijkbaar heeft Matlab in dit geval gekozen voor Q4.12 (16 bits met maximale fractional length).

Realisatie van FIR-filters

Filters kunnen in allerlei programmeertalen gerealiseerd worden. Om een FIR-filter te realiseren, is het voldoende om programmacode te schrijven om de convolutie uit te rekenen.

In de meeste DSP-systemen wordt een timer-interrupt gebruikt om de AD-converter op regelmatige tijdstippen aan te sturen. De code voor het verwerken van één sample kan dan in de interrupt service routine (ISR) worden gezet.

In het voorbeeld hiernaast is een FIR-filter met drie coëfficiënten *{ 1, 2, 1 }* geïmplementeerd.

Het nieuwe sample wordt gelezen van de ADC en in de delay line *x* geplaatst. Daarna worden de samples in *x* vermenigvuldigd met de filter-coëfficiënten in *b*. Het resultaat wordt opgeslagen in *y*.

Vervolgens worden de samples in de delay line elk één plaats opgeschoven en wordt de resultaat van de convolutie naar de DAC geschreven.

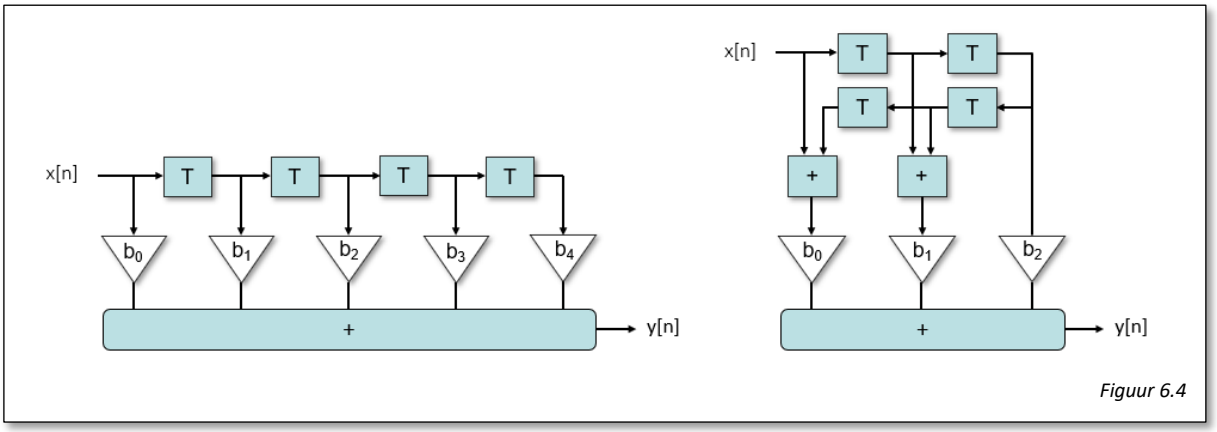
Het is van belang dat de code kort en efficiënt is, zodat een zo hoog mogelijke samplefrequentie gebruikt kan worden. De ISR moet uitgevoerd kunnen worden binnen één sampleperiode.
Er zijn verschillende manieren om de code te optimaliseren.

```
#define N 3
double x[N] = { 0 };
double b[N] = { 1, 2, 1 };

void adc_interrupt()
{
  x[0] = readADC();
  double y = 0;

  for (int i = 0; i < N; i++)
  {
    y += b[i] * x[i];
  }

  for (i = N - 2; i >= 0; i--)
  {
    x[i + 1] = x[i];
  }

  writeDAC(y);
}
```

Figuur 6.3

Het vermenigvuldigen van de samples met de filtercoëfficiënten en het verschuiven van samples in de delay line kan in één loop geplaatst worden. Het is mogelijk dat de code hier iets sneller door wordt. Het is ook mogelijk om een circulaire buffer te gebruiken voor de delay line. In plaats van telkens samples door de delay line te schuiven, kunnen een indexvariabele of een pointer worden gebruikt om de langs de samples in de delay line te lopen. Dit levert vooral tijdwinst op bij filters met veel coëfficiënten.

Verder is het zo dat de filtercoëfficiënten heel vaak gespiegeld zijn rond de middelste coëfficiënt. Bij een laagdoorlaatfilter hebben de coëfficiënten bijvoorbeeld de vorm van een sinc-functie, die symmetrisch is rond het midden. Daar zou bij de implementatie van het filter gebruik van kunnen worden gemaakt, door eerst bepaalde samples bij elkaar op te tellen, en daarna pas te vermenigvuldigen. Daarmee kan in veel gevallen wel de helft van het aantal vermenigvuldigingen worden bespaard. In onderstaand figuur zijn twee implementatiemogelijkheden van hetzelfde filter te zien. Vergelijk het aantal optellingen en het aantal vermenigvuldigingen van beide implementaties.



Figuur 6.4

Uiteraard is de keuze voor floating point of fixed point belangrijk voor de verwerkingssnelheid van het filter. Als een processor geen FPU heeft, dan zijn floating point berekeningen heel erg traag.

Bij gebruik van een DSP-processor met bijbehorende compiler, zal de optimizer in de meeste gevallen de code voor de convolutie herkennen en assembly-code genereren die optimaal gebruik maakt van de mogelijkheden van de processor. DSP-processoren hebben vaak hardwarematige ondersteuning voor het gebruik van circulaire buffers. Verder hebben ze vaak meerdere MAC's aan boord (Multiply-Accumulate units), zodat ze tegelijkertijd meerdere vermenigvuldigingen en optellingen kunnen uitvoeren. Dit is natuurlijk heel gunstig voor de snelheid van het filter.

Digitale filters kunnen ook in een FPGA gerealiseerd worden. Doordat alle bewerkingen dan parallel worden uitgevoerd, kunnen hele snelle filters gemaakt worden. De FPGA moet dan wel voldoende capaciteit hebben om vermenigvuldigingen uit te voeren. In bijlage 3 staat een voorbeeld van de VHDL-code voor een FIR-filter met drie coëfficiënten.

Bijlage 1 – Eigenschappen DFT

Bewijs voor lineariteitseigenschap DFT

$$DFT\{a \cdot x_1[n] + b \cdot x_2[n]\} = \sum_{n=0}^{N-1} (a \cdot x_1[n] + b \cdot x_2[n]) \cdot e^{-j\omega n} =$$

$$a \cdot \sum_{n=0}^{N-1} x_1[n] \cdot e^{-j\omega n} + b \cdot \sum_{n=0}^{N-1} x_2[n] \cdot e^{-j\omega n} = a \cdot X_1[\omega] + b \cdot X_2[\omega]$$

Bewijs voor tijdverschuivingseigenschap DFT

$$DFT\{x[n-m]\} = \sum_{n=0}^{N-1} x[n-m] \cdot e^{-j\omega n} =$$

$$stel\colon\ p = n - m \rightarrow n = p + m$$
$$dus\colon\ n = 0 \rightarrow p + m = 0 \rightarrow p = -m$$

$$\sum_{p=-m}^{N-1-m} x[p] \cdot e^{-j\omega(p+m)} = \sum_{p=0}^{N-1} x[p] \cdot e^{-j\omega p} \cdot e^{-j\omega m} = X[\omega] \cdot e^{-j\omega m}$$

Bewijs voor convolutie-eigenschap DFT

$$DFT\left\{\sum_{m=0}^{N-1} x_1[m] \cdot x_2[n-m]\right\} = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} (x_1[m] \cdot x_2[n-m]) \cdot e^{-j\omega n} =$$

$$\sum_{m=0}^{N-1} x_1[m] \cdot \sum_{n=0}^{N-1} x_2[n-m] \cdot e^{-j\omega n} = \sum_{m=0}^{N-1} x_1[m] \cdot X_2\,[k] \cdot e^{-j\omega m} =$$

$$\sum_{m=0}^{N-1} x_1[m] \cdot e^{-j\omega m} \cdot X_2[k] \ = X_1[k] \cdot X_2[k]$$

$$X_1[k] * X_2[k] = \sum_{m=-\infty}^{\infty} X_1[m] \cdot X_2[k-m] = \sum_{m=-\infty}^{\infty} \sum_{n=0}^{N-1} x_1[n] \cdot e^{-j\omega_m n} \cdot \sum_{n=0}^{N-1} x_2[n] \cdot e^{-j\omega_{k-m} n} =$$

$$\sum_{m=-\infty}^{\infty} \sum_{n=0}^{N-1} x_1[n] \cdot x_2[n] \cdot e^{-j\omega_m n} \cdot e^{-j\omega_k n} \cdot e^{-j\omega_{-m} n} = \sum_{n=0}^{N-1} x_1[n] \cdot x_2[n] \cdot e^{-j\omega_k n} = DFT\{x_1[n] \cdot x_2[n]\}$$

Bijlage 2 – Rechthoek ↔ sinc-functie

DFT van rechthoek van tijd- naar frequentiedomein

$$\text{rect}(t) = \begin{cases} 0, & t < -½ \\ 1, & -½ < t < ½ \\ 0, & t > ½ \end{cases}$$

$$X(\omega) = \int_{-\infty}^{\infty} rect(t) \cdot e^{-j\omega t} \cdot dt = \int_{-½}^{½} e^{-j\omega t} \cdot dt = \left[\frac{e^{-j\omega t}}{-j\omega}\right]_{-½}^{½} = \frac{e^{-j\frac{\omega}{2}} - e^{j\frac{\omega}{2}}}{-j\omega} =$$

$$= \frac{2}{\omega}\left(\frac{e^{j\frac{\omega}{2}} - e^{-j\frac{\omega}{2}}}{2j}\right) = \frac{2}{\omega} \cdot \sin\frac{\omega}{2} = \frac{\sin {}^{\omega}/_{2}}{{}^{\omega}/_{2}} \qquad X(f) = \frac{\sin \pi f}{\pi f} = \text{sinc}\, f$$

DFT van rechthoek van frequentie- naar tijddomein

$$\text{rect}(\omega) = \begin{cases} 0, & \omega < -½ \\ 1, & -½ < \omega < ½ \\ 0, & \omega > ½ \end{cases}$$

$$x(t) = \int_{-\infty}^{\infty} \text{rect}(\omega) \cdot e^{j\omega t} \cdot d\omega = \int_{-½}^{½} e^{j\omega t} \cdot d\omega = \left[\frac{e^{j\omega t}}{jt}\right]_{-½}^{½} = \frac{e^{j\frac{t}{2}} - e^{-j\frac{t}{2}}}{jt}$$

$$= \frac{2}{t}\left(\frac{e^{j\frac{t}{2}} - e^{-j\frac{t}{2}}}{2j}\right) = \frac{2}{t} \cdot \sin\frac{t}{2} = \frac{\sin {}^{t}/_{2}}{{}^{t}/_{2}} = \text{sinc}\, {}^{t}/_{2}$$

Bijlage 3 – Voorbeeld implementatie FIR-filter in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY filter IS
   PORT( clk:             IN std_logic;
         clk_enable:      IN std_logic;
         reset:           IN std_logic;
         filter_in:       IN real;
         filter_out:      OUT real );
END filter;

ARCHITECTURE rtl OF filter IS
  TYPE delay_pipeline_type IS ARRAY (NATURAL range <>) OF real;
  CONSTANT coeff1:        real := 0.33333;
  CONSTANT coeff2:        real := 0.33333;
  CONSTANT coeff3:        real := 0.33333;
  SIGNAL delay_pipeline:  delay_pipeline_type(0 TO 2) := (0.0,0.0, 0.0);
  SIGNAL product3:        real := 0.0;
  SIGNAL product2:        real := 0.0;
  SIGNAL product1_cast:   real := 0.0;
  SIGNAL product1:        real := 0.0;
  SIGNAL sum:             real := 0.0;
  SIGNAL output_register: real := 0.0;

BEGIN
  Delay_Pipeline_process: PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      delay_pipeline(0 TO 2) <= (OTHERS => 0.0);
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        delay_pipeline(0) <= filter_in;
        delay_pipeline(1 TO 2) <= delay_pipeline(0 TO 1);
      END IF;
    END IF;
  END PROCESS Delay_Pipeline_process;

  product3 <= delay_pipeline(2) * coeff3;
  product2 <= delay_pipeline(1) * coeff2;
  product1 <= delay_pipeline(0) * coeff1;
  product1_cast <= product1;

  sum <= product1_cast + product2 + product3;

  Output_Register_process: PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      output_register <= 0.0;
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        output_register <= sum;
      END IF;
    END IF;
  END PROCESS Output_Register_process;
  filter_out <= output_register;
END rtl;
```