Digital lydbehandling

Øyvind Hammer
Norsk nettverk for Teknologi, Akustikk og Musikk

Oslo, 6. januar 1997

Innhold

1 Innledning 4

2 Matematikk 6

2.1 Funksjoner . . . 6

2.1.1 Hva er en funksjon? . . . 6

2.1.2 Lineære funksjoner . . . 7

2.1.3 Polynomer . . . 7

2.1.4 Hyperbler og røtter . . . 10

2.1.5 Eksponensialfunksjoner, logaritmer . . . 11

2.2 Trigonometri . . . 14

2.2.1 Rettvinklede trekanter . . . 14

2.2.2 De trigonometriske funksjonene . . . 15

2.2.3 Fourier-rekker . . . 18

3 Akustikk 19

3.1 Hva er lyd? . . . 19

3.2 De tre lyd-parametrene . . . 20

3.2.1 Amplityde, lydstyrke . . . 20

3.2.2 Frekvens, tonehøyde . . . 21

3.2.3 Klangfarge . . . 21

3.2.4 Dynamikk . . . 25

3.3 Beats, kritiske bånd . . . 26

3.3.1 Kritiske bånd . . . 29

3.4 Spektre og tonalitet . . . 29

3.4.1 Overtoner og kritiske bånd . . . 29

3.4.2 Musikalske skalaer . . . 30

3.4.3 Harmoniske spektre, cocktailparty-effekten . . . 30

3.4.4 Virtuelle grunntoner . . . 31

3.5 Musikkinstrumentenes fysikk . . . 31

3.5.1 Gitar . . . 31

3.5.2 Fiolin . . . 33

3.5.3 Piper med en lukket og en åpen ende . . . 33

3.5.4 Trompet . . . 35

3.5.5 Menneskestemmen . . . 35

3.6 Lokalisering av lydkilder . . . 36

3.6.1 Avstanden til lydkilden . . . 36

3.6.2 Retningen til lydkilden . . . 37
3.7 Morsomme småting . . . 39
3.7.1 Cerebral hemisfære-dominans . . . 39
3.7.2 Shepard-toner . . . 39
3.7.3 Differanse-toner . . . 39
3.7.4 Sympatiske vibrasjoner . . . 39

4 Sampling og rekonstruksjon 40
4.1 Diskretisering . . . 40
4.2 Samplingsteoremet, aliasering . . . 40
4.3 Rekonstruksjon . . . 43
4.4 Kvantisering . . . 45

5 Digital lydbehandling 47
5.1 Lydbehandling og syntese i tidsdomenet . . . 47
5.1.1 Amplityde- og ringmodulasjon . . . 47
5.1.2 Miksing . . . 48
5.1.3 Kurveform-syntese . . . 48
5.1.4 Additiv syntese . . . 49
5.1.5 FM-syntese . . . 49
5.1.6 Ulineær kurveforming . . . 54
5.1.7 Amplityde-følging . . . 55
5.1.8 Pitch-følging . . . 59
5.2 Fysisk modellering . . . 59
5.2.1 En enkel fiolin-modell . . . 60
5.2.2 Karplus-Strong-algoritmen . . . 60
5.2.3 Waveguides . . . 61
5.2.4 Modal syntese . . . 63
5.3 Lydbehandling og syntese i frekvensdomenet . . . 63
5.3.1 Generelt om filtre . . . 64
5.3.2 Digital filtrering med differenslikninger . . . 67
5.3.3 FFT, spektralanalyse . . . 68
5.3.4 Vindusfunksjoner, overlapping . . . 72
5.3.5 Ting man kan gjøre med et spektrum . . . 72
5.3.6 IFFT, resyntese . . . 73
5.3.7 Fase-vokoderen . . . 73
5.3.8 Lineær-prediktiv koding . . . 73
5.3.9 Filterbanker . . . 74
5.4 Granulære metoder . . . 74
5.4.1 VOSIM-syntese . . . 75
5.4.2 FOF-syntese . . . 75
5.4.3 Granulering med vindusfunksjoner . . . 76
5.4.4 Granulering med nullgjennomgangsanalyse . . . 76
5.4.5 Granulering og FFT . . . 77
5.4.6 Wavelets . . . 77
5.5 Spatialisering . . . 77
5.5.1 Ekko . . . 77

5.5.2 Romklang 78
5.5.3 Plassering av lydkilder i rommet 79
5.5.4 Andre effekter 79
5.6 Analyse 80

6 C-sound **82**
6.1 Et lite eksempel 82
6.1.1 Header 83
6.1.2 Instrumentet 83
6.1.3 Partituret 84
6.2 Flere triks 84
6.2.1 P-felter 84
6.2.2 Envelopes, GENs, carry 85
6.3 Synteseteknikker 86
6.3.1 Ringmodulering 86
6.3.2 FM-syntese 87
6.3.3 Subtraktiv syntese 87
6.3.4 Waveshaping 88
6.3.5 FOF 89
6.3.6 Karplus-Strong 90
6.4 Lydbehandling 90
6.4.1 Kor 90
6.4.2 Global romklang 91
6.4.3 Fasevokoding 92
6.4.4 Ceres 92
6.4.5 Lineær-prediktiv koding 93
6.4.6 Deltoneanalyse/additiv resyntese 93
6.4.7 Videre 94

Kapittel 1

Innledning

Dette kompendiet er ment som en innføring i produksjon, omforming og analyse av *lyd* ved hjelp av datamaskiner. Uthevelsen er gjort for å markere forskjellen fra arbeid med *noter*, dvs. MIDI-teknologi, algoritmisk komposisjon og tradisjonelle notasjonsprogrammer. Vekten vil bli lagt på teoretiske aspekter, men det finnes også et kapittel som gir en innføring i lydbehandlingsprogrammet "CSound".

Kompendiet er omarbeidet fra hefter som ble skrevet i forbindelse med kurs avholdt på NoTAM våren 1994. Ingen forkunnskaper skulle være nødvendige. Mange lesere vil være godt kjent med innholdet i kapitlene om matematikk og akustikk; man kan da hoppe over disse uten tap av kontinuitet.

Dokumentet er skrevet ved hjelp av typesettingsprogrammet LaTeX. Grafene er laget med matematikkprogrammet MATLAB. Tex-filer og Postscript-filer er tilgjengelige via anonym FTP på notam.uio.no (gjør `cd pub/doc/kurs`).

Hva skal vi med digital lydbehandling?

Digital lydbehandling kan virke som et spesielt og teknisk fagfelt. Hvorfor er det nødvendig for komponister og musikkvitere å kjenne til dette?

Digital lydbehandling i skapende virksomhet

Med datamaskiner kan man syntetisere og omforme lyd ved hjelp av en lang rekke avanserte teknikker. Dette muliggjør produksjon av klanger som ville være umulige å frambringe på andre måter. Dessuten kan komponisten spesifisere lyden med det høyeste nivå av nøyaktighet og stringens, samtidig som det ved "uhell" og eksperimentering kan oppstå uventede og spennende lydbilder som kan anvendes i verket.

Disse mulighetene har gjort digital lydbehandling til et sentralt verktøy for samtidskomponister. "Computermusikk" er etterhvert en gammel (minst 50 år) og velfundert musikkgren med et stort repertoar. Ikke bare i komposisjon, men like mye for forståelse og analyse av denne musikken, kreves et godt grunnlag i den digitale lydbehandlingens begreper.

Digital lydbehandling for musikologisk analyse og notasjon

Digital lydbehandling er en viktig teknologi for musikologisk analyse. Folkemusikkvitere trenger f.eks. nøyaktige, publiserbare målinger av grunntonefrekvens, for å kunne studere intona-

sjon og mikrotonalitet. Nøyaktige målinger av starttidspunktene for enkeltnoter i innspilt lyd er viktige i studier av rytmikk. Med digital lydbehandling kan man produsere disse måltallene og mange andre (lydstyrke, støyinnhold, spektralt tyngdepunkt, spektral entropi etc.). Resultatet kan presenteres i tall eller grafikk, og behandles videre.

Med *spektralanalyse* kan man få et godt grafisk inntrykk av klang og dynamikk i kortere eller lengre musikkstykker. Foruten å være en god notasjonsform for elektroakustisk musikk, er spektralanalyser nyttige i analyse av instrumentalmusikk.

Digital lydbehandling i kommersielle anvendelser

Foruten disse anvendelsene i samtidsmusikk og musikologi, står digital lydbehandling helt sentralt i kommersiell musikk. Ved siden av den opplagte bruken innen synthesizerteknologi etc., gjennomsyrer slike teknikker studioproduksjonen av all populærmusikk. Kunnskaper i digital lydbehandling er derfor nyttig for forståelse av dagens populærmusikk.

Kapittel 2

Matematikk

Det er en kjensgjerning at mange komponister og musikere ikke har den aller beste bakgrunn i og interesse for matematikk. Den russiske nevrofysiologen Luria gjorde en gang en stor studie der han så på sammenhengen mellom musikalitet og matematiske evner. Blant de tusener av mennesker som var med i hans materiale, fant han ikke en eneste person som scoret høyere enn gjennomsnittet både på den musikalske og den matematiske testen. Jeg antar imidlertid at han ikke testet f.eks. Xenakis!

Når man arbeider med computermusikk, er det likevel tvingende nødvendig å kjenne endel enkle matematiske prinsipper. Dette blant annet fordi man oftest er nødt til å spesifisere matematiske funksjoner for å styre forløp i lyden. I dette kapitlet skal vi repetere enkle forhold fra skolematematikken, men også se på noen resultater fra mer avansert matematikk som er viktige for oss.

Jeg har valgt ut emner som har direkte relevans for praktisk arbeid med computermusikk. Dette betyr at sentrale felter som differensial- og integralregning er helt utelatt. Nesten alle krav til stringens er dessuten kastet på båten.

2.1 Funksjoner

Den viktigste ideen i matematikken må vel være funksjonsbegrepet. Vi skal se litt på dette, og dessuten presentere en zoo av funksjoner som man trenger i sin verktøykasse når man skal konstruere musikalske forløp.

2.1.1 Hva er en funksjon?

En funksjon er en regel som anvendes på et tall x som vi putter inn i funksjonen, og som leverer et annet tall $f(x)$. Selve regelen kalles *funksjonsforskriften.*

Eksempler:

$$f(x) = 1$$

vil levere tallet 1 uansett hvilken x man putter inn (dette kalles en konstant funksjon).

$$f(x) = x$$

vil sende ut samme tallet som vi sendte inn (dette kalles identitetsfunksjonen).

$$f(x) = x + 1$$

vil addere 1 til tallet vi sendte inn (dette er en lineær funksjon, se under).

Vi kan betrakte en funksjon som en maskin som omformer tall. Når vi arbeider med musikkprogrammet MAX, vil en funksjon i praksis bety en liten boks på skjermen med en inngang og en utgang, hvor vi har skrevet funksjonsforskriften inne i boksen.

Vi skal se på endel spesielle funksjoner under, men først må man ha klart for seg hva en *funksjonsgraf* er. En slik graf er en framstilling av funksjonsverdiene plassert i et *koordinatsystem*. Vi plotter alle "fornuftige" x (de som tilhører funksjonens *definisjonsområde*) og deres tilhørende funksjonsverdier $f(x)$, slik man er kjent med fra alle mulige kurver i avisene. En kurve fra børsen framstiller tiden langs x-aksen, og aksjekursen til ethvert tidspunkt som $f(x)$-verdier.

Med blant funksjonene hører også funksjoner av flere variable, f.eks.

$$f(x, y) = x + y$$

og funksjoner som opererer på andre typer objekter enn vanlige tall (vektorer, matriser, komplekse tall).

2.1.2 Lineære funksjoner

En lineær funksjon er på formen

$$f(x) = ax + b,$$

der a og b er faste tall som vi velger. Eksempel:

$$f(x) = 2x + 3.$$

Her er $a = 2$ og $b = 3$. Det betyr at funksjonen multipliserer x med 2 og legger til 3.

Det finnes endel spesialtilfeller av lineære funksjoner, f.eks. nullfunksjonen $f(x) = 0$ der $a = 0$ og $b = 0$, identitetsfunksjonen $f(x) = x$ der $a = 1$ og $b = 0$, konstante funksjoner $f(x) = b$ og multiplikasjon med konstant $f(x) = ax$.

Lineære funksjoner har fått sitt navn fordi deres graf alltid er en *rett linje*, som vist i figur 2.1. Legg merke til at konstanten a (her 2) styrer hvor *bratt* kurven er, mens konstanten b (her 3) bestemmer verdien $f(0)$.

Lineære funksjoner er på mange måter spesielle, og lette å håndtere. For eksempel er det bare lineære funksjoner og forsinkelsesfunksjoner (se under) som slipper igjennom lyd uten at klangfargen endres.

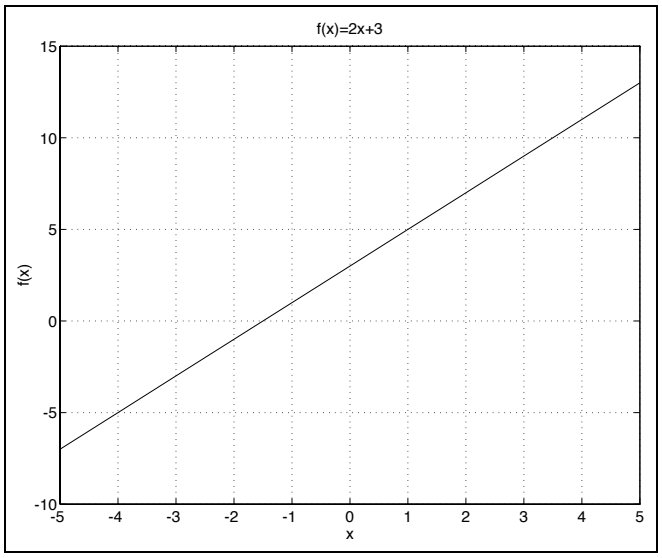
2.1.3 Polynomer

Dersom vi multipliserer et tall x med seg selv ($x \cdot x$) sier vi at vi har opphøyet tallet i andre potens, og vi skriver $f(x) = x^2$.

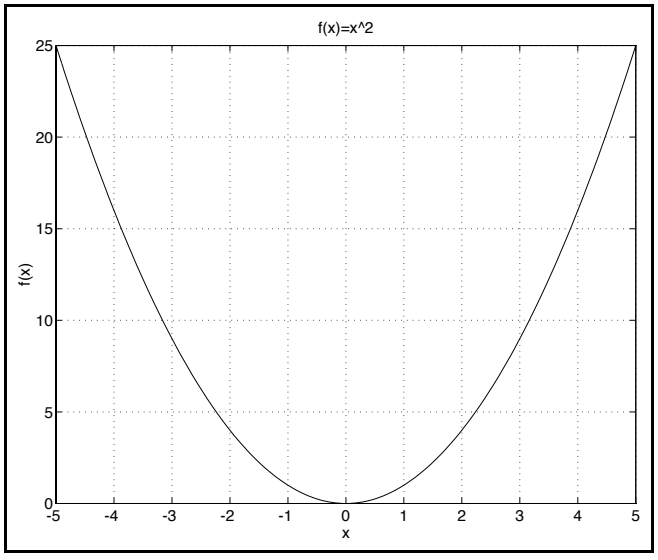
Eksempel: $3^2 = 3 \cdot 3 = 9$.

Grafen til x^2 er en såkalt *parabel*, og er vist i figur 2.2. Her er det at par ting å legge merke til. For det første ser vi at alle negative x-verdier ender opp som positive $f(x)$. For det andre er det tydelig at grafen er *symmetrisk om x=0*; formelt skriver vi $f(x) = f(-x)$ (kan du finne ut av det?). Vi sier også at x^2 er en *like* funksjon.

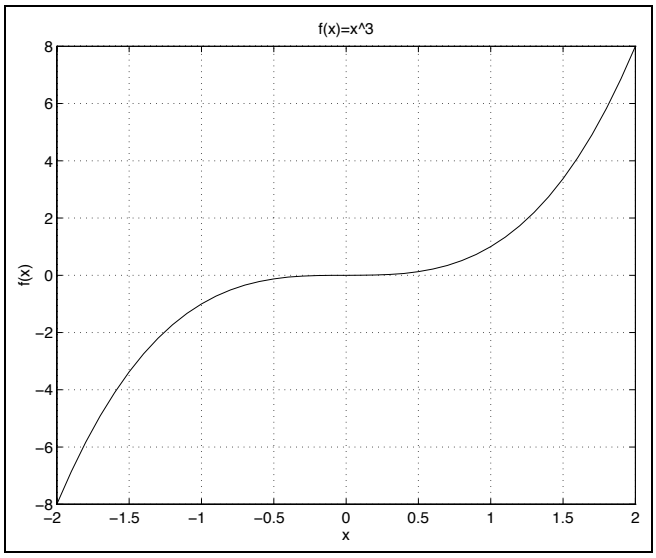
Figur 2.3 viser grafen til x^3.



Figur 2.1: Lineær funksjon



Figur 2.2: Parabel



Figur 2.3: Tredjegradspolynom

x^3 er *antisymmetrisk*, slik at $f(x) = -f(-x)$. Vi sier også at x^3 er en *odde* funksjon. Det gjelder generelt at like tall i potensen gir like funksjoner, og odde tall i potensen odde funksjoner.

Vi skal nå se på funksjoner som er en sum av de typer funksjoner vi har behandlet hittil. For eksempel er $3x^2 + 2x + 1$ en sum av funksjonen x^2 (riktignok med en faktor 3 foran), og den lineære funksjonen $2x + 1$. En slik funksjon kalles et *polynom*. Polynomer har en *grad* etter den høyeste potensen som inngår.

Eksempler:
$5x + 2$ og $3x$ er polynomer av 1. grad.
$4x^2 - 7x + 3$ er et polynom av 2. grad.
$7x^{14} + 9x^8 - x + 64$ er et polynom av 14. grad.

Det finnes mange forskjellige spesielle klasser av polynomer, hvor koeffisientene er konstruert etter bestemte regler. F.eks. er det de såkalte *Bessel-polynomene* som bestemmer styrken på deltonene i spektret ved FM-syntese.

Eksempel 1: Simulering av rørforsterker

En forsterker tar et inngangssignal x og forsterker det med en faktor a, som styres av en volumkontroll. En forsterker kan dermed beskrives ved en såkalt *overføringsfunksjon*. Ideelt sett er forsterkeren *lineær*, det vil si at den har en lineær overføringsfunksjon $f(x) = ax$. La oss anta at $a = 3$ (det spiller ingen rolle), slik at en ideell forsterker har $f(x) = 3x$ som overføringsfunksjon. Lyden går da gjennom forsterkeren uten å endres i det hele tatt, bortsett fra at amplityden tredobles.

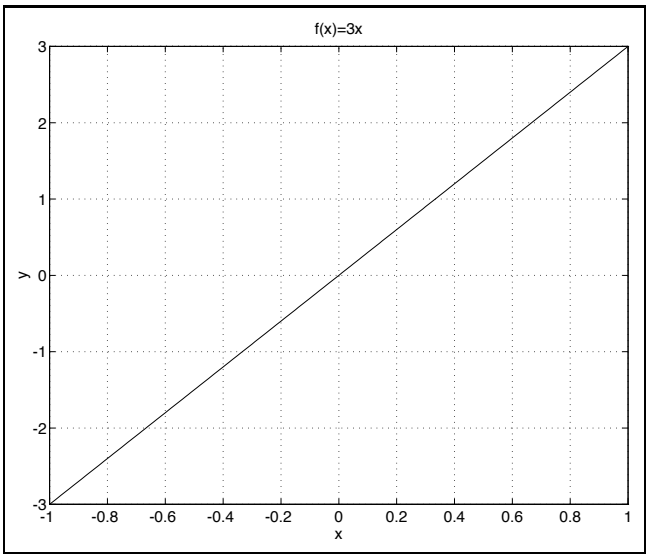
Alle virkelige forsterkere har imidlertid en større eller mindre grad av forvrengning, slik at $f(x)$ avviker fra en lineær funksjon. En typisk effekt er *klipping*, det vil si at når inngangs-

signalet får for stor amplityde går forsterkeren i *metning* og nekter å slippe igjennom mer lyd.

Transistorforsterkere klipper *hardt*, det vil si at signalet er noenlunde intakt opp til et visst nivå, men så kuttes det brått. Den stygge, sprakende forvrengningen er velkjent for de fleste.

Forsterkere med radiorør klipper derimot mere *bløtt*, det vil si at det er en gradvis overgang fra ikke-klipping til klipping. Dette lager en spesiell sound som særlig gitarister er glade i. Det finnes til og med egne effektbokser som etterlikner slik "tube amp sound". Vi skal lage en slik boks på en datamaskin, og da må vi lage oss en overføringsfunksjon som maskinen kan sende lyden igjennom for å oppnå den riktige effekten. Den norske komponisten Anders Vinjar har holdt på med nettopp dette.

La oss forsøke med et polynom. I utgangspunktet ønsker vi at forsterkeren skal være lineær, så vi setter $f(x) = 3x$ (figur 2.4).



Figur 2.4: Overføringsfunksjon for lineær forsterker

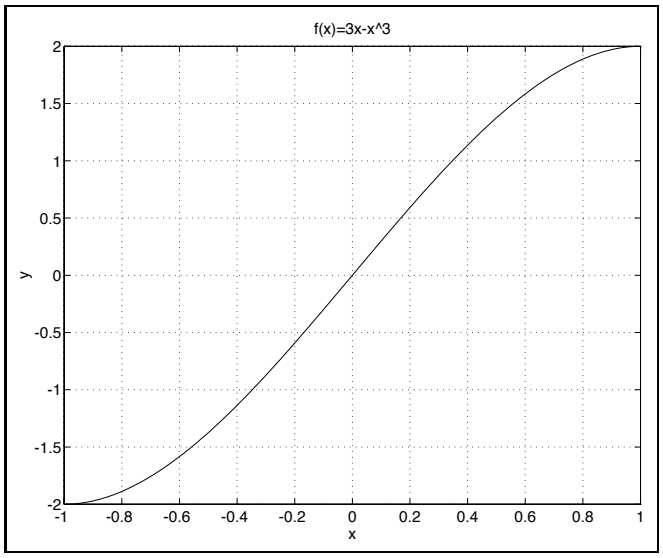
Det ser jo fornuftig ut, men så må vi prøve å introdusere litt "soft clipping". Dette innebærer at grafen må flate ut gradvis i endene. På den måten vil store x (nær pluss og minus 1) bli begrenset.

Vi kan forsøke å trekke fra en x^3, så vi får overføringsfunksjonen $f(x) = 3x - x^3$. Det er ikke så dumt, for som vi ser av grafen for x^3 over ligger den nær null når x er nær null, slik at vår lineæritet ikke ødelegges for små x. For større x blir imidlertid utslagene store, og de vil slå riktig vei dersom vi trekker x^3 fra $3x$ (figur 2.5).

Dette tredjegradspolynomet kan vi sette inn i en MAX-patch eller i et C-Sound-program og oppnå en utmerket "tube amp". Vi har nå sett et eksempel på en lydbehandlingsmetode som kalles *ulineær kurveforming* eller nonlinear waveshaping. I ulineær kurveforming benyttes ofte en spesiell type polynomer som heter *Chebychev-polynomer*. Chebychev-polynomer har pene egenskaper som gjør det lett å forutsi hvordan klangfargen endres.

2.1.4 Hyperbler og røtter

Funksjonen $f(x) = \frac{1}{x}$ kalles en *hyperbel*. Her må vi huske på at definisjonsområdet ikke inneholder null, fordi divisjon med null er forbudt.



Figur 2.5: Overføringsfunksjon for ulineær forsterker

De motsatte funksjonene av potensene over kalles *røtter*. Det tallet som opphøyet i andre potens gir x, heter *kvadratroten* til x, og vi skriver $f(x) = \sqrt{x}$. Kvadratroten til 9 er således 3, kvadratroten til 16 er 4. Igjen må vi passe på definisjonsområdet. Det er forbudt å ta kvadratroten av et negativt tall, for det finnes ingen tall som opphøyet i andre potens gir et tall mindre enn null.

Det tallet som opphøyet i tredje potens gir x, heter *kubikkroten* eller *tredje-roten* av x, og vi skriver $f(x) = \sqrt[3]{x}$. Her er det tillatt å putte inn negative x (finn ut hvorfor!).

Tilsvarende kan vi definere høyere røtter. De fleste programmeringsspråk har imidlertid ikke egne funksjoner for annet enn kvadratrøtter. Det er lett å lure fram høyere røtter ved å benytte seg av måten ikke-heltallige potenser er definert på. Vi har for en n'te rot at

$$\sqrt[n]{x} = x^{\frac{1}{n}},$$

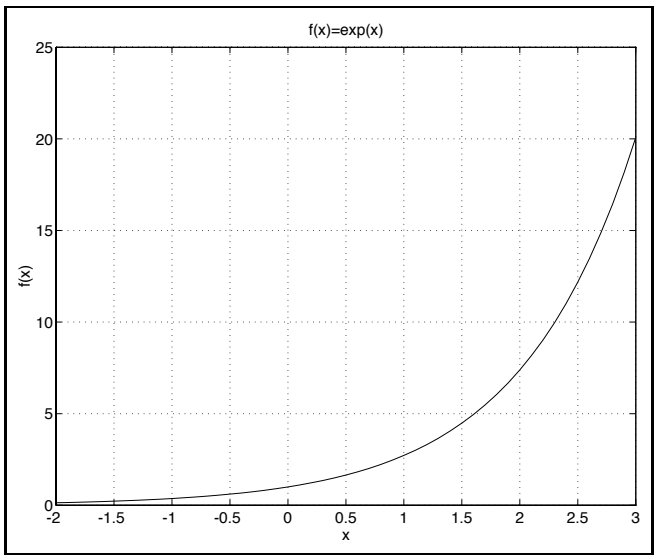
slik at f.eks. fjerderoten av x kan skrives som $x^{0.25}$.

2.1.5 Eksponensialfunksjoner, logaritmer

Hvis vi bruker potenser som beskrevet over, men setter x i *eksponenten* i stedet for i grunntallet, får vi en *eksponensialfunksjon*. Dette er en meget nyttig funksjon og er blant de mest brukte til musikkformål.

Eksempler på slike funksjoner er $f(x) = 2^x$, $f(x) = 3^{4x}$, $f(x) = 3.14^{2x+5}$. I disse eksemplene brukte vi 2, 3 og 3.14 som grunntall (det tallet vi opphøyer i noe). Siden vi alltids kan skru til disse funksjonene som vi vil ved å multiplisere med faktorer etc., er det på en måte likegyldig hvilket grunntall vi bruker. I matematikken viser det seg at alle formler blir mye enklere dersom vi alltid bruker et magisk tall som heter e som grunntall. Vi skal ikke gå mer inn på årsakene til dette, men vi bør være klar over det, bl.a. fordi datamaskiner oftest har en funksjon $exp(x) = e^x$ som vi kan benytte oss av. Tallet e er forøvrig omtrent 2.71, og er like viktig som π for matematikerne.

Funksjonen $f(x) = e^x$ stiger mer og mer, og brattere og brattere. Før eller senere, for store nok x, blir den faktisk større enn alle tenkelige polynomer. Grafen er vist i figur 2.6 (legg merke til at $e^0 = 1$):



Figur 2.6: Eksponensialfunksjonen e^x

Noen regneregler for funksjoner med potenser kan være nyttige:

$$x^{a+b} = x^a x^b,$$

$$x^{-a} = \frac{1}{a},$$

$$x^{a-b} = \frac{x^a}{x^b},$$

$$x^{ab} = x^{a^b}.$$

De omvendte funksjoner av eksponensialfunksjoner heter *logaritmer*. Det finnes forskjellige typer logaritmer, avhengig av hvilket grunntall vi snakker om. Logaritmen til x, med grunntall 10, skriver vi $f(x) = \log_{10} x$, og vi mener da at $f(x)$ er det tallet vi må opphøye 10 i for å få x (det der krever kanskje litt fordøying). Vi har dermed f.eks. at $\log_{10} 100 = 2$ og $\log_{10} 1000 = 3$.

Logaritmen med grunntall e kalles den *naturlige* logaritmen, og skrives $f(x) = \ln(x)$. Vi har dermed at $\ln(e) = 1$. Den logaritmefunksjonen som oftest finnes på datamaskiner, og som gjerne heter *log*, er den naturlige logaritmen.

En viktig regneregel for logaritmer er at $\ln(x^a) = a \ln(x)$. Siden $\ln(x)$ er den motsatte (inverse) funksjonen av e^x, har vi også at $e^{\ln(x)} = x$; hvis vi tar e og opphøyer i den naturlige logaritmen til x, får vi simpelthen x tilbake. Disse to regnereglene kan vi bruke til å lage oss potenser dersom programmeringsspråket vårt ikke har dette tilgjengelig (dette er faktisk svært vanlig):

$$x^a = e^{\ln(x^a)} = e^{a \ln(x)}.$$

Eksempel 2: Beregning av frekvens fra notenummer

Et av de vanligste spørsmålene som dukker opp i denne bransjen er følgende: Vi har et MIDI notenummer (60 for C, 61 for Ciss etc.). Hvilken frekvens i Hz tilsvarer dette?

Vi skal ikke gå igjennom teorien om skalaer her, men anta at vi vil bruke "equal temperment". Vi går ut fra at kammertonen A har frekvensen 440 Hz. Nå er det slik at vi vil få neste halvtone ved å multiplisere frekvensen med en konstant k. Når vi har multiplisert 12 ganger skal vi ha gått opp en oktav, dvs. at frekvensen skal være doblet. For å oppnå dette setter vi $k = \sqrt[12]{2} = 1.0594631$. Hvis MIDI-noten til kammertonen er 69, får vi frekvensen

$$f = 440 \cdot 1.0594631^{n-69}.$$

for MIDI-note n. Dette er en eksponensialfunksjon, og dette viser at øret ikke oppfatter frekvens lineært. Bemerk også at n ikke behøver å være et helt tall, så mikrotonalitet håndteres problemfritt med denne formelen.
En datamaskin vil kanskje ha formelen på denne måten:

```
f=440*pow(1.0594631,(n-69));
```

eller i verste fall slik:

```
f=440*exp((n-69)*log(1.0594631));
```

Eksempel 3: Beregning av notenummer fra frekvens

Det å finne et MIDI-notenummer fra en frekvens i Hz er også ofte nyttig. Da tar vi simpelthen formelen fra forrige eksempel og løser likningen med hensyn på n. Dette gjøres ved å ta den naturlige logaritmen på hver side av likhetstegnet, og deretter anvende regnereglene som er gitt over:

$$n = \frac{\ln(f) - \ln(440)}{\ln(1.0594631)} + 69.$$

Dette er en logaritme-funksjon. På en datamaskin skriver man noe slikt:

```
n=(log(f)-log(440))/log(1.0594631)+69;
```

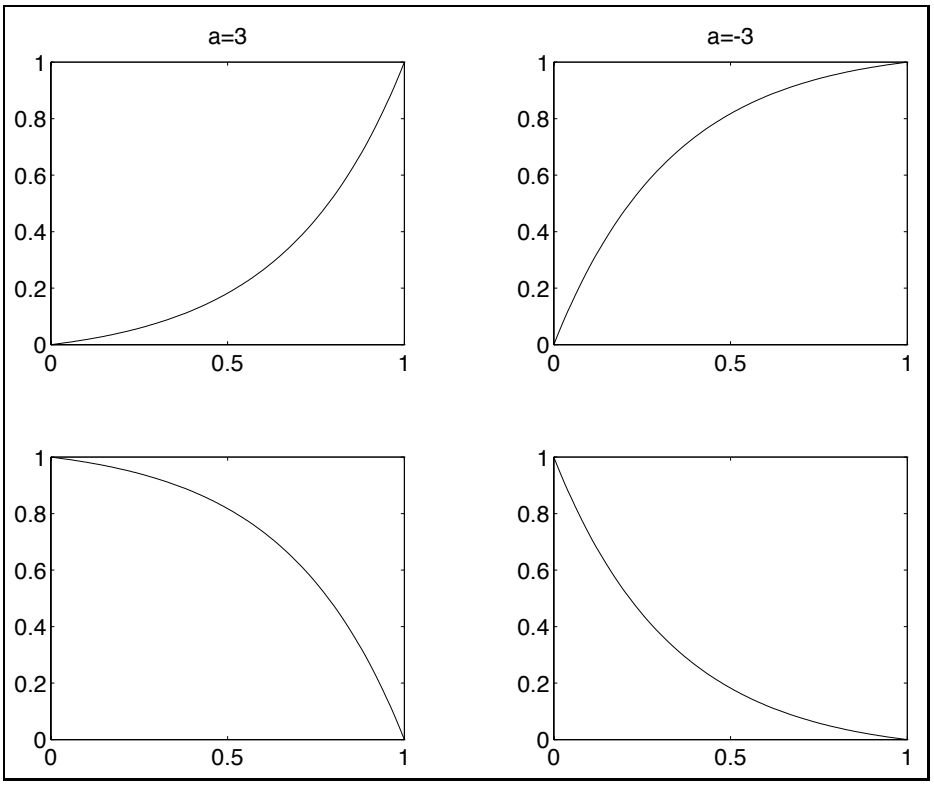
Eksempel 4: Eksponensielle kontrollfunksjoner

Som vi har sett, så er eksponensialfunksjoner fornuftige å bruke dersom vi vil at øret skal oppfatte en jevnt glidende tonehøyde. Amplityder bør også styres med slike funksjoner, fordi øret oppfatter lydstyrke på en liknende måte. Eksponensialfunksjoner er dermed mye brukt i omhyllingskurver (envelopes) og andre kontrollfunksjoner. Spesielt velkjent er envelopes av ADSR-typen (Attack-Decay-Sustain-Release). Slike envelopes består av tre eksponensialfunksjoner som er satt sammen, samt en konstant Sustain-fase.

I C-Sound og mange andre systemer finnes en funksjonsgenerator som lar brukeren spesifisere start- og endepunkter for slike eksponensialfunksjoner. Hvis vi programmerer f.eks. i C, kan vi bruke noe slikt:

$$f(x) = startverdi + (sluttverdi - startverdi)\frac{1 - e^{ax}}{1 - e^{a}}.$$

Hvis vi lar x løpe fra 0 til 1, vil $f(x)$ løpe fra *startverdi* til *sluttverdi*, med en eksponensiell form som bestemmes av a (positiv eller negativ). Figur 2.7 viser noen eksempler.



Figur 2.7: Eksponensielle kontrollfunksjoner

2.2 Trigonometri

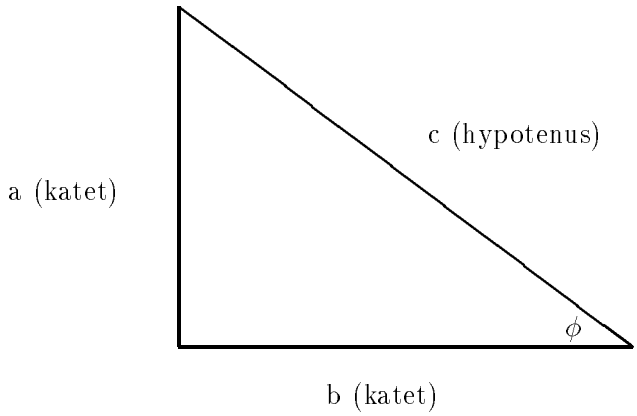
Trigonometri betyr trekantmåling. Når vi studerer forholdene mellom sidene i rettvinklede trekanter (trekanter hvor ett hjørne er rettvinklet), dukker det opp såkalte trigonometriske funksjoner: *Sinus*, *cosinus*, *tangens* etc. Disse funksjonene er morsomme i seg selv, men det underlige er at de samme funksjonene dukker opp overalt i fysikken. Trigonometriske funksjoner brukes til å beskrive f.eks. en pendels bevegelser og bølger på havet. Trigonometriske funksjoner er sentrale også i forbindelse med akustikk og vibrasjon i musikkinstrumenter, og er derfor av spesiell interesse for oss. Summetonen i telefonen er en temmelig ren sinusfunksjon.

2.2.1 Rettvinklede trekanter

En rettvinklet trekant består av de to sidene som ligger inntil den rette vinkelen (de to *katetene*) og den siste, "skrå" siden (*hypotenusen*). Vi kaller lengden til katetene for a og b, og lengden til hypotenusen for c. En av vinklene kaller vi ϕ. Det der var en gresk bokstav som heter *phi*. Den ser flott ut på papiret, og er "standardbokstaven" for vinkler.

Forholdene mellom a, b og c er gitt ved *Pythagoras' teorem*:

$$c^2 = a^2 + b^2,$$



Figur 2.8: Rettvinklet trekant

som vi uttaler "kvadratet på hypotenusen er summen av kvadratene på katetene".

Eksempel 5: Avstanden til en lydkilde

Anta at en lytter er plassert i origo i et koordinatsystem, altså i punktet (0,0). Vi ønsker å simulere at en lydkilde beveger seg i planet på en eller annen måte, og vi har allerede beregnet (x,y)-koordinatene til lydkilden.

Som en del av simuleringen må vi kontrollere tidforsinkelsen fra lyden sendes ut til den mottas av lytteren. Dette vil gi en fin Doppler-effekt når lydkilden beveger seg. Denne tidsforsinkelsen avhenger direkte av avstanden til lydkilden. Vi må også kunne kontrollere amplityden, som faller som en direkte hyperbelfunksjon av avstanden (lydstyrken vil da falle med kvadratet av avstanden).

Alt dette koker dermed ned til at vi må finne avstanden s fra punktet (0,0) til punktet (x,y). Ved å tegne opp situasjonen i et koordinatsystem, ser vi at vi får en rettvinklet trekant, og fra Pythagoras har vi

$$s = \sqrt{x^2 + y^2}.$$

2.2.2 De trigonometriske funksjonene

Du er sikkert vant til at vinkler måles i *grader*. Når vi arbeider med matematikk og datamaskiner måler vi derimot nesten alltid vinkler i *radianer*.

180 grader tilsvarer π radianer, slik at 1 radian er $180/\pi$, ca 57 grader (husk at $\pi = 3.141592654...$). Eksempelvis er da 30 grader $\pi/6$ radianer, 45 grader $\pi/4$ radianer, 90 grader $\pi/2$ radianer. Vi kutter som oftest ut enheten radianer, og sier f.eks. at vinkelen $\phi = 1.28$.

La oss gå tilbake til tegningen av den rettvinklede trekanten. Vi *definerer* nå en funksjon

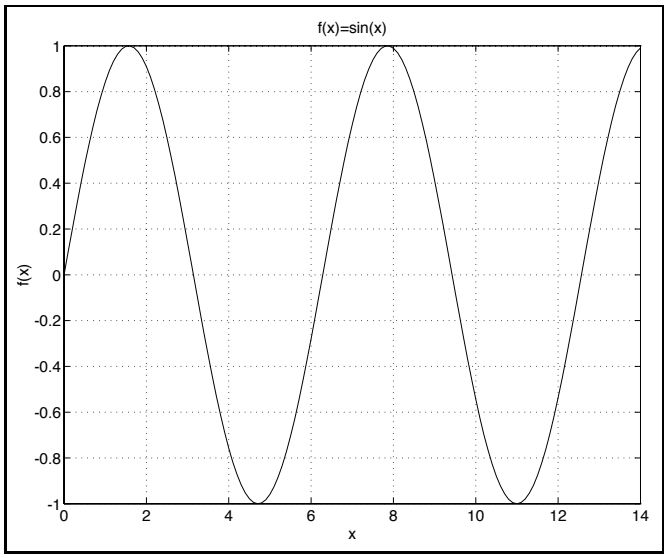
$f(x) = \sin(\phi)$ på denne måten:

$$\sin(\phi) = \frac{a}{c},$$

som vi uttaler "sinus til en vinkel er forholdet mellom den motstående katet og hypotenusen". Ettersom ϕ varierer, må lengdene til sidene variere for at vi fortsatt skal ha en rettvinklet trekant. Forholdet mellom motstående katet og hypotenusen vil alltid følge den samme funksjonen $\sin(\phi)$, uansett hvor stor trekanten er.

I trekanten over gir det ikke mening å snakke om ϕ større enn $\pi/2$. Vi utvider derfor definisjonsområdet til å omfatte alle mulige tall, positive og negative, og definerer hvordan $\sin(x)$ skal variere her (geometrisk gjøres dette ved å studere den såkalte enhetssirkelen, men det hopper vi over nå).

Sinusfunksjonen er vist i figur 2.9. Det viktigste å legge merke til her er at sinusfunksjonen er



Figur 2.9: Sinusfunksjonen

periodisk, med periode $2\pi \approx 6.28$. Dette betyr at funksjonen gjentar seg bortover i det uendelige. Hvis vi fester en blyant på en svingende pendel, og lar blyanten tegne på et langt papir som vi trekker forbi pendelen, så vil vi få en graf som denne. En pendel beskrives altså ved en sinusfunksjon, og i naturen er det svært mange prosesser som oppfører seg som pendler.

Vi kan styre *frekvensen*, dvs. hvor mange ganger sinuskurven gjentar seg hvert sekund, ved å lage oss funksjoner av typen

$$F(t) = \sin(2\pi f t),$$

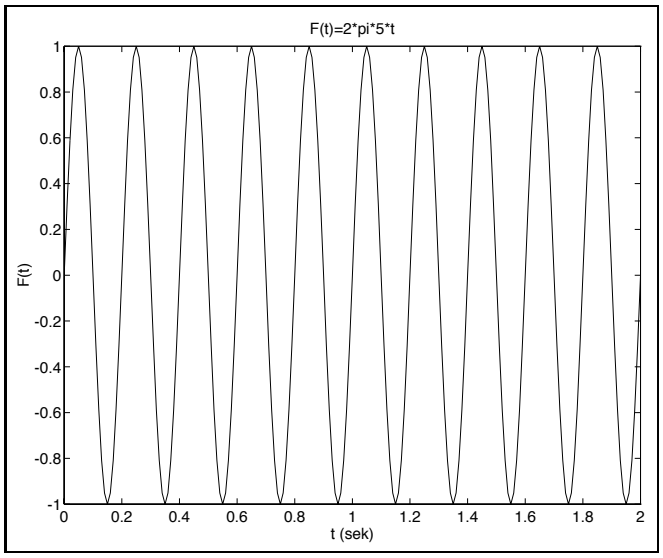
der f er frekvensen i svingninger pr. sekund og t er tiden i sekunder. Enheten "svingninger pr. sekund" har fått et eget navn, *Hertz*, forkortet Hz. Hvis frekvensen f.eks. er 5 Hz, ser funksjonen ut som vist i figur 2.10.

Tilbake til den rettvinklede trekanten igjen. Vi definerer nå en ny funksjon $f(x) = \cos(\phi)$ på denne måten:

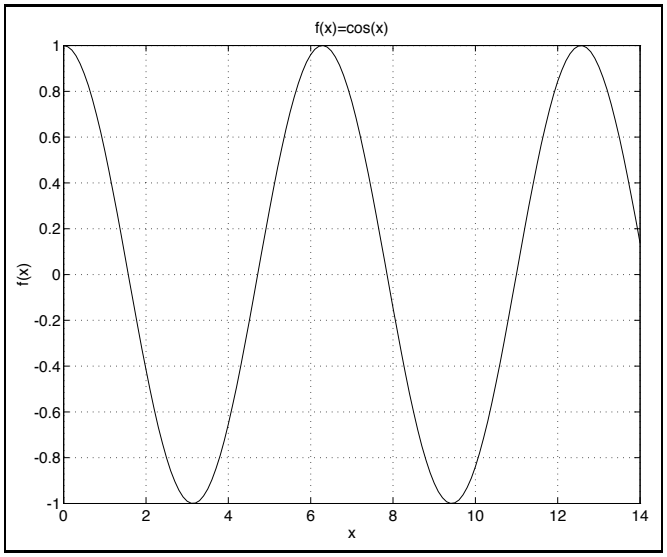
$$\cos(\phi) = \frac{b}{c},$$

som vi uttaler "cosinus til en vinkel er forholdet mellom den hosliggende katet og hypotenusen".

Vi utvider igjen definisjonsområdet, og får funksjonen vist i figur 2.11.



Figur 2.10: Sinusoid funksjon



Figur 2.11: Cosinusfunksjonen

Denne grafen er identisk med grafen for sinus, bortsett fra at den er forskjøvet et stykke i horisontal retning. Vi sier at sinus og cosinus, uansett frekvens, begge er *sinusoider*, men at de har forskjellig *fase*. Hvis vi spiller en sinusoid ut gjennom en høyttaler, vil vi høre en ren, myk tone. Tonehøyden vil avhenge av frekvensen, men fasen vil ikke ha noen betydning for hvordan lyden oppfattes.

Det må bemerkes at man ofte bruker ordet "sinus" om alle sinusoider, uavhengig av frekvens og fase. Dette er ikke helt korrekt, fordi sinus er en strengt definert funksjon med en bestemt periode og fase.

2.2.3 Fourier-rekker

Vi har nå definert sinus og cosinus, og vi har sett at begge er bølgeformede, periodiske funksjoner.

Enhver periodisk funksjon kan konstrueres ved å summere et antall sinus- og cosinusfunksjoner, hver med en frekvens som er et heltallsmultiplum av frekvensen til den periodiske funksjonen.

Dette er *Fourier's teorem*, og en sum av sinuser og cosinuser som bygger opp en funksjon heter en *Fourier-rekke*. Fourier's teorem er et av de dypeste og vakreste resultater i matematikken.

Dette er veldig viktig når vi driver med lyd, for øret oppfatter som hovedregel en lyd i *frekvensdomenet*, dvs. som de sinusoidene som den er bygget opp av. Dette er nærmere forklart i heftet om akustikk.

Bemerk at vi *enten* kan spesifisere en Fourier-rekke som en sum av ekte sinus- og cosinusfunksjoner, *eller* som en sum av sinusoider der fasen er oppgitt. Dette er likegyldige betraktningsmåter, fordi vi kan lage en sinusoid med hvilken som helst fase ved å addere en sinus og en cosinus med de riktige amplityder.

Dersom funksjonen vi ønsker å konstruere har sprang (diskontinuiteter), må vi i prinsippet addere et uendelig antall sinusoider. Dette er imidlertid ikke av interesse for oss. Alle digitale lyder er nemlig *båndbegrensede*, og inneholder bare frekvenser opp til halve samplingfrekvensen.

Fourier's teorem omhandler periodiske funksjoner. Men hva så med ikke-periodiske funksjoner? De kan også håndteres, på en av to måter. Enten sier vi at en funksjon bare er definert i et avgrenset område (en lyd varer f.eks. bare en viss tid). Vi utvider så funksjonen periodisk, dvs. at vi *later som om* funksjonen gjentar seg periodisk utenfor det opprinnelige definisjonsområdet. Funksjonen er så klar for Fourier's teorem, og vi kan etterpå glemme funksjonsverdiene utenfor det opprinnelige området.

Den andre angrepsmåten er å si at funksjonen faktisk *er* periodisk, men med uendelig lang periode. Grunnfrekvensen blir da uendelig liten, og de øvrige sinusoidene kommer uendelig nær hverandre. Fourier-rekka går da over til et *Fourier-integral*.

Disse tingene blir forhåpentlig mer intuitivt forståelige når vi ser på anvendelsene innen akustikken.

Kapittel 3

Akustikk

Hvis man skal ha håp om å kunne produsere lyd med en datamaskin, er det en forutsetning at man har en grunnleggende forståelse for hva lyd er, hvordan den oppstår og hvordan den påvirkes av fysiske hindringer. Læren om disse forholdene heter *akustikk*. Innenfor dette feltet har vi også andre spesielle fagområder som er av interesse for oss: Musikkakustikk, romakustikk, fonetikk, elektroakustikk etc. Det er dessuten nødvendig å vite hvordan forskjellige aspekter ved lyden oppfattes av mennesket (psykoakustikk).

Dette kapittelet vil gi en introduksjon til disse tingene.

3.1 Hva er lyd?

Lyd er hurtige variasjoner i lufttrykk, som brer seg som bølger med en hastighet på 343 meter i sekundet (1235 km/t) ved romtemperatur. Fra en punktformet lydkilde i rommet brer lyden seg utover i alle retninger, slik at bølgefronten består av et stadig større kuleskall. Arealet av et slikt kuleskall øker med kvadratet av radien i kula. [1] Dette betyr at lyden får en stadig større luftflate å bevege, og siden det ikke tilføres noe energi må det innebære at lydstyrken svekkes tilsvarende. Derfor vil lydstyrken falle med kvadratet av avstanden.

Som vi ser er luft (eller en annen gass) nødvendig for at lyd skal kunne forplante seg. Det nytter lite å rope til hverandre på månen, og drønnene vi kan høre når romskip eksploderer i dårlige science-fiction-filmer er lite realistiske, selv om sjokkbølger fra ekspanderende gasser kanskje kan gi et svakt plopp.

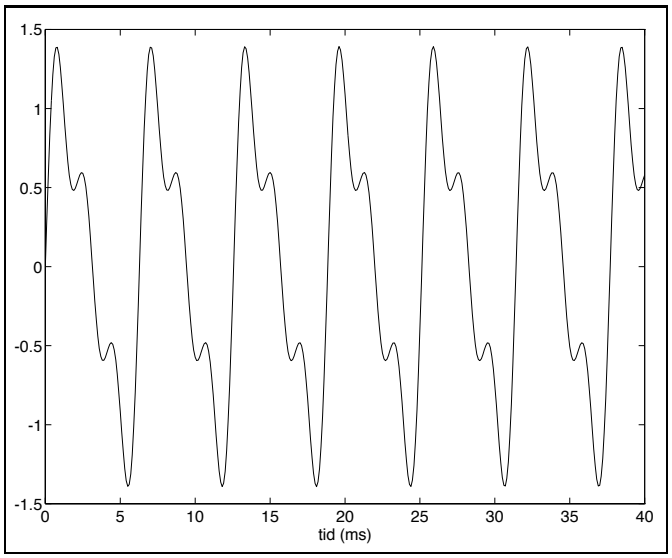
Vanlige overflatebølger på vann er *transversale*, det vil si at bølgebevegelsen (opp og ned) går på tvers av utbredelsesretningen (bortover vannflaten). Lydbølger er derimot *longitudinale*, hvilket betyr at bølgevegelsen (fram og tilbake) går langs samme retning som utbredelsesretningen.

Vanligvis tegner vi likevel lydbølgene som om de var transversale. Vi får da en graf som kalles et *oscillogram*, der tiden går langs x-aksen og lufttrykket langs y-aksen. Grafen er normalisert slik at det "normale" lufttrykket kalles null. Oscillogrammer kan vises på en billedskjerm ved hjelp av et *oscilloskop*.

[1] $A = 4\pi r^2$

3.2 De tre lyd-parametrene

Ved å betrakte et oscillogram kan vi si endel om lydens egenskaper. Dersom vi har en statisk lyd, dvs. at den ikke endrer seg over måletidsrommet, kan oscillogrammet se ut f.eks. som i figur 3.1.



Figur 3.1: Statisk lyd

Tre grunnleggende parametre bestemmer hvordan en lyd oppfattes av øret: Amplityde, frekvens og klangfarge.

3.2.1 Amplityde, lydstyrke

Amplityden er graden av utslag, og leses av som maksimalverdier i oscillogrammet. Amplityden til lyden over er ca. 1.4. Lydstyrken vi vil oppfatte avhenger av amplityden: Større amplityde gir kraftigere lyd. Nå har det seg imidlertid slik at øret oppfatter lydstyrke logaritmisk, slik at sammenhengen mellom lydstyrke og amplityde ikke er lineær. Vi bruker derfor gjerne enheten *desi-Bel* (etter Alexander Graham Bell) som mål for amplityde. Når det gjelder akustisk lyd, går vi fram på følgende måte. Først bestemmer vi oss for at den svakeste lyden vi kan høre skal kalles 0 dB (dette er definert som $10^{-16} W/cm^2$ for en sinustone på 2 kHz). Amplityden til denne lyden kan vi kalle a_{ref}, fordi den brukes som en referanse som vi sammenlikner andre amplityder med. En lyd med amplityde a har nå følgende styrke målt i dB [2]:

$$A = 20 \cdot \log_{10} \frac{a}{a_{ref}}.$$

Når vi benytter dB-skalaen, kan vi si endel kvantitative ting om hvordan øret oppfatter lydstyrke. Den minste endring i lydstyrke som vi kan merke er ca. 1 dB. Den kraftigste lyd vi kan høre uten at vi får vondt er på ca. 120 dB. Vi sier at ørets *dynamiske område* er 120 dB. Dette tilsvarer et amplityde-område på en million til en! Det er svært vanskelig å lage

[2] Faktoren 20 ser litt tilfeldig ut. dB-skalaen er egentlig tenkt brukt i forbindelse med *effekter*, ikke amplityder. Effekten er proporsjonal med kvadratet av amplityden. Faktoren blir derfor 10 hvis vi snakker om effekter. Hvis vi i tillegg snakker om Bel istedenfor desi-Bel, blir faktoren 1, i analogi med f.eks. liter og desiliter.

elektronikk som kan håndtere et så stort spenningsområde. Profesjonelt lydutstyr har oftest et dynamisk område på under 100 dB.

En viktig konsekvens for oss data-musikere er at vi ikke kan øke amplityden lineært og så regne med at vi får et naturlig crescendo. Vi må istedet bruke en eller annen eksponensiell funksjon.

3.2.2 Frekvens, tonehøyde

I oscillogrammet over kan vi se en periodisitet; den samme kurveformen gjentar seg igjen og igjen. Slike lyder oppfattes som en kontinuerlig og ren tone. Antall gjentakelser pr. sekund gir frekvensen til den grunntonen vi oppfatter, målt i Hertz. Har vi en grunntone på f.eks. 110 Hz, betyr dette at den periodiske kurveformen gjentar seg selv 110 ganger i sekundet. Dersom vi dobler til 220 Hz, hører vi dette som et sprang en oktav opp. Videre dobling til 440 Hz gir enda en oktav. Dette innebærer at heller ikke sammenhengen mellom frekvens og oppfattet tonehøyde er lineær. Jo høyere vi går opp i registret, jo lengre avstand er det mellom halvtonetrinnene, målt i Hz. For å gå opp et halvtonetrinn, kan vi ikke legge til en fast frekvens. Vi må istedet multiplisere frekvensen med en faktor. Når vi har 12 halvtonetrinn i en oktav er denne faktoren 12-roten av 2, eller 1.05946. Musikalske skalaer vil bli omtalt senere.

Vi har sett at både amplityde og frekvens oppfattes logaritmisk av øret. Det viser seg at de fleste sanser virker på samme måte: Multipliserer vi et stimulus med en faktor, oppfattes dette som et fast tillegg. Dette gjelder vår oppfattelse av lys, varme, smak, lukt, tid osv. At sammenhengen mellom stimuli og persepsjon er logaritmisk, (*Fechners lov*) ble først påstått av G. T. Fechner i boka *Elements of Psychophysics* i 1860.

Hvis kurveformen *ikke* er periodisk, kan dette bety to ting. Enten er kurven en sum av to eller flere periodiske kurver med frekvenser som ikke står i enkle forhold til hverandre. Kurven kalles da *kvasiperiodisk*, og vil oppfattes som en dissonant akkord. Dette vil bli omtalt senere. Den andre muligheten er at kurven er "bare rot", og ikke er en enkel sum av et lite antall periodiske kurver. Dette vil oppfattes som en susing eller støylyd, ofte uten noen bestemt tonehøyde (unpitched).

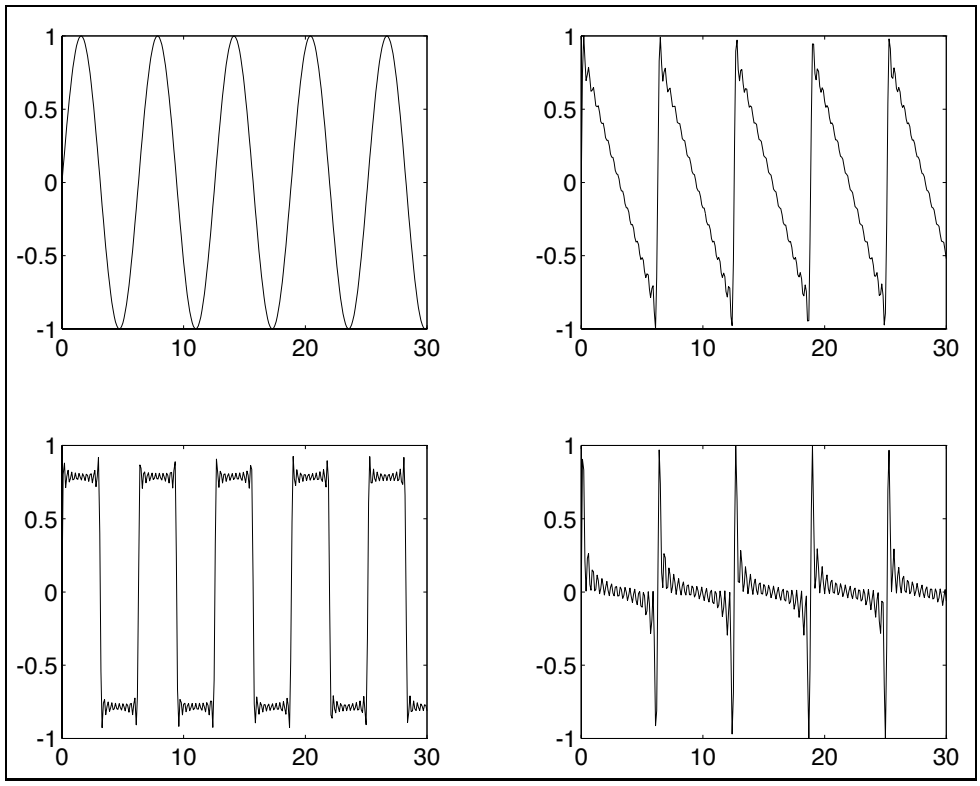
3.2.3 Klangfarge

Den tredje parameteren, klangfargen, er en svært kompleks ting, og bestemmes av lydkurvens *form*.

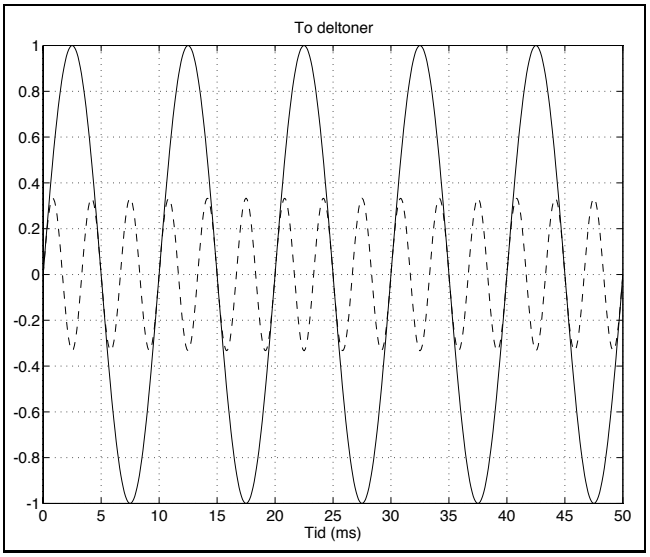
Figur 3.2 viser fire ulike periodiske og statiske lyder. De har samme frekvens og amplityde, og vil alle oppfattes som kontinuerlige toner med samme tonehøyde og omtrent samme lydstyrke. Men kurvene har ulik form. Vi ser en sinustone, en sagtanntone, en firkanttone og et pulstog. Dette gir ulik klangfarge for de forskjellige lydene.

Matematikeren Fourier viste at enhver repeterende kurveform kan genereres ved å addere (mikse) et antall sinusoide deltoner, hver med en frekvens som er et heltallsmultiplum av grunnfrekvensen. Figur 3.3 viser prinsippet. Her adderes en sinusformet grunntone med frekvens 100 Hz til en overtone med frekvensen 300 Hz, dvs. en oktav og en kvint over. Resultatet vises i figur 3.4.

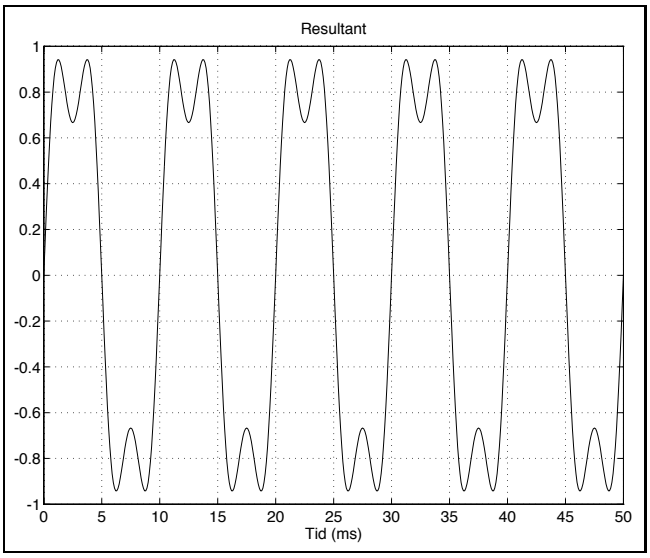
De forskjellige deltonene i en ren tone kalles ofte de *harmoniske*. Grunntonen heter da den første harmoniske, første overtone på to ganger grunnfrekvensen heter andre harmoniske, andre overtone på tre ganger grunnfrekvensen heter tredje harmoniske etc. Som man ser er terminologien litt forvirrende.



Figur 3.2: Forskjellige kurveformer



Figur 3.3: Grunntone og andre overtone



Figur 3.4: Grunntone og andre overtone mikset sammen

I figur 3.3 begynte begge deltonene på samme sted i sinuskurven, dvs. at deltonene har samme fase. Hvis deltonene har ulike faser, vil den resulterende kurveformen naturligvis se annerledes ut. Men øret oppfatter i utgangspunktet ikke faseinformasjon, det oppfatter bare amplityden til de forskjellige deltonene. Forholdet mellom disse amplitydene bestemmer klangfargen. Vi kan med andre ord ha to lyder som ser radikalt ulike ut på et oscilloskop, fordi de har ulik faseinformasjon, men som låter nøyaktig likt, fordi deltonenes amplityder er de samme.

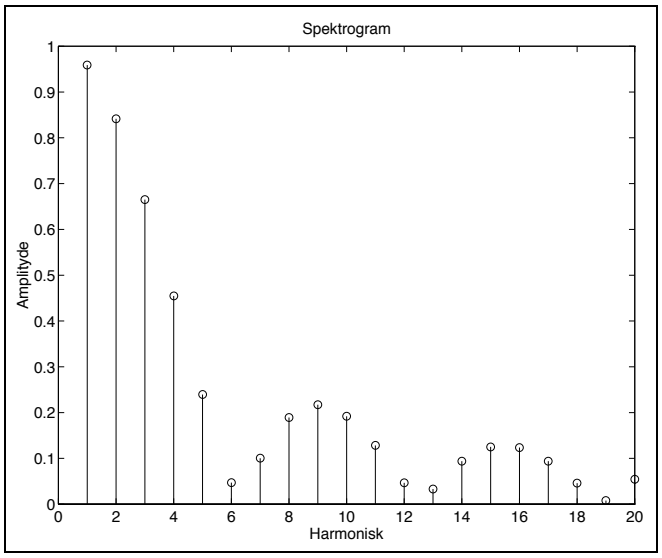
Dersom vi er interessert i klangfarge, er det derfor mest praktisk å beskrive lyden i et stolpediagram der deltonenes amplityder er vist som funksjon av frekvensen. En slik figur kalles et spektrogram, og kan avbildes ved hjelp av et måleinstrument som kalles en spektralanalysator eller *spektrograf*. Et spektrogram kan f.eks. se ut som i figur 3.5. Overtonene er gjerne svakere enn grunntonen. Legg også merke til de to "toppene" rundt den 9. og den 15. harmoniske. Slike topper er typiske i naturlig lyd, og kalles *resonanser* (fonetikerne bruker begrepet *formanter*).

Dette er altså en enkel tone med en veldefinert tonehøyde. Kurveformen er da som vi har sett periodisk, og spektrogrammet består av adskilte stolper med fast avstand lik grunntonens frekvens. Vi sier at spekteret er *diskret*.

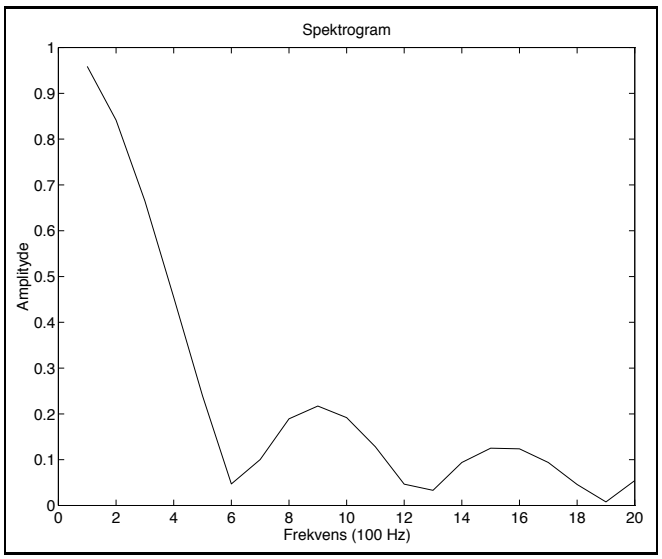
Hvis lyden ikke er periodisk, og dermed uten tonehøyde, vil vi kunne få et *kontinuerlig* spektrum der alle mulige frekvenser er tilstede. Dette er typisk for støylyder. Spektrogrammet kan da se ut f.eks. som i figur 3.6 (oftest er det langt mer "hårete").

Dersom en lyd vises som et oscillogram, sier vi at lyden er representert i *tidsdomenet*. En lyd vist som et spektrogram er representert i *frekvensdomenet*. Det er viktig å forstå at disse to representasjonsformene inneholder den samme informasjonen, dersom vi ser bort fra faseforhold. [3] Omregningen mellom tidsdomenet og frekvensdomenet foretas ved hjelp av en spesiell regnemetode som kalles *Fouriertransformasjon*. Fouriertransformasjon er sentral i digital lydbehandling. Omregningen fra tidsdomenet til frekvensdomenet kan vi også kalle *spektralanalyse*. Omregningen fra frekvensdomenet til tidsdomenet kan vi kalle *additiv syntese*, fordi vi adderer sammen alle de sinusoide deltonene i spekteret for å lage en lyd i tidsdomenet.

[3] Det er imidlertid ikke uvanlig at spektrografer også kan vise "fasespekteret"



Figur 3.5: Spektrogram



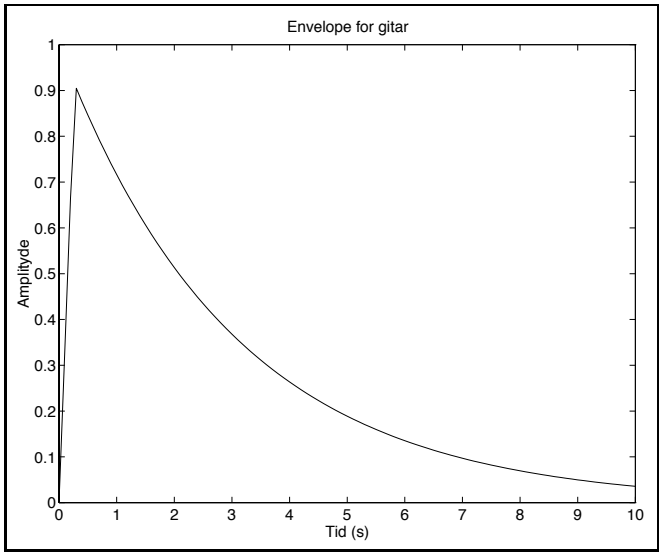
Figur 3.6: Kontinuerlig spektrum

3.2.4 Dynamikk

Hittil har vi betraktet lyd som ikke endrer seg over tid. Dette gir kontinuerlige toner. Skal vi lage musikk, må selvsagt de tre parametrene i lyden forandre seg hele tiden.

Dynamikk i amplityden

Dynamikk i amplityden er det første vi må få til. Alle musikkinstrumenter fungerer slik at amplityden forandrer seg i løpet av hver tone. En note på en gitar f.eks. starter med en svært hurtig øking av amplityden (attack) fulgt av en langsom utdøing (decay). Dette kan vises i en graf som figur 3.7.



Figur 3.7: Gitar-envelope

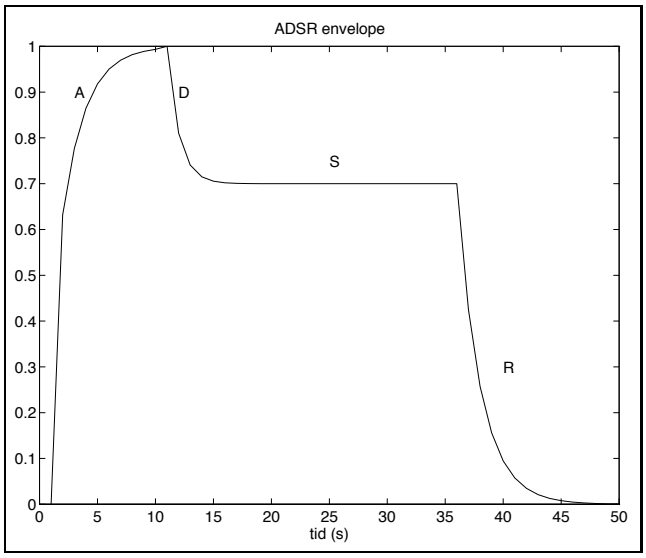
En slik graf kalles en *envelope*. [4] Når vi skal syntetisere lyd er det selvsagt ønskelig å kunne tegne enhver envelope, og dette er mulig med datamaskiner. Mange synthesizere begrenser seg imidlertid til en generalisert envelope-type med fire linjesegmenter. Dette heter *ADSR*, for Attack-Decay-Sustain-Release (figur 3.8). Knekkpunktene mellom segmentene kan plasseres fritt av brukeren. Bemerk at linjene ikke er rette, dette kommer av den ulineære sammenhengen mellom amplityde og oppfattet lydstyrke. Attack-biten er litt rar, og gir ikke et naturlig crescendo. Det viser seg imidlertid at naturlige instrumenter ofte oppfører seg på en liknende måte.

Vi kan også la amplityden kontrolleres delvis av en lavfrekvent sinusoid. Dette lager en effekt som heter *tremolo*.

Dynamikk i frekvens

Dynamikk i frekvens er selvsagt nødvendig for å få fram forskjellige noter, men det er også

[4] Det finnes et godt norsk ord, nemlig *omhyllingskurve*. Dette er imidlertid ikke mye brukt i musikkbransjen.



Figur 3.8: ADSR-envelope

vanlig å variere frekvensen kontinuerlig, enten fra en note til en annen (glissando), eller ved hjelp av en lavfrekvent sinusoid (vibrato).

Dynamikk i klangfargen

For at lyden ikke skal bli helt død, må klangen endres hele tiden. Denne kompliserte oppgaven forsøkes løst på ulike måter ved hjelp av forskjellige syntesemetoder.

Spektrogrammene vi har sett på viser klangen bare på ett bestemt tidspunkt. Når klangen varierer over tid, må vi bruke diagrammer som forsøker å vise lyden i tre dimensjoner (vi får en tidsakse i tillegg til frekvens- og amplitydeaksene). Dette kan gjøres i et "landskapsdiagram" der frekvensen går mot høyre og tiden går innover i bildet (figur 3.9).

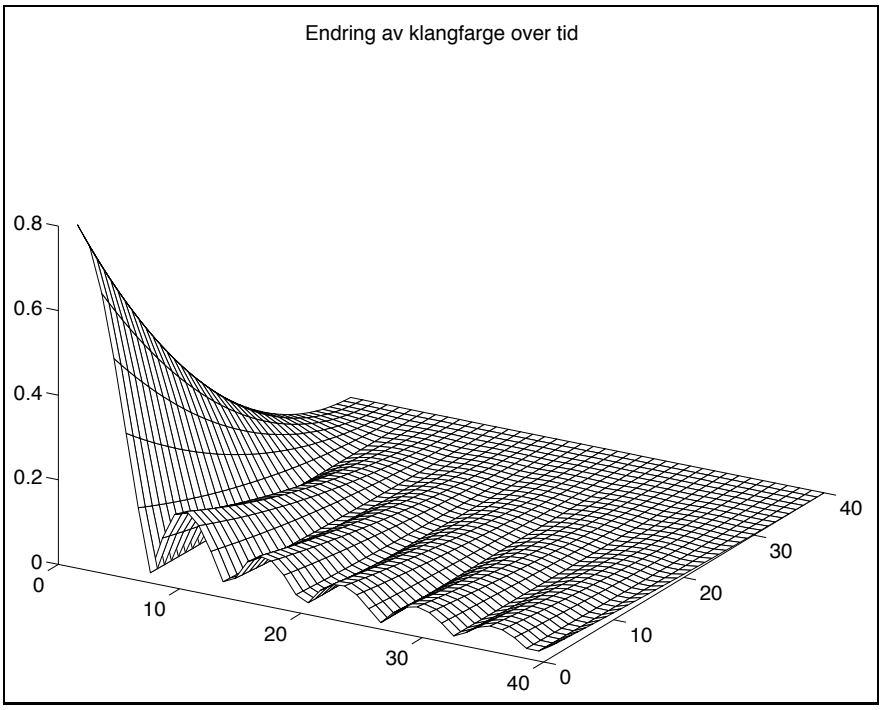
Slike diagrammer kan være vanskelige å lese. En representasjonsform som ofte er mer praktisk er *sonogrammet*, der tiden løper langs x-aksen, frekvensen langs y-aksen og amplityden vises som gråtoner (figur 3.10).

3.3 Beats, kritiske bånd

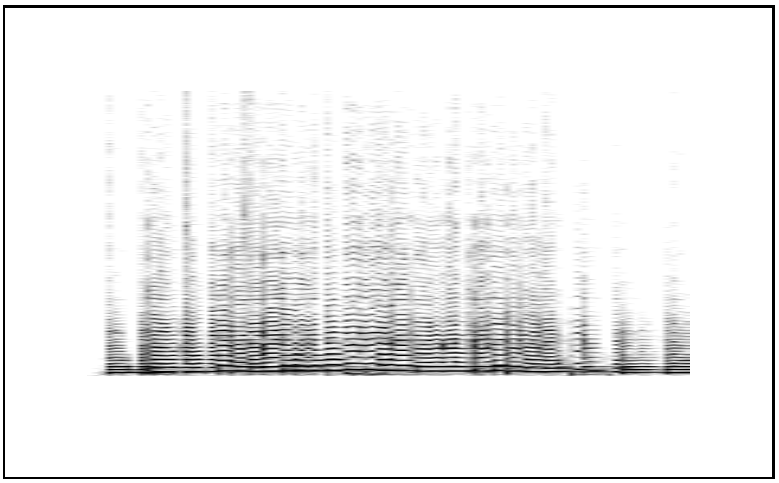
I dette avsnittet skal vi se på et psykoakustisk fenomen som kalles *beats*. Dette er interessant i seg selv, men først og fremst er det meningen at dette stoffet skal gi en bedre forståelse for dualiteten mellom tids- og frekvensdomenet.

To samtidige sinusoider

Dersom vi spiller av to sinusoider samtidig, vil de mikses sammen. Miksing er det samme som at utslagene *summeres*. Denne summeringen (eller *superposisjonen*) kan skje internt i en miksepult eller en datamaskin, eller dersom sinusoidene avspilles fra separate høyttalere, vil



Figur 3.9: Tredimensjonalt spektrogram

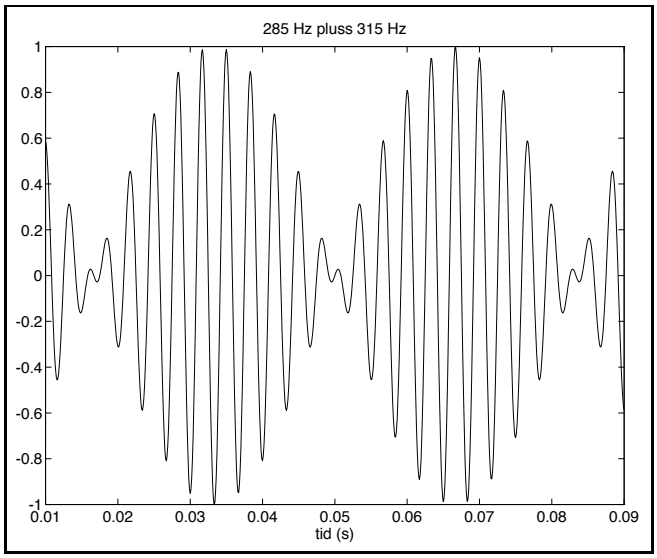


Figur 3.10: Sonogram

summeringen først skje når lydbølgene blandes (*interfererer*) i luften. Vi har allerede sett et eksempel på miksing av sinusoider, i avsnittet om klangfarge.

Hvis disse sinusoidene ligger langt fra hverandre i frekvens, vil vi høre dem som to separate toner. Siden øret deler lyden opp i sine enkelte frekvenskomponenter, kan vi si at vi her oppfatter lyden i frekvensdomenet.

Hvis vi nå skrur frekvensene nærmere hverandre, f.eks. til en forskjell på 30 Hz, vil vi etterhvert få en "sur" og "grumsete" lyd. Vi er kommet inn i det *dissonante overgangsområdet*. Oscillogrammet i figur 3.11 viser den resulterende lydbølgen.



Figur 3.11: Beats

La oss nå skru frekvensene enda nærmere, f.eks. slik at de ligger bare 10 Hz fra hverandre. Vi vil da ikke lenger oppfatte de to sinusoidene separat i frekvensdomenet. I stedet vil vi høre en enkel tone med frekvens som ligger midt imellom de to komponentene. Denne tonen vil pulsere med en frekvens på 5 Hz, som om vi hadde en enkelt tone hvis amplityde ble styrt av en lavfrekvent sinusoid med frekvens 5 Hz (tremolo eller ringmodulasjon). [5] Denne pulseringen kan tydelig sees på grafen over, men der var pulsfrekvensen så høy (15 Hz) at øret ikke fikk den med seg.

Det som nå er skjedd, er at vi begynner å oppfatte lyden mer i tidsdomenet, dvs. at vi kan høre pulseringen utstrakt i tid, som *beats*. Det er viktig å forstå at dette er en egenskap ved *øret*; det er ikke skjedd noen kvalitativ endring av selve lyden i og med at vi skrudde sinusoidene stadig nærmere hverandre. På et spektrogram vil den pulserende sinusoiden vises som de to tette frekvenskomponentene som lyden vitterlig består av.

Litt tallmagi

Hvis vi vil lage en sinusoid med frekvens f_s som pulserer med frekvensen f_p, kan vi altså

[5] Vi vil faktisk høre 10 pulser i sekundet, fordi vi får en amplitydetopp både for den positive og den negative halvperioden til den modulerende sinusoiden. Den sanne periodisiteten er imidlertid 5 Hz, fordi annenhver topp er "snudd opp ned" (denne fotnoten er sikkert bare forvirrende).

enten tenke i frekvensdomenet og addere to sinusoider, slik:

$$f(t) = \frac{1}{2}[\cos(2\pi(f_s + f_p)t) + \cos(2\pi(f_s - f_p)t)],$$

(sinusoidene ligger da $2f_p$ Hz fra hverandre), *eller* vi kan tenke i tidsdomenet og modulere en sinusoid med en annen, lavfrekvent sinusoid, slik:

$$g(t) = \cos(2\pi f_p t)\cos(2\pi f_s t).$$

Hvilken måte vi tenker på er ett fett, $f(t) = g(t)$. For å vise dette gjør vi som alltid når vi trenger et trøstens ord i hverdagen: Slår opp i Rottmanns "Mathematische Formelsammlung". Der finner vi at

$$\frac{1}{2}[\cos(a+b) + \cos(a-b)] = \cos(a)\cos(b).$$

Dette er forøvrig nyttig kunnskap for elektronmusikeren, fordi det forteller hvordan et spektrum forandres når vi utsetter lyden for ringmodulasjon. Dette kommer vi til i den digitale lydbehandlingen.

3.3.1 Kritiske bånd

Vi har sett at når to sinusoider nærmer seg hverandre, vil de bli vanskeligere å skille fra hverandre. Til slutt vil de smelte helt sammen til en enkelt, pulserende sinusoid. Eksakt når slutter vi å oppfatte de separate deltonene? Målinger viser at dette inntrer når sinusoidene kommer innenfor samme *kritiske bånd*. Et kritisk bånd er et frekvensområde som mer eller mindre korresponderer med et avgrenset område på ørets *basiliærmembran*.

Basiliærmembranet er en lang "tunge" som er kveilet opp inne i sneglehuset (cochlea) i det indre øret. Forskjellige områder på dette membranet reagerer på forskjellige frekvenser, og membranet fungerer derfor som første ledd i den kompliserte spektralanalysen som øret foretar. Sansehårene for de laveste frekvensene ligger lengst inne, mens sansehårene for de høyeste frekvensene ligger ytterst ut mot mellomøret. [6]

De kritiske båndene ligger spredt oppover med ca. 1/3 oktavs mellomrom, og overlapper kraftig slik at det ikke blir noen hull i det spekteret vi kan høre.

3.4 Spektre og tonalitet

I dette avsnittet skal vi se mer på komplekse lyder og kritiske bånd. Vi skal se at dette gir oss en rimelig god forklaring på fenomenet tonalitet.

3.4.1 Overtoner og kritiske bånd

Vi har sett at vi får en "grumset" lyd når to sinusoider ligger innenfor samme kritiske bånd. Dette forklarer hvorfor f.eks. C og Ciss (halvtonetrinn) gir dissonans. Tersen er såvidt et stort nok intervall til at vi får konsonans. Men hvorfor er da f.eks. C4 og Ciss5 dissonant? Dette intervallet er jo større enn en ters?

[6] At de "lyse" sansehårene ligger mest ubeskyttet er nok noe av årsaken til at vi mister de høye frekvensene først når øret skades av kraftige lyder.

Vi kan forklare dette ved at vi får kollisjon mellom *overtonene*. Grunntonene C4 og Ciss5 ligger langt fra hverandre, og gir ikke dissonans. Første overtone til grunntonen C4 er imidlertid C5, og den ligger innenfor samme kritiske bånd som grunntonen Ciss5.

En naturlig følge av denne diskusjonen skulle være at to sinustoner alltid vil låte konsonant, unntatt hvis de kommer for nær hverandre. Sinustoner, som bare har en grunntone, kan jo ikke gi kollisjon i overtonene. I litteraturen påstås det at dette faktisk er tilfelle, men det er nok vanskelig å verifisere dette. For det første vil ulineæriteter alltid føre til at vi får enkelte svake overtoner, og for det andre vil enhver musikalsk trenet person umiddelbart gjenkjenne et dissonant intervall, og dermed oppfatte lyden som dissonant.

3.4.2 Musikalske skalaer

Dersom vi ønsker konsonans, bør frekvensene til grunntonene stå i *enkle tallmessige forhold til hverandre*. På den måten unngår vi kollisjoner i overtonene. Forholdet 2:1 (f.eks. 500 Hz og 250 Hz) er veldig konsonant (oktav), mens forholdet 200:141 er dissonant. Andre konsonante intervaller er 3:2 (kvint), 4:3 (kvart), 5:3 (sekst), 5:4 (ters), 6:5 (liten ters) etc.

Hvis vi vil ha en skala der frekvensene står i enkle forhold til hverandre, kan vi gå fram f.eks. slik for C-dur: Vi kaller frekvensen til C for 1 (dette kan skaleres senere). Vi ønsker at C, E og G skal stå i forholdet 4:5:6, og setter derfor opp E og G som $\frac{5}{4}$ og $\frac{3}{2}$ ($1{:}\frac{5}{4}{:}\frac{3}{2}{=}4{:}5{:}6$). Deretter vil vi sette en tilsvarende akkord G,H,D oppå G (dominanten), fremdeles i forholdet 4:5:6. Hvis vi sender D ned i samme oktav får vi etter litt regning at H havner på $\frac{15}{8}$ og D på $\frac{9}{8}$. Til slutt lager vi en treklang på subdominanten (F,A,C) i forholdet 4:5:6, og vi ender opp med den *rettstemte* skalaen (just intonation):

$C = 1, D = \frac{9}{8}, E = \frac{5}{4}, F = \frac{4}{3}, G = \frac{3}{2}, A = \frac{5}{3}, H = \frac{15}{8}, C = 2.$

Hvis vi nå sier at C skal være 100 Hz, vil f.eks. G havne på 150 Hz. Det leie er imidlertid at dersom vi bruker en annen utgangsnote (f.eks. D) når vi konstruerer skalaen vår, vil notene havne på litt andre frekvenser. Kvinten over D, altså A, vil f.eks. plasseres på $100 \cdot \frac{27}{16} = 168.75$ Hz, og det er ikke det samme som A'en vi hadde i C-dur, nemlig $100 \cdot \frac{5}{3} = 166.67$ Hz. Dette betyr at vi må stemme om instrumentene for hver gang vi skifter toneart. Meget upraktisk.

Den mest brukte skalaen er den *tempererte* ("equal temperament"), som lar alle halvtonetrinn være like store. Man passer på at oktaven har forholdet 2:1, men ellers legger man ikke vekt på enkle tallforhold. Heldigvis treffer frekvensene noenlunde riktig likevel. Hvis vi starter med C på 100 Hz, går vi til Ciss ved å multiplisere med 12-roten av 2, eller 1.05946. Vi havner da på 105.946 Hz. Vi lager D ved å multiplisere med 1.05946 enda en gang, og får 112.246 Hz. Kvinten G blir da 149.8 Hz, ikke langt fra 150 Hz som vi hadde over. Etter å ha multiplisert med 12-roten av 2 tolv ganger, har vi gått opp en oktav og havner på eksakt 200 Hz.

Det finnes som kjent mange andre skalaer (pythagoreisk, meantone etc.), og dette er i det hele tatt et stort felt. Vi skal ikke gå mer i detalj om skalaer her.

3.4.3 Harmoniske spektre, cocktailparty-effekten

Dersom et spekter bare inneholder en grunnfrekvens og heltallsmultipla av denne (100 Hz, 200 Hz, 300 Hz osv.), vil øret oppfatte dette som en enkelt lyd med grunntone og sine overtoner, som kommer fra en enkelt lydkilde. Dette kalles et *harmonisk spektrum*.

Dersom et spekter inneholder frekvenser som ikke passer inn i dette, vil hjernen forsøke å finne andre grunnfrekvenser og overtoner som kan være tilstede, og gruppere frekvensene sammen i harmoniske klasser. Hver slik harmonisk klasse vil tolkes som en separat lydkilde.

Det er utrolig hvordan vi på denne måten (i kombinasjon med endel andre "clues") kan splitte et komplisert spektrum opp i enkelte lydkilder. Fra et hav av skravling i et selskap kan vi skille ut og konsentrere oss om en enkelt stemme. Dette kalles *cocktailparty-effekten*. Denne effekten kan observeres i ekstrem grad på Sør-Georgia i Sør-Atlanteren. Når en kongepingvin kommer inn fra havet, kan den raskt lokalisere ungen i en koloni på flere hundre tusen individer, selv om alle ungene er i konstant bevegelse rundt i kolonien. Forsøk har vist at foreldrene kjenner igjen ungen på skrikene, og dette skjer altså i en øredøvende kakafoni der kanskje hundre tusen unger skriker samtidig. En slik utrolig signalbehandling vil vi neppe klare å etterlikne på våre datamaskiner på en god stund ennå.

3.4.4 Virtuelle grunntoner

Hvis hjernen synes at det "burde vært" en grunntone på en viss frekvens, vil vi kunne oppfatte en grunntone selv om det ikke finnes noen. Dersom vi f.eks. lager oss et spektrum med frekvensene 200 Hz, 300 Hz, 400 Hz etc., vil vi oppfatte en *virtuell grunntone* på 100 Hz, og dette gir den pitch vi vil høre.

Bjeller og klokker har ofte inharmoniske spektre der frekvensene står i svært enkle forhold til hverandre. Dette produserer gjerne virtuelle grunntoner.

En annen illustrasjon av dette fenomenet har vi for de dypeste tangentene på et piano. Grunntonen til disse tangentene ligger helt nede under 30 Hz, og dette er helt på grensen av det vi kan høre. Vi kan imidlertid høre overtonene helt utmerket, og ut fra dette konstruerer vi oss en grunntone.

3.5 Musikkinstrumentenes fysikk

Vi kan ikke gå i noen detalj om hvordan musikkinstrumentene fungerer her, men det er viktig å forstå *eksitator-resonator-modellen* som gir en viss forståelse av mange forskjellige instrumenter.

3.5.1 Gitar

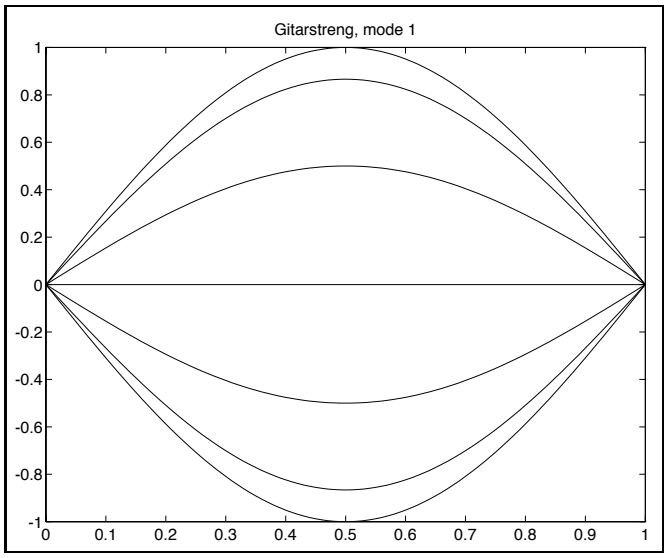
Når vi klimprer en gitarstreng med lengde L meter, vil det forplante seg en bølge til begge sider i strengen. Denne bølgen reflekteres i endene, og farer fram og tilbake. Vi får dermed en svingning som kunne ha vart til evig tid, hvis det ikke var for at friksjonen demper bevegelsene etterhvert.

Hva slags svingninger kan vi så få? Siden strengen er spikret fast i endene, må det kreves at posisjonen her er null, og dessuten at strengen står stille her. [7] Dette gir såkalte *stående bølger*. Strengens form på ethvert tidspunkt kan beskrives som en sum av sinusoider. [8] Den første mulige sinusoiden som er null i begge ender, har bølgelengde lik to ganger strengens lengde ($\lambda = 2L$), og vi får en bevegelse opp og ned overalt på strengen, men mest på midten (figur 3.12).

Dette er en halv sinusbølge, og denne halvdelen har altså en lengde $\frac{\lambda}{2} = L$. Det er derfor vi har at $\lambda = 2L$.

[7] Dette heter *homogene randbetingelser* i matematikken

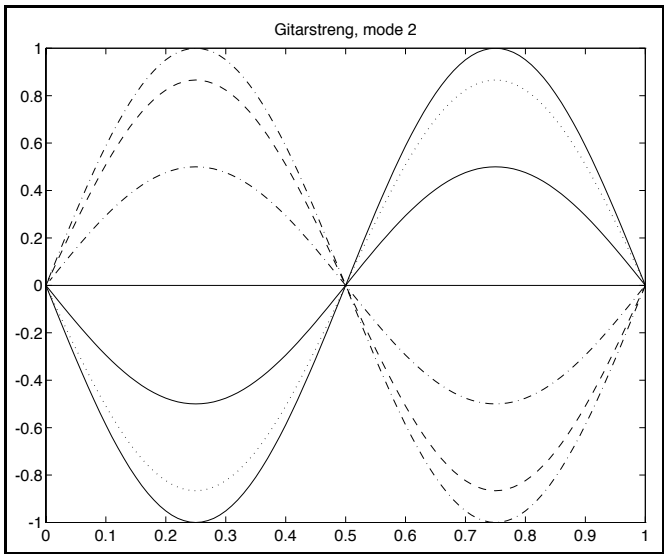
[8] Hvis vi tenker oss at vi utvider strengen periodisk, følger dette av Fouriers teorem



Figur 3.12: Grunnsvingning på gitarstreng

Hvilken frekvens vi får, vil foruten bølgelengden avhenge av lydhastigheten gjennom strengen, som igjen avhenger av strengens tykkelse, masse og stivhet. Sammenhengen mellom frekvens, fart og bølgelengde er $f = \frac{v}{\lambda}$.

Den andre mulige sinusoiden har bølgelengde lik strengens lengde ($\lambda = L$) og vi får en bevegelse opp og ned med en stillestående *knute* på midten (figur 3.13). Siden bølgelengden



Figur 3.13: Første overtone på gitarstreng

her er halvparten av den over, vil denne svingningen ha dobbelt så høy frekvens.

Vi kan også ha svingninger med bølgelengde lik halvparten av strengelengden, en tredel osv. Disse bølgelengdene vil gi en grunnfrekvens (mode 1) med overtoner (mode 2, 3 etc.). Alle disse svingningene vil være tilstede på en gang, og summere seg sammen til et helt spektrum med grunntone og overtoner. Hvis strengens lengde varierer, vil bølgelengdene variere tilsvarende,

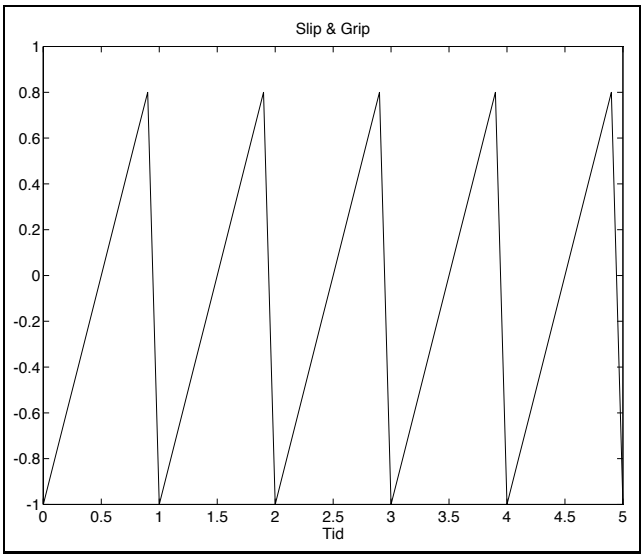
og vi får ulike grunnfrekvenser.

Spekteret fra strengen farges videre av gitarkassen, som virker som et filter med forskjellige *resonanser*, dvs. at visse frekvensområder framheves.

I dette tilfellet kan vi kalle strengen for en *eksitator* (lydprodusent), og gitarkassen for en *resonator*.

3.5.2 Fiolin

Etter at vi har klimpret en gitarstreng svinger den fritt. På en fiolin derimot, opprettholdes svingningen av buens bevegelse. Dette kan vi kalle *tvungen svingning* (forced vibration). Denne mekanismen fungerer ved at friksjonen mellom buen og strengen gjør at strengen dras med buen et lite stykke. På et visst tidspunkt blir imidlertid motkraften for stor, og strengen spretter momentant tilbake. Så tar buen tak i strengen igjen, og prosessen gjentar seg. Fysikere kaller denne prosessen for *slip and grip*. Vi får dermed en slags sagtannkurve (figur 3.14).



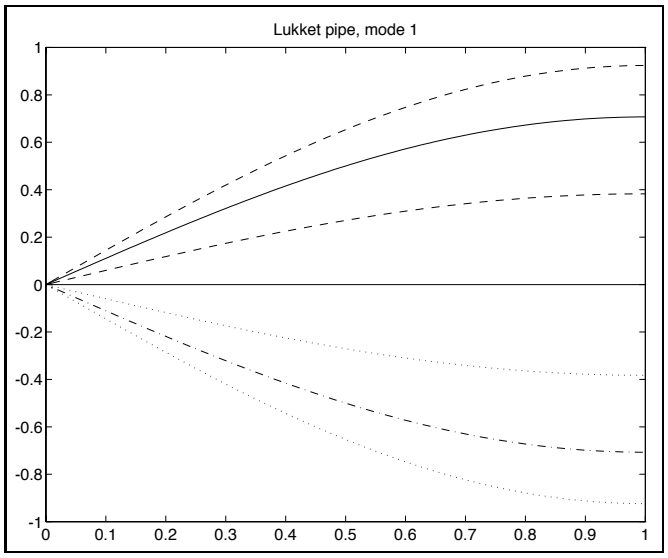
Figur 3.14: Sagtannkurve

Vi får samme type svingning når vi f.eks. drar et bord bortover gulvet. Den fryktede skrikingen av kritt mot tavle er et annet eksempel.

Som for gitaren, vil strengen ha visse frekvenser (grunntone og overtoner) som det er mulig for den å svinge på. Det er viktig at frekvensen til sagtannkurven faller sammen med en slik naturlig frekvens. Det ville vært en håpløs oppgave for fiolinisten å måtte tilpasse frekvensene manuelt. Heldigvis skjer det en tilbakekobling (feedback) fra strengen til buen, som gjør at de to frekvensene låses til hverandre helt av seg selv.

3.5.3 Piper med en lukket og en åpen ende

Hvis en trykkbølge farer fram og tilbake i et rør som er lukket i den ene enden og åpent i den andre, vil det settes opp en stående bølge. Vi får null utsving (knute) i luftttrykket i den åpne enden, og maksimalt utsving i den lukkede enden. Som vist i figur 3.15, der den åpne enden er til venstre, gir dette modus 1 (grunntonen).

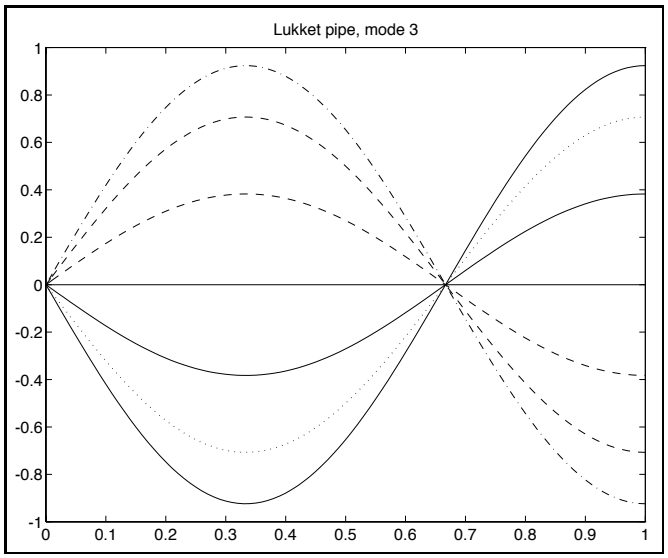


Figur 3.15: Grunnsvingningen i en pipe med en lukket og en åpen ende

Hvis røret er L meter langt, ser vi at bølgelengden til grunntonen er $\lambda = 4L$ meter. Fra sammenhengen mellom frekvens, fart og bølgelengde ($f = \frac{v}{\lambda}$) får vi frekvensen til grunntonen, f.eks. for et 140 cm langt rør:

$$f_1 = \frac{v}{4L} = \frac{343m/s}{4 \cdot 1.4m} = 61Hz.$$

Neste mulige stående bølge er vist i figur 3.16.



Figur 3.16:

Bølgelengden er her $\lambda = \frac{4L}{3}$, som gir frekvensen

$$f_3 = \frac{v}{\frac{4L}{3}} = 3f_1 = 184Hz.$$

Videre vil vi kunne få svingninger med frekvens $f_5 = 5f_1$, $f_7 = 7f_1$ etc. Vi ser at vi får et spektrum med *bare odde overtoner*.

Parasaurolophus

Andeøglene (hadrosaurene) er en gruppe dinosaurer fra Kritt-tiden. Mange andeøgler har en underlig "hjelm" på hodet, som inneholder lange rør av bein. Mange rare forklaringer ble gitt på dette tidligere. Det var f.eks. en vanlig misforståelse at disse svære dyrene levde i vann, og at hode-apparatet ble brukt som luft-reservoar eller snorkel.

Det er nå klarlagt at det dreier seg om et akustisk apparat, en resonator som hjalp til i produksjonen av brøl og støt som må ha vært helt fantastiske. Et eksempel er Parasaurolophus, som hadde en lang kam på hodet, hvor to parallelle piper var kveilet dobbelt. Pipene begynner nede i halsen (lukket ende), hvor det formodentlig satt en slags eksitator. Det finnes noen skrå beinflater som minner mistenkelig om dem vi finner i orgelpiper, så kanskje dyret bare blåste luft fra sine digre lunger. Pipene endte i hvert sitt nesebor (åpen ende), men det er ikke funnet noe som likner på sjalstykker. Den amerikanske paleontologen Jim Bakker har laget en plastmodell som man kan bruke som blåseinstrument, og det pågår arbeid med digital fysisk modellering for å syntetisere lyden.

Lengden på hver pipe er tilsammen ca. 2.4 meter, [9] og ut fra dette kan vi regne ut frekvensen til grunntonen:

$$f_1 = \frac{v}{4L} = \frac{343m/s}{4{\cdot}2.4m} = 36Hz.$$

Dette er virkelig en lav frekvens, og det betyr at lyden bar langt. Dagens elefanter kommuniserer også på svært lave frekvenser, tildels under menneskets hørbare område (subsonisk lyd eller infralyd).

3.5.4 Trompet

Et trompet er ikke et enkelt rør, og dette gir betydelige endringer sammenliknet med situasjonen over. Et komplisert samspill mellom munnstykke, rør og sjalstykke medfører at vi får resonanser for f_2, f_4, f_6 etc. Vanlig dyp tone (med alle ventiler åpne) for en messingblåser har grunntonen på f_4, og overtonene på f_8, f_{12} etc. En trompet har et ca. 140 cm langt rør, og dette skulle gi en f_1 på 61 Hz, og dermed grunntone f_4=244 Hz. Dette er omtrent hva man observerer.

Når man spiller en trompet, vil svingningene fra leppene (eksitator) synkroniseres med frekvensen til en av de mulige svingningsmodi, akkurat som vi så for fiolinen. Dette gjør at det er nærmest umulig å spille toner mellom de som er gitt fra rørets lengde. For å utvide antallet tilgjengelige noter, kan man endre rørlengden med ventiler eller glidende rør (trombone).

Sjalstykket ("tuten") fungerer som en pussig resonator, som former spekteret videre.

3.5.5 Menneskestemmen

Menneskestemmen kan også innpasses i eksitator-resonator-modellen. Eksitator er stemmebåndene, som sitter i strupehodet (glottis). Disse svinger med et meget rikt overtonespektrum. Grunnfrekvensen styres først og fremst av strammimngen av stemmebåndene, men lengden på røret

[9] Målt på et eksemplar som er utstilt på Paleontologisk Museum i Oslo. Gå og ta en titt selv!

over strupehodet spiller også inn. Hvis man synger en lys tone, vil strupehodet plassere seg høyt oppe i halsen. Når man går ned i tonehøyde, vil man tydelig kunne se at strupehodet faller langt nedover. Hastigheten lyden forplanter seg med i røret er også viktig, fordi $f = \frac{v}{\lambda}$. Hvis man fyller rommet over glottis med helium, vil lyden gå fortere og tonehøyden vil øke.

Når man hvisker, blåser man bare luft istedet for å lage en svingning med en grunnfrekvens. Like over glottis vil vi da ha *hvit støy*, dvs. at alle frekvenser er like sterkt til stede i et kontinuerlig spektrum.

Enten man snakker eller hvisker, vil luftrøret over glottis, samt munnhulen, fungere som et filter (resonator) som farger det rike spekteret fra glottis på ulike måter avhengig av hvordan vi former munnhulen. På den måten oppstår de ulike vokalene.

3.6 Lokalisering av lydkilder

3.6.1 Avstanden til lydkilden

Hvordan kan vi oppfatte avstanden til en lydkilde? Den viktigste retningslinjen er nok lydstyrken; amplityden faller proporsjonalt med avstanden. Men dette er ikke så enkelt som det umiddelbart kan virke, for lydstyrken blir den samme enten vi har en sterk lyd langt unna eller en svak lyd som er nær oss. Dette betyr at hjernen er nødt til å ha en ide om hvor sterk lyden var i utgangspunktet. Vi kjenner lydstyrken til de fleste lydkilder (biler, kirkeklokker), og kan derfor bedømme avstanden forholdsvis godt. Riktignok kan man rope både svakt og kraftig, men dette gir endringer i stemmekvalitet slik at vi kan ta dette med i beregningen og få en god avstandsbestemmelse.

Etterhvert som lyden forplanter seg gjennom lufta endrer den karakter. Luft virker som et lavpassfilter, og lyden mister mer diskant jo lenger den går. Dersom vi har en ide om det opprinnelige frekvensinnholdet, hjelper dette oss i avstandsbedømmelsen.

For lyder innendørs er romklangen en viktig ledetråd. Hvis lyden er langt unna, vil etterklangen (den reflekterte lyden) få større lydstyrke i forhold til den direkte lyden.

Alle disse tre effektene bør brukes hvis vi vil gi en realistisk simulering av avstand.

Som sagt er det viktig for avstandsbedømmelsen at vi vet hvor kraftig lyden var i utgangspunktet, og hvilket frekvensinnhold den hadde. Syntetiske lyder, og særlig sinustoner (som bl.a. ikke mister diskant med avstanden), er nesten umulige å avstandsbedømme. *Endring* i avstanden hører vi imidlertid lettere, som vi skal se i neste avsnitt.

Doppler-effekten

Dersom en lydkilde beveger seg mot oss, vil lydkildens egen hastighet komme i tilleg til hastigheten som lyden beveger seg mot oss med. Dermed vil flere bølgetopper enn normalt nå oss pr. sekund, og den høyere frekvensen medfører et tydelig hørbart pitch-skift oppover. Tilsvarende vil vi få et pitch-skift nedover når lydkilden beveger seg fra oss. Dette kalles *Doppler-effekten*. Frekvensene vil multipliseres med en faktor k som er gitt ved

$$k = \frac{v}{v + v_s}$$

der v er lydens hastighet (343 m/s) og v_s er hastigheten lydkilden beveger seg mot (negativ v_s) eller fra (positiv v_s) observatøren med. Når $v_s = 0$ blir $k = 1$, og vi får ikke noe pitch-skift.

Bemerk at når lydkilden beveger seg mot observatøren med lydens hastighet ($v_s = -v$), får vi null i nevner. Dette betyr at frekvensen går mot uendelig; vi har en *singularitet* som medfører at alle bølgetoppene faller sammen og summeres til et kraftig smell. Det er dette som heter å *bryte lydmuren*.

3.6.2 Retningen til lydkilden

Retningsbedømmelse er en meget komplisert affære, og mye av kunnskapen om dette har vi fått først i senere år. Det viser seg at det er mange forskjellige effekter som utnyttes sammen for å gi en nøyaktig bestemmelse av retningen til lydkilden.

Amplityde som retningsindikator

Hvert av våre to ører har størst følsomhet rett ut til siden. Dette betyr at lyd som kommer rett fra venstre registreres kraftigst i venstre øre, mens lyd som kommer rett fra høyre registreres kraftigst i høyre øre. Dette kan benyttes for å gi en grov retningsbestemmelse. Imidlertid kan vi ikke ut fra dette vite om en lyd kommer forfra eller bakfra, ovenfra eller nedenfra. Vi får bare en viss informasjon om retning venstre-høyre.

Panoreringsknappen som finnes på miksebord styrer retningen til lyden bare ved å regulere lydstyrken i venstre og høyre høyttaler. Dette må betraktes som primitivt.

Mange pattedyr av rovdyrenes orden jakter på små byttedyr som er ute om natten og/eller befinner seg under jorden eller snøen. For slike bytteetere er korrekt retningsbestemmelse av stor betydning, og vi ser ofte at de har parabolformede og meget unidireksjonale ører. Katt og rev er gode eksempler. Hensikten med f.eks. ørkenrevens store ører er kanskje ikke først og fremst å samle mye lyd, men å gi stor retningsselektivitet. Ved å vri ørene i ulike retninger kan dyret finne et maksimum i responsen og derved bestemme retningen til byttet med stor nøyaktighet [10]. Vi er her inne på et spennende felt som vi kan kalle *bioakustikk*, dvs. læren om hvordan og hvorfor dyrene produserer og lytter til lyd.

Fase

Dersom en lydkilde befinner seg til venstre for oss, vil alle "toppene" i trykkvariasjonene komme tidligere til venstre enn til høyre øre. Hvis vi regner at ørene sitter 20 cm fra hverandre (?), vil vi kunne få en maksimal tidsforskjell på $\frac{0.2m}{343m/s} = 0.6ms$. Denne tidsforskjellen faller til null når lydkilden befinner seg rett foran/bak/over/under oss, og dette gir oss ytterligere informasjon om lydkildens retning venstre/høyre.

Retningsbestemmelse basert på fase blir mer unøyaktig når vi går oppover i frekvens. Når vi kommer til ca. 2 kHz, vil en tidsforskjell på 0.6 ms tilsvare avstanden mellom bølgetoppene. Begge ører vil derfor oppfatte samme fase, selv om de egentlig registrerer hver sin topp.

Faseinformasjon er den viktigste retningslinjen for å bestemme retningen venstre-høyre for lave frekvenser. For høye frekvenser blir amplityde det viktigste.

Harlekinfroskene i Mellom-Amerika er bare ca. 1.5 cm lange, og har smale hoder. Avstanden mellom ørene blir dermed for liten til å gi god retningsbestemmelse. Disse froskene har derfor

[10] At ørkenreven har så store ører kan delvis også ha med temperaturkontroll å gjøre

utviklet lydfølsomme membraner på hoftene, der kroppsbredden er størst.

Entydig retningsbestemmelse hos dyr

Amplityde og fase kan sammen gi en god retningsbestemmelse høyre-venstre. Det gjenstående problemet er hvordan man finner retningen opp-ned og bak-foran. Som vi har sett benytter katter seg av at de kan snu på ørene i nesten alle retninger og dermed lokalisere lydkilden i alle plan. [11]

Ugler må ha god retningsbestemmelse opp-ned. Ugler i våre strøk er spesielt avhengige av dette, fordi de gjerne sitter lavt over bakken og skal lokalisere mus under snøen. Hvis byttet feilaktig lokaliseres i en høy retning, vil ugla slå ned langt bortenfor musa. Enkelte ugler, som haukugla, bestemmer retningen ved å legge hodet 45 grader på skakke mot venstre og foreta en høyre-venstre-bestemmelse i dette skrå planet. Deretter legger de hodet 45 grader mot høyre og foretar en ny bestemmelse. Etter en rask kalkulasjon beregner de krysningslinjen mellom de to planene og finner en entydig retningsvektor i rommet. Mange ugler finner tydeligvis dette upraktisk, særlig i flukt. De fleste ugler i nordlige strøk har derfor utviklet asymmetriske ører, der bl.a. det ene øret er plassert mye høyere på hodet enn det andre. På den måten kan de gjøre to retningsbestemmelser ved å rotere hodet i stedet for å legge det på skrå.

Ugler har forøvrig mange andre underlige mekanismer for å bedre hørselen. [12]

Frekvensinnhold

Vi mennesker benytter en kanskje enda mer fantastisk, men neppe like nøyaktig, metode for å finne retningen opp-ned og bak-fram. Hele hodet, men særlig ytterøret, virker som et filter som farger lydspekteret ørlite, avhengig av hvilken retning lyden kommer fra. De intrikate brusk-vindingene (pinnae) i det ytre øret bidrar betydelig til dette. Hvis vi så kjenner lydens opprinnelige frekvensinnhold, kan vi anslå lydens retning entydig. Igjen betyr dette at syntetiske lyder, som hjernen ikke har noen modell for, kan være vanskelige å retningsbestemme opp-ned. Sinustoner kan ikke få endret sitt spektrum, fordi de består av bare en deltone. Sinusoider er derfor ekstra vanskelige å retningsbestemme opp-ned og bak-fram.

I de seneste år er det kommet digitale filtre som etterlikner frekvensresponsen til det ytre øret for ulike retninger. På den måten kan man panorere en lyd opp-ned og bak-fram i bare to kanaler.

Haas-effekten

Når vi hører en lyd f.eks. i en konsertsal, vil vi først motta den opprinnelige lyden fra den retningen den ble produsert. Deretter vil vi høre et antall refleksjoner fra alle mulige retninger.

[11] Når en katt jakter, peker ørene framover mot byttet som den vil lokalisere. Når den er redd, peker ørene utover, slik at den får en mer omnidireksjonal respons som kan oppfange fiender som kan komme fra alle retninger.

[12] Se Norges Dyr, Fuglene, bind 5

Hjernen plukker ut den "kopien" som ankommer først, og anvender denne til retningsbestemmelsen[13].

Vi kan benytte dette til å simulere lydens retning, enten alene eller i kombonasjon med vanlig panorering. Hvis vi presenterer lyden noe tidligere for venstre øre (v.h.a. en liten digital-delay i høyre kanal) får vi en tydelig følelse av at lyden kommer fra venstre. Dette kalles *Haas-effekten*. Forsinkelsene som eksisterer i MIDI kan gi uønsket panorering pga. Haas-effekten.

3.7 Morsomme småting

3.7.1 Cerebral hemisfære-dominans

Hørselsnervene krysses, slik at lyd som kommer inn i venstre øre behandles i høyre hjerne-halvdel og omvendt. Det viser seg at ikke- musikere gjenkjenner melodier lettere i venstre øre, det vil si at det meste av behandlingen skjer i høyre hjernehalvdel. Det er høyre hemisfære som tar seg av mønstergjenkjenning, kombinasjonsevne etc. Musikere derimot, gjenkjenner melodier lettere i høyre øre, og det betyr at musikere har lært seg til å oppfatte lyd mest med venstre hjernehalvdel. Venstre hemisfære er sentrum for språk og logikk.

3.7.2 Shepard-toner

Hvis man lager seg en grunntone med overtoner som langsomt faller i frekvens, får man en fallende glissando. Dersom man da langsomt fader ut de dypeste deltonene ettersom deres frekvenser nærmer seg null og langsomt fader inn nye deltoner med høye frekvenser, vil tonen late til å falle i det uendelige. Dette er en ganske pussig effekt.

3.7.3 Differanse-toner

Hvis man spiller to sinustoner kraftig, vil ulineæriteter i overføringsmediet og i øret føre til at vi kan høre en tredje tone med frekvens lik differansen av de to. 500 Hz og 400 Hz vil således gi en differanse-tone på 100 Hz. Det er en utbredt misforståelse å blande dette sammen med beats. Disse to tingene har *ingenting* med hverandre å gjøre!

3.7.4 Sympatiske vibrasjoner

Hvis man setter f.eks. en streng i svingning, vil lydbølgene kunne indusere en svingning i en streng som er plassert et stykke unna. Dette kalles *sympatisk vibrasjon*. Effekten er spesielt sterk dersom den passive strengen har en naturlig svingefrekvens (resonans) som ligger nær frekvensen til den aktive svingeren.

En flott demonstrasjon av sympatisk vibrasjon får vi ved å rope ned i et flygel. Hver av delfrekvensene i ropet vil da indusere en svingning i den strengen som har nettopp denne resonansfrekvensen. Siden tangentene på flygelet dekker det meste av frekvensområdet, vil de induserte svingningene i store trekk gjenoppbygge spekteret i ropet, og vi får en vakker etter-klang hvor stemmens klang kan gjenkjennes. Dette er faktisk en akustisk versjon av vocoding.

Hardingfela er et annet eksempel, der bordunstrengene (understrengene) svinger i sympatisk vibrasjon med hovedstrengene.

[13] Senere refleksjoner gir ytterligere informasjon som også bidrar til retningsbestemmelsen

Kapittel 4

Sampling og rekonstruksjon

4.1 Diskretisering

Lyd er en analog signaltype, som må omformes til digital form for å kunne behandles av en digital datamaskin. Denne omformingen kalles *sampling*, og utføres av et apparat kalt en *analog/digital-omformer* (A/D-omformer).

Et analogt signal er *tidskontinuert*, det vil si et det kan endre seg kontinuerlig over tid og er definert til ethvert tidspunkt. Det er også *amplitydekontinuert*, dvs. at momentanutslaget kan anta enhver verdi innenfor et intervall som begrenses av systemets dynamiske område.

For å kunne behandles av en digital datamaskin med endelig hukommelse, ordbredde og regnehastighet, må signalet bringes over i *diskret form*. Dette vil si at signalet måles på et endelig antall tidspunkter (signalet gjøres tidsdiskret) og med et endelig antall mulige måleverdier (signalet gjøres amplitydediskret). Denne prosessen medfører en forringelse av signalet, som vi ønsker å minimalisere.

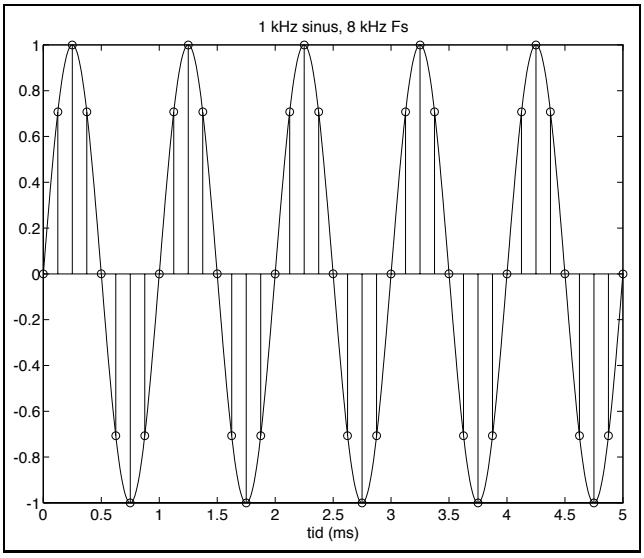
Omformingen fra et tidskontinuert til et tidsdiskret signal er en følge av at vi måler (sampler) signalet på et endelig antall tidspunkter. Hvilke verdier signalet antar mellom disse målingene kan vi ikke vite, og signalet er derfor ikke definert mellom samplingstidspunktene. I de systemer vi skal omtale, er tidsrommene mellom målingene konstante, slik at vi kan snakke om en konstant målefrekvens. Denne frekvensen kalles *samplingfrekvensen*. Hvis vi måler signalverdien 10 000 ganger i sekundet, sier vi at samplingfrekvensen er 10 kHz. Et sentralt spørsmål når vi skal sample et signal, er hvor høy samplingfrekvens vi bør bruke. Åpenbart ønsker vi å benytte en så lav samplingfrekvens som mulig, uten at signalet forringes for mye. Derved begrenser vi datamengden, og dermed kravene til datamaskinens hukommelse og regnehastighet.

4.2 Samplingsteoremet, aliasering

En grunnleggende regel i samplingsteorien er at samplingfrekvensen må være mer enn dobbelt så høy som den høyeste frekvenskomponenten vi ønsker å kunne registrere. Sagt omvendt: Den høyeste frekvenskomponenten vi kan registrere er mindre enn halve samplingfrekvensen. Dette er en del av *Shannon og Nyquist's samplingsteorem*. Halvparten av samplingfrekvensen kalles *Nyquist-frekvensen*. Vi skal ikke teoretisere for mye rundt samplingsteoremet, men illustrere det ved et eksempel. Dette eksemplet vil også demonstrere det fryktede *aliaserings*-fenomenet.

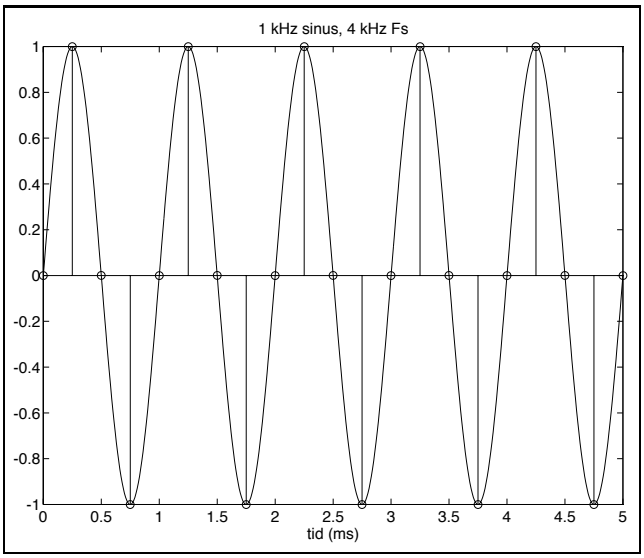
Figur 4.1 viser et analogt, sinusformet signal, la oss si med frekvens 1 kHz (1000 Hz). Vi

ønsker å sample dette signalet, dvs. måle momentanverdien med jevne mellomrom. Figuren viser signalet samplet med 8 kHz samplingfrekvens. De målte verdiene vises ved hjelp av stolper. Vi ser at signalet er godt registrert.



Figur 4.1: 1 kHz sinusoid samplet med 8 kHz

I figur 4.2 går vi ned til 4 kHz samplingfrekvens. Signalet er fremdeles registrert.

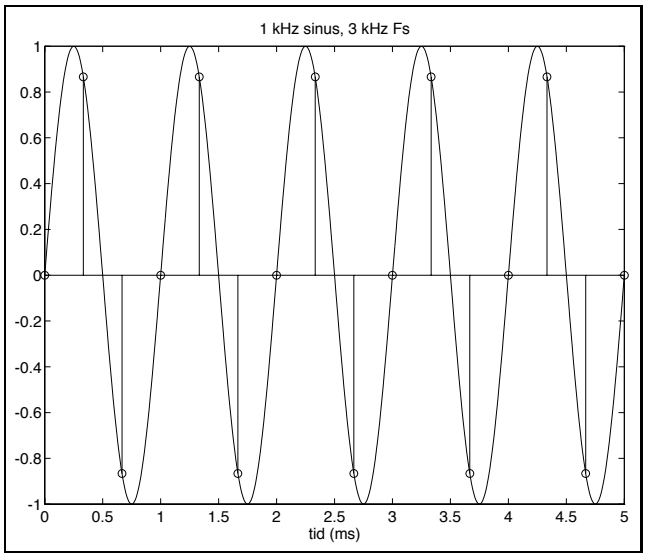


Figur 4.2: 1 kHz sinusoid samplet med 4 kHz

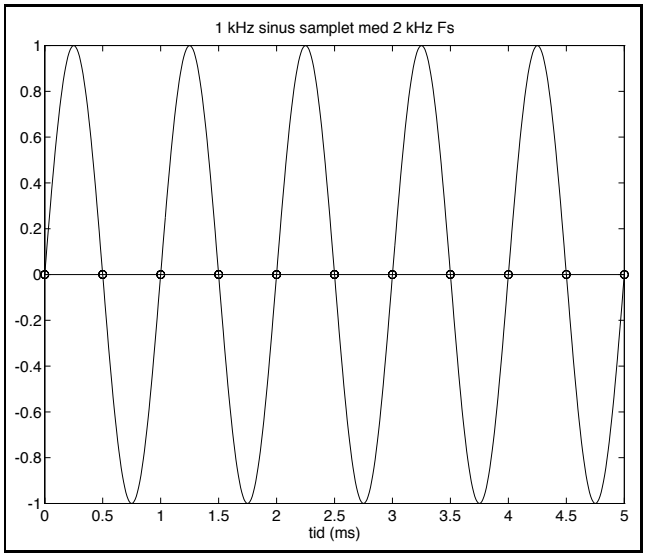
I figur 4.3 sampler vi med 3 kHz, signalet er fremdeles registrert.

Så går vi helt ned til 2 kHz samplingfrekvens (figur 4.4). Vi ser at det går dårlig. I dette spesielle tilfellet er vi så uheldige å sample akkurat i nullgjennomgangene, og vi får ikke ut noen ting.

En samplingfrekvens på 3 kHz er altså tilstrekkelig for å registrere en frekvenskomponent på 1 kHz, mens 2 kHz blir for lite. Dette stemmer med samplingsteoremet.

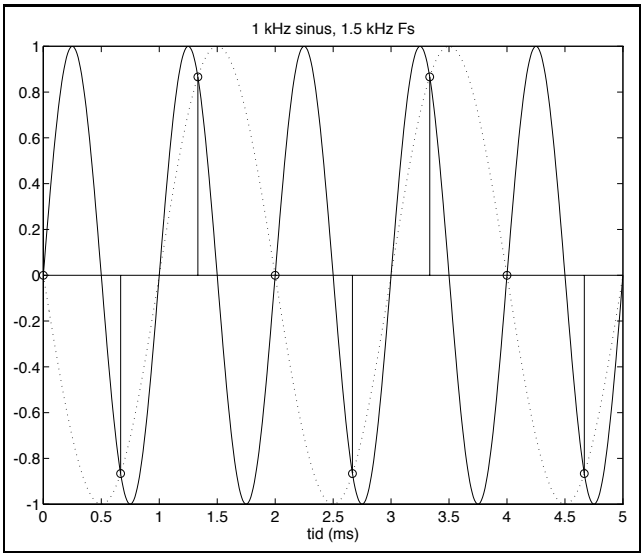


Figur 4.3: 1 kHz sinusoid samplet med 3 kHz



Figur 4.4: 1 kHz sinusoid samplet med 2 kHz

La oss så forsøke å senke samplingfrekvensen ytterligere. I figur 4.5 sampler vi med 1500 Hz. Det samplede signalet beskriver nå en sinustone på bare 500 Hz (stiplet linje).



Figur 4.5: Undersampling

Signalet på 1000 Hz er altså blitt feilregistrert som et signal på 500 Hz. Med andre ord blir signaler som ligger over halve samplingfrekvensen foldet (speilet) nedover i spekteret, og danner falske frekvenser. Dette er et alvorlig problem i digital lyd, som kalles *aliasering*. Vi tar det en gang til:

Aliasering er et fenomen som kan oppstå i en samplingsprosess, dersom signalet vi sampler inneholder frekvenskomponenter over halve samplingfrekvensen. Disse komponentene speiles nedover i spekteret og danner falske frekvenser.

Bemerk at "samplet signal" også inbefatter signaler som er beregnet ved en matematisk metode internt i en datamaskin. Under beregningene må vi alltid passe på at signalet ikke inneholder komponenter over halve samplingfrekvensen. Disse vil forårsake aliasering. Et enkelt forsøk på en FM-synthesizer (DX 7) vil demonstrere dette. Hent inn en lyd med et rikt spektrum, og spill den øverst på klaviaturet (bruk ev. *Key Transpose* for å komme enda høyere opp). De lyseste komponentene blir speilet nedover, og forvrenger lyden kraftig.

Når vi skal sample en lyd, må vi altså passe på at signalet er så båndbreddebegrenset at aliasering unngås. For å sikre dette, er det vanlig å sette et lavpassfilter før A/D-omformeren, et såkalt *antialiaseringsfilter*. Antialiaseringsfilteret må være så skarpt som mulig, og klippe like under halve samplingfrekvensen.

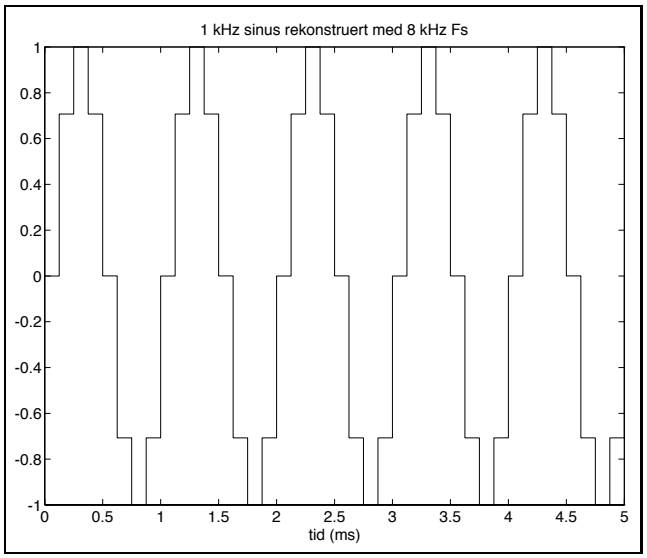
4.3 Rekonstruksjon

Nå er altså signalet samplet og lagret digitalt, en tallverdi for hver måling. I dette avsnittet skal vi se på hvordan tallverdiene kan omformes tilbake til et analogt signal. Dette kalles *rekonstruksjon*, og er den omvendte prosess av sampling. En digital/analog-omformer (D/A-omformer) står sentralt i denne prosessen.

En D/A-omformer produserer en spenning på grunnlag av en digital verdi som blir sendt ut fra datamaskinen. For å rekonstruere et analogt signal ut fra digitale verdier, kan vi således

sende sekvensen av verdier ut på D/A-omformeren med samme hastighet som signalet ble samplet med. Hvis vi har samplingfrekvens 30 kHz, må vi sende 30 000 tallverdier ut på D/A-omformeren hvert sekund. Vi kan også endre denne hastigheten, med hastighets- og pitch-endring som resultat.

Figur 4.6 viser utgangen fra D/A-omformeren når vi sender ut sekvensen som vi fikk da vi samplet en 1 kHz sinus med 8 kHz samplingfrekvens. Tatt i betraktning at det originale, analoge signalet var en ren sinus, ser jo dette ut som et nokså mislykket resultat. Akustisk vil den "firkantede" formen arte seg som masse bråk og ringing fra Nyquist-frekvensen (her 4 kHz) og oppover.



Figur 4.6: Utgang fra D/A-omformeren

Ved å betrakte samplingsprosessen som modulasjon med et tog av enhetsimpulser, kan man vise at dette bråket er speilinger av det originale spekteret om Nyquist-frekvensen og heltallsmultipla av denne. For de som måtte ha glede av det, kan dette formaliseres ved at den Fourier-transformerte av signalet x modulert med toget av forsinkede Dirac-impulser er konvolusjonen av den Fourier-transformerte av x med et tilsvarende tog i frekvensdomenet:

$$\mathcal{F}\left[x(t)\sum_{n=-\infty}^{\infty}\delta(t-nT)\right] = X(f) * \left[\sum_{m=-\infty}^{\infty}\delta(f-mf_s)\right]$$

Det originale spekteret er altså intakt, men har fått masse uønskede komponenter i tillegg, fra Nyquist-frekvensen og oppover. Disse komponentene kan fjernes ved hjelp av et skarpt lavpassfilter, som klipper like under Nyquist-frekvensen. Dette filteret kalles et *rekonstruksjonsfilter*. Etter rekonstruksjonsfilteret skal signalet i prinsippet være perfekt rekonstruert. Trappetrinnene i figuren over vil da være fjernet, og vi vil ha en perfekt sinus igjen. Dette er egentlig et litt merkelig (og for oss svært heldig) resultat [1].

Det komplette samplings- og rekonstruksjons-systemet er vist i figur 4.7. Det analoge signalet lavpassfiltreres i antialiaseringsfilteret, slik at vi sikrer oss mot aliasering. Signalet samples

[1] Dette avsnittet er ikke helt korrekt og fullstendig. En av flere kompliserende faktorer er effekter av pulsbredden fra D/A-omformeren.

og måles så i A/D-omformeren, og lagres og/eller behandles i et digitalt system (datamaskin). Tallverdiene sendes videre ut til D/A-omformeren, som produserer en trappetrinn- aktig versjon av det analoge signalet. Trappetrinnene fjernes så i rekonstruksjonsfilteret.

Figur 4.7: Komplett system for sampling og rekonstruksjon

4.4 Kvantisering

Det finnes enda en viktig faktor som er med på å bestemme lydkvaliteten, nemlig effekten av endelig ordbredde, og dermed avrundingsfeil, i A/D- og D/A-omformerne. Dette gir opphav til *kvantiseringsstøy*.

Ordbredde betyr antall bits (1 eller 0) som brukes for å kode hver sample-verdi. Vi kan til å begynne med tenke oss en 3-bits A/D-omformer. En slik omformer kan skjelne mellom $2^3 = 8$ spenningsnivåer, som kodes med følgende binære verdier (her er fullt utslag 8 volt, og vi regner med bare positive tall):

Spenning inn på omformeren	Kodet verdi	Desimalverdi
Under 1 V	000	0
1-2 V	001	1
2-3 V	010	2
3-4 V	011	3
4-5 V	100	4
5-6 V	101	5
6-7 V	110	6
Over 7 V	111	7

Vi ser her at den amplitydekontinuerte, analoge spenningen, som kan anta enhver verdi mellom 0 og 8 V, blir kvantisert til en av 8 mulige binære verdier. Dette fordi A/D-omformeren har et endelig antall bits (her 3). Kvantisering betyr det samme som at signalet blir gjort amplitydediskret.

Hvilken betydning har så kvantisering for lydkvaliteten? Vi kan finne ut av dette ved å si at kvantisering gir addisjon av støy til det ideelle analoge signalet, som følge av avrundingsfeilene. I eksemplet over ser vi at en virkelig signalverdi på 1.00 V blir kodet som 1. Dermed er signalet

målt korrekt. Men en signalverdi på 2.99 V blir kodet som 2, hvilket tilsvarer addisjon av spenningen -0.99 V til det ideelle signalet.

Vi ser at sampling av signalet med vår 3-bits omformer gjør at spenninger fra 0 til -0.99 V blir addert (mikset) til signalet. Hvilken kurveform disse spenningene danner avhenger av det analoge signalets kurveform, men i de fleste tilfeller vil kvantiserings- støyen arte seg som hvit støy. Denne støyen får amplityden 1 V. Sammenlikner vi dette med analogsignalets maksimale utslag, får vi et signal/støy-forhold på

$$\frac{8V}{1V} \approx 18dB.$$

Et signal-støy-forhold på 18 dB holder jo ikke. For å få et bedre resultat, må vi måle spenningen med større nøyaktighet. Dette krever flere bits i omformeren, slik at vi får flere enn 8 nivåer å skjelne mellom. En 8-bits omformer gir f.eks. $2^8 = 256$ forskjellige binærkoder. Kvantiseringsstøyens amplityde blir da $\frac{8V}{256} = \frac{1}{32}V$. Dette gir signal/støy-forhold

$$\frac{8V}{\frac{1}{32}V} \approx 48dB.$$

Mer generelt kan vi sette opp følgende uttrykk for signal/støy- forholdet som følge av kvantiseringen:

$$SN = \frac{A}{\frac{A}{2^n}} = 2^n \approx 6n\ db\ (effekt)$$

der A er maksimalt utslag og n er antall bits. Av dette får vi følgende tommelfinger-regel: Signal/støyforholdet som følge av kvantisering, målt i effekt-decibel, er tilnærmet lik 6 ganger antall bits i A/D- og D/A-omformerne. [2]

12-bits omforming gir dermed ca. 72 dB S/N-forhold, mens 16 bits gir ca. 96 dB. Dette er imidlertid rent teoretiske verdier, og unøyaktigheter i omformerne og støy i tilliggende analoge kretser gjør at det reelle S/N-forholdet aldri når opp til denne teoretiske grensen.

Vi kan nå oppsummere hvilken innvirkning sampling har på lydkvaliteten:

- Samplingfrekvensen bestemmer frekvensgangen til systemet. Høyeste målbare frekvens er lavere enn halve samplingfrekvensen (Nyquistfrekvensen). Dersom signalet ikke er tilstrekkelig båndbreddebegrenset, oppstår aliasering.
- Dersom utgangen fra D/A-omformeren ikke passerer et godt rekonstruksjonsfilter, får vi støy på høye frekvenser som følge av spektrale foldinger rundt multipla av Nyquist-frekvensen. [3]
- Signal/støy-forholdet er teoretisk begrenset av ordbredden i omformerne. S/N-forholdet i dB er tilnærmet lik 6 ganger antall bits i omformerne.

[2] Igjen er dette noe forenklet, idet kvantiseringsstøyens energi-innhold gjennomsnittlig er noe mindre enn peak-verdiene vi har sett på. En mer korrekt likning er $SN = (6.02 \cdot n + 1.76)dB$.

[3] Dette kan ofte høres som en ringing på samplere når man transponerer langt ned, fordi rekonstruksjonsfiltrene ikke følger med samplingsfrekvensen nedover. Denne effekten blir ofte feilaktig kalt aliasering.

Kapittel 5

Digital lydbehandling

Dette kapittelet omhandler teknikker for å syntetisere, analysere og omforme lyd i digital form ved hjelp av datamaskiner.

5.1 Lydbehandling og syntese i tidsdomenet

5.1.1 Amplityde- og ringmodulasjon

I analoge synthesizere benyttes spenningsstyrte forsterkere (VCA) til å endre amplityden til en lyd. En varierende kontrollspenning styrer da amplityden til lyden som passerer gjennom VCA'en, og vi sier at lyden *amplitydemoduleres* av kontrollsignalet (modulatoren).

I den digitale verdenen gjør vi dette ved å multiplisere hver av sample-verdiene i lyden med tilsvarende samples i kontrollsignalet. Hvis vi multipliserer med en konstant, vil hele lyden endre amplityde uniformt. Multiplikasjon med 0 gir total dempning, multiplikasjon med 0.5 gir 6 dB dempning, 1.0 endrer ikke amplityden, 2.0 gir 6 dB forsterking.

Ved å endre kontrollsignalet over tid, kan vi lage envelopes. Siden øret ikke kan oppfatte hurtige styrkevariasjoner, er det mulig å benytte en lav samplingfrekvens i envelopetabellen, kanskje under 1 kHz. I programmet C-Sound gjøres dette ofte for å spare beregningstid. Det kan da imidlertid ofte oppstå klikk. Den ideelle løsningen er kanskje å bruke en liten envelopetabell, men å interpolere mellom samplene i kontrollsignalet.

Hvis vi modulerer med en sinustone, får vi *ringmodulasjon.* [1] Hvis vi modulerer med en sinustone med frekvens f_m, vil hver av deltonene i lydspekteret forsvinne. I stedet får vi to nye deltoner, med avstand $\pm f_m$ fra den opprinnelige deltonen og med halve amplityden. Disse deltonene kalles *sidebånd.* Som vi forklarte i heftet om akustikk, kan dette vises ved hjelp av formelen for multiplikasjon av to sinusoider:

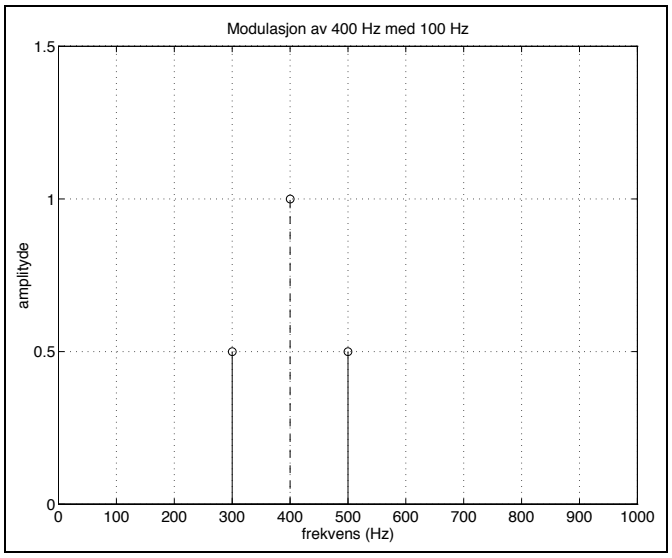
$$cos(a)cos(b) = \frac{1}{2}[cos(a+b) + cos(a-b)].$$

Figur 5.1 demonstrerer dette.

Den opprinnelige frekvenskomponenten på 400 Hz er her utsatt for ringmodulering med modulatorfrekvens 100 Hz. Vi har da fått sidebånd på 300 Hz og 500 Hz.

Ringmodulasjon er en "klassisk" metode for lydomforming som ble mye brukt særlig på 50- og 60-tallet (Stockhausen, Nordheim etc.).

[1] Også kjent under navn som blanding, heterodyning, balansert modulasjon



Figur 5.1: Ringmodulasjon

5.1.2 Miksing

Miksing av digital lyd skjer simpelthen ved at man adderer korresponderende sample-verdier. Dette krever at alle lydene er samplet med samme frekvens. Akkurat som i analog lyd må man passe på at summen holder seg innenfor maskinens dynamiske område, ellers får man klipping.

Når vi mikser vil vi gjerne fade ting inn og ut. Dette gjøres ved å legge på envelopes på hver lyd før addisjonen.

5.1.3 Kurveform-syntese

Den enkleste metode for lydsyntese er at brukeren simpelthen angir kurveformen direkte som et oscillogram. Kurven kan tegnes med et tegneverktøy på skjermen, eller genereres ved en angitt matematisk funksjon. Den vanligste metoden er å legge en periode av kurven inn i en tabell med opptil noen få tusen samples. Denne tabellen presses eller strekkes automatisk, avhengig av hvilken grunntonefrekvens vi ønsker, og så gjentas (loopes) denne perioden så lenge vi vil. Vi kan siden legge på envelopes eller behandle lyden ved andre omformingsmetoder.

Et lite problem ved kurveform-syntese (og generelt i digital lyd) er *interpolasjonsstøy*. Anta for eksempel at vi har en kurvetabell med 100 samples, og at samplingfrekvensen er 20 kHz. Dersom tabellen avspilles som den er, vil det ta $100 \cdot \frac{1}{20000}$ sekunder å spille en periode, hvilket gir en grunntonefrekvens på 200 Hz. Hvis vi nå ønsker en grunntonefrekvens på 300 Hz, må vi gå gjennom tabellen med en hastighet på 1.5 ganger det "normale". Måten å gjøre dette på, er å plukke ut hvert 1.5'te sample for avspilling. I praksis må da synteseprogrammet enten bare hente det nærmeste samplet i tabellen, eller man må interpolere. Slik interpolasjon er aldri perfekt, og dermed får vi støy.

Interpolasjonsstøy kan minimaliseres på to måter: Enten benytte en større kurveformtabell, eller benytte en bedre (og dermed mer beregningskrevende) interpolasjonsmetode. Vi får dermed en avveining mellom hukommelsesbruk, regnehastighet og lydkvalitet.

Kurveform-syntese er morsomt og lærerikt, fordi en lyd kan tegnes helt direkte. Med litt øvelse kan man også oppnå mange spennende effekter. Men øret oppfatter nå engang lyden i

frekvensdomenet, slik at det ofte kan være vanskelig å forutsi hvordan en lyd som er angitt i tidsdomenet vil klinge. Dessuten kan det være kronglete å oppnå dynamikk i klangfarge.

Kurveform-syntese er en metode som har vært kommersielt populær, under navn som "PCM-syntese" (Casio) og "Waveform Synthesis" (Korg).

5.1.4 Additiv syntese

Additiv syntese går simpelthen ut på at brukeren angir spekteret direkte, enten statisk eller varierende over tid. Datamaskinens oppgave er så å omforme lyden fra frekvensdomenet til tidsdomenet. Fra Fourier's teorem følger det at enhver kurveform kan syntetiseres ved hjelp av additiv syntese. Kommunikasjonen med brukeren foregår dessuten i frekvensdomenet, som er den samme representasjonsform som øret benytter. Dette gjør additiv syntese til en meget kraftig metode, bortsett fra at antall parametre (amplityde og frekvens til hver av deltonene) kan virke litt overveldende. Ofte vil imidlertid disse parametrene ha sitt grunnlag i en foregående, maskinell spektralanalyse av en samplet lyd. Vi skal komme mer tilbake til slik *analyse-resyntese*.

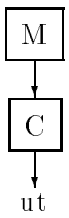
Additiv syntese kan være en nokså beregningskrevende sak, fordi man skal regne ut et stort antall sinusoider med varierende amplityde og mikse disse sammen. En alternativ og langt hurtigere beregningsmetode er å bruke invers Fast Fourier-transformasjon (IFFT). Dette kommer vi også tilbake til.

Det finnes et antall kommersielle synthesizere som benytter additiv syntese (f.eks. fra Kawai).

5.1.5 FM-syntese

FM-syntese er kanskje den mest populære digitale syntesemetoden, ikke minst på grunn av Yamahas DX7-serie. FM er en indirekte og lite intuitiv teknikk. Det krever øvelse å kunne bestemme synteseparametrene riktig for å få en ønsket lyd. Grunnen til denne metodens popularitet er først og fremst at den kan produsere forholdsvis komplekse klanger uten at datamaskinen behøver å slite for mye. Det var amerikaneren *John Chowning* som først pekte på FM-syntese som musikalsk verktøy.

Som en introduksjon til FM kan vi betrakte et oppsett som lager sinustoner med vibrato. Vi benytter to sinusoscillatorer: En oscillator C som lager selve lyden, og en lavfrekvent oscillator (LFO) M som styrer frekvensen til C. Dette kan tegnes slik:



I FM-sjargong kalles M for *modulator*, og C for *carrier*. [2] Frekvensen som M svinger med

[2] Denne ordbruken er tatt fra radioteknikken

kalles modulatorfrekvensen, og gjennomsnittsfrekvensen til C kalles *carrierfrekvensen*. Amplityden til M heter *modulasjonsindeks*, og benevnes her med bokstaven I. Amplityden til C spiller bare rolle for lydstyrken. I dette systemet har vi dermed tre interessante parametre som kan endres: Modulatorfrekvens, carrierfrekvens og modulasjonsindeks.

M svinger her med en lav frekvens, og vi får en vanlig sinus med vibrato. Dersom vi imidlertid øker frekvensen til modulator opp i det hørbare området, vil vi få kraftig endring av klangfargen. Hvilken klangfarge vi får, avhenger av forholdet mellom frekvensen til C og frekvensen til M (såkalt C:M ratio) og av modulasjonsindeksen.

Vi kan sette opp følgende "regler" for hvordan spekteret til ovenstående system er bygget opp:

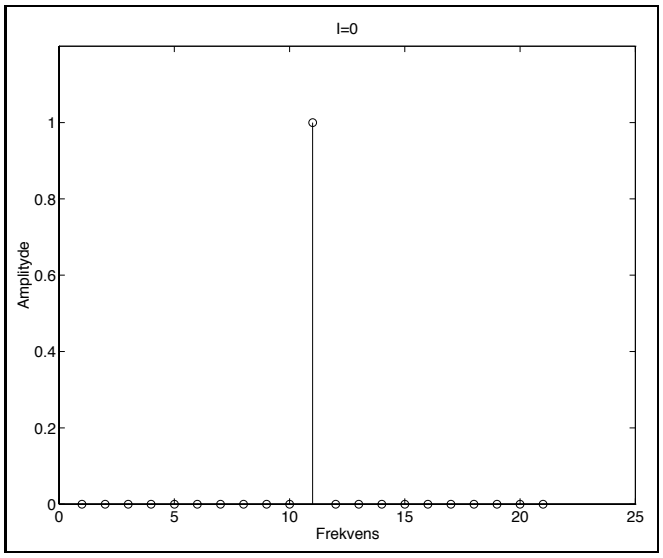
- Carrier vil være tilstede som en deltone i spekteret
- På hver side av carrierfrekvensen vil det oppstå et stort antall nye frekvenskomponenter (sidebånd)
- Avstanden fra carrierfrekvensen til de to nærmeste sidebånd og mellom hvert av sidebåndene tilsvarer modulatorfrekvensen [3]
- Amplityden til carrier og sidebåndene avhenger av modulasjonsindeksen. Jo større I, jo svakere carrierfrekvens og jo sterkere og flere sidebånd. Vi kan si at det stjeles energi fra carrier som fordeles utover på sidene. Dette kan tallfestes ved hjelp av såkalte *Besselpolynomer*.
- Det skjer nesten alltid at de nederste sidebåndene får en frekvens som er mindre enn 0. Disse reflekteres da oppover i spekteret igjen, og blander seg med de andre komponentene.
- Disse refleksjonene forklarer hvorfor heltallige C:M-forhold gir harmoniske spektre (refleksjonene treffer da akkurat på de opprinnelige deltonene), mens ikke-heltallige C:M-forhold gjerne gir inharmoniske spektre (refleksjonene treffer da innimellom de opprinnelige deltonene).

I figurene 5.2-5.6 kan vi se hvordan spekteret blir rikere ettersom vi øker modulasjonsindeksen. Carrierfrekvensen er her så stor i forhold til modulatorfrekvensen at vi ikke får refleksjoner.

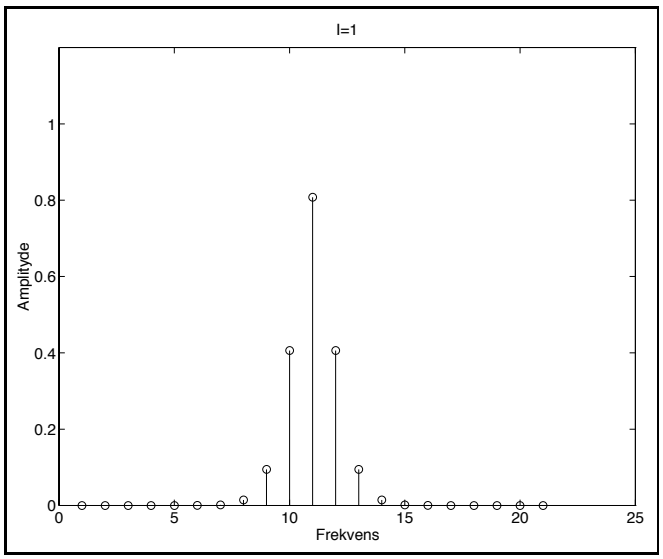
Det er fristende å vise dette i tre dimensjoner, selv om situasjonen kanskje blir litt utydelig (figur 5.7).

Foreløpig har vi betraktet et system med bare to oscillatorer: En carrier og en modulator. Parametrene er dessuten statiske. I FM-syntese pleier vi imidlertid å bruke flere enn to oscillatorer. For å oppnå dynamikk i klangfarge, legger vi envelopes på modulatorene, og for å få dynamikk i styrke, legger vi envelopes på carrierne. Av dette følger en generell byggeblokk som kan brukes både som carrier og modulator, nemlig en oscillator med envelope på utgangen og med styrbar frekvens. En slik byggeblokk kalles en *operator*. DX7 har 6 slike operatorer, som kan kobles sammen på 32 forskjellige måter. En slik sammenkoblingsmåte (patch) kalles en *algoritme*. Her en noen eksempler på FM-algoritmer:

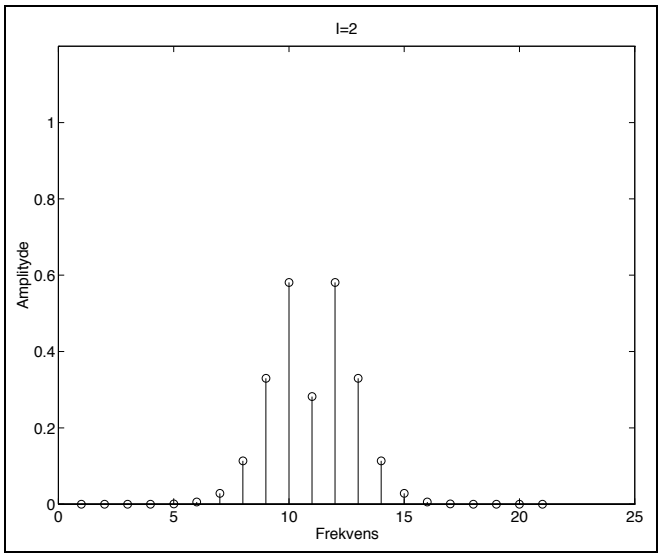
[3] Bemerk analogen til ringmodulasjon, men der hadde vi bare to sidebånd



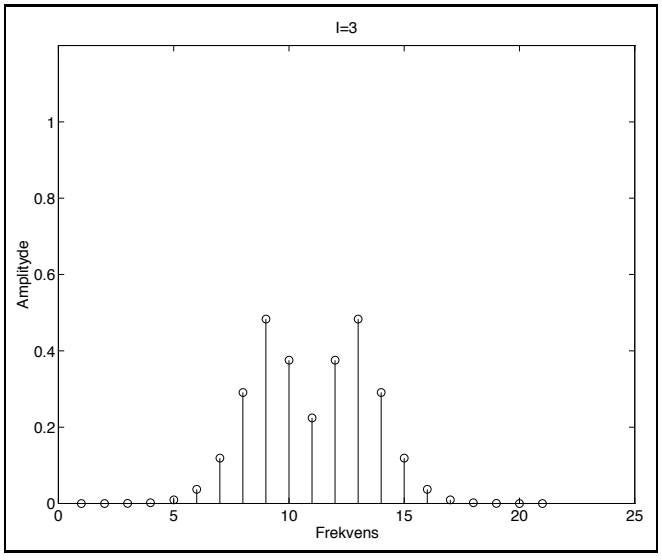
Figur 5.2: FM



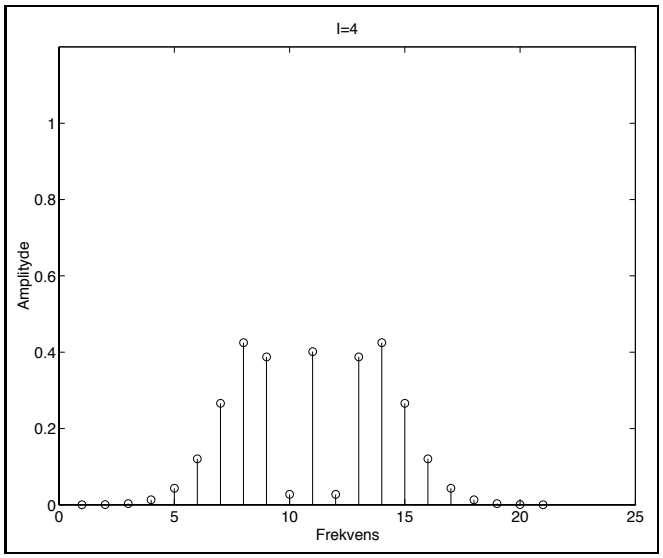
Figur 5.3: FM



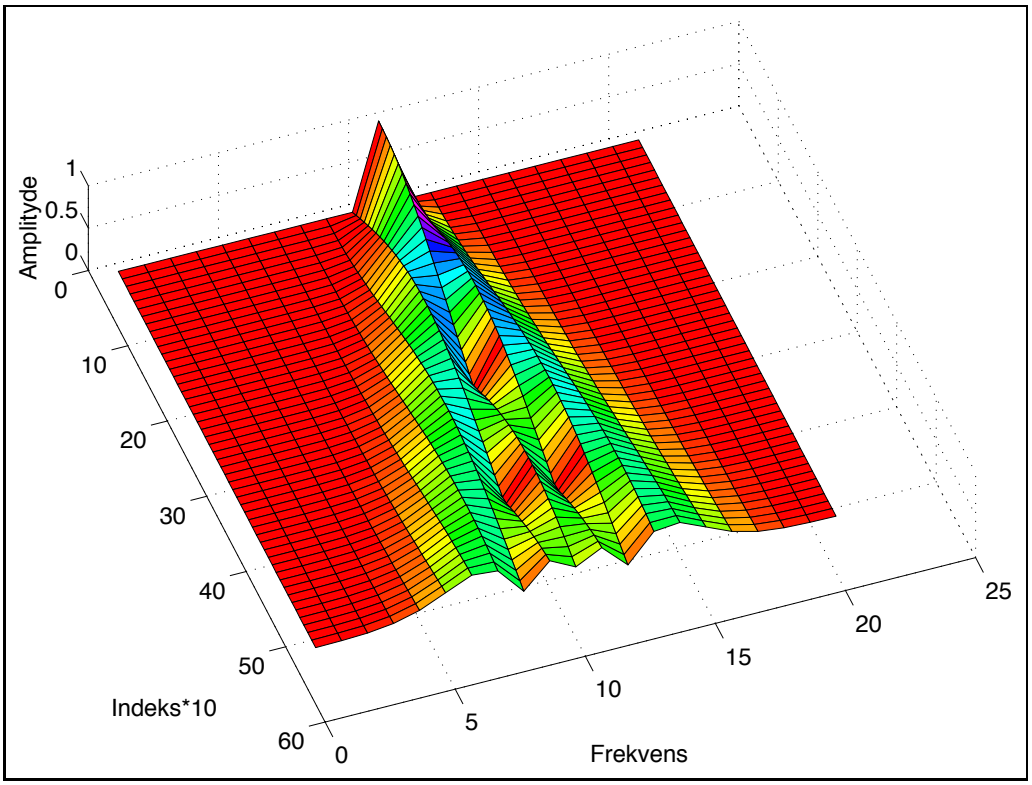
Figur 5.4: FM



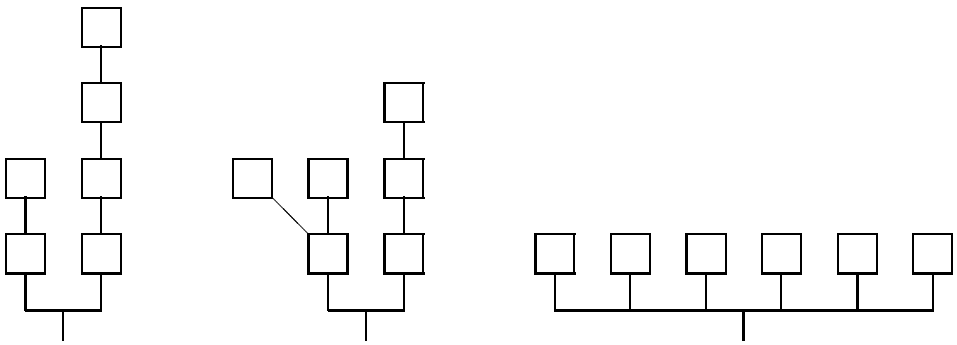
Figur 5.5: FM



Figur 5.6: FM



Figur 5.7: FM-spektrum med varierende indeks



Algoritmen lengst til høyre er egentlig ikke FM-syntese, men en enkel additiv syntese med 6 deltoner.

5.1.6 Ulineær kurveforming

Denne metoden heter på engelsk *Non-linear Distortion*, på fransk *Distortion Non-Lineaire*, forkortet DNL. Poenget er å kjøre lyd (sinustoner eller naturlig lyd) gjennom en ulineær overføringsfunksjon. Denne funksjonen kan f.eks. implementeres som en tabell der input-sampleverdien brukes som en indeks som peker ut en verdi i tabellen. Denne verdien blir output-verdi.

Overføringsfunksjonen kan tegnes på en dataskjerm, eller den kan beregnes matematisk. Enhver funksjon kan legges inn i tabellen. Her er noen eksempler (du finner flere i heftet om matematikk):

$$f(x) = x$$

(identitetsfunksjonen) endrer ikke lyden i det hele tatt, fordi output-samplene $f(x)$ er de samme som input-samplene x.

$$f(x) = 2x$$

vil doble amplityden. Dette gir en forsterking på 6 dB. Disse to funksjonene er begge lineære; funksjonsgrafen er en rett linje. Dette betyr at klangfargen beholdes.

$$f(x) = abs(x)$$

der $abs(x)$ er absoluttverdien av x (fortegnet kastes) er en ulineær funksjon som endrer lydens klangfarge og gir en fuzz-aktig effekt.

Av spesiell interesse er en type overføringsfunksjoner som kalles *Chebychev-polynomer*. Disse polynomene har den interessante egenskap at dersom vi sender en sinustone gjennom dem, får vi en harmonisk ut. Et polynom av andre grad gir andre harmoniske, tredje grad gir tredje harmoniske osv. Ved å legge sammen flere polynomer, får vi et nytt polynom som kan produsere et helt spektrum når vi sender en sinustone gjennom det. Vi har da full kontroll over deltonenes amplityder. Dette forutsetter imidlertid at input-sinusen har amplityde 1. Ved mindre amplityder får vi et fattigere spektrum ut. På denne måten vil klangen endre seg (bli rikere og lysere) med større amplityde. En liknende oppførsel er kjent fra de fleste musikkinstrumenter.

Det bør nevnes at både kurveform-syntese og FM-syntese kan betraktes som spesialtilfeller av ulineær kurveforming. Dette er nok imidlertid mer av teoretisk enn av praktisk interesse.

5.1.7 Amplityde-følging

RMS-analyse (Root-Mean-Square) er den klassiske måte å måle signalstyrke på. En signalstyrkemåler kan brukes til å trekke ut amplityde-envelopen til en lyd. Denne envelopen kan så endres eller anvendes på andre lyder. Dette siste er et enkelt eksempel på *kryss-syntese*, der man trekker ut informasjon fra en lyd og anvender på en annen.

Ved RMS-analyse kvadrerer vi først hvert sample. Dette gjør at vi bare får å gjøre med positive verdier i det følgende. Deretter integrerer vi (tar gjennomsnittet) over et visst tidsrom. Bemerk at dersom vi ikke hadde kvadrert til å begynne med ville enhver sinusoid integrere til null, og resultatet ville vært verdiløst. Til slutt trekker vi ut kvadratroten for å oppveie kvadreringen. Vi beregner altså kort sagt *roten av gjennomsnittet av kvadratet*, derav navnet RMS.

Det interessante spørsmålet her er hvordan og over hvilke tidsrom vi skal ta gjennomsnittet. Det finnes flere forskjellige måter å produsere et gjennomsnitt på:

- *Langtids-gjennomsnitt*, der vi setter integrasjonstiden T lik hele obervasjonstiden. Vi får da ut ett enkelt tall, som er den gjennomsnittlige amplityden for hele lyden.

- *Stegvis gjennomsnitt*, der vi tar gjennomsnittet over en lyd-bit som er T sekunder lang. Deretter tar vi gjennomsnittet over en ny bit som starter ved slutten av den forrige, etc. Vi får dermed ut en amplityde-verdi for hver bit. Hvis $T = 0.1$ sek får vi ut 10 RMS-verdier i sekundet.

- *Løpende gjennomsnitt* er en mer korrekt metode, der vi hele tiden tar gjennomsnittet over de T siste sekundene. Vi får da like mange RMS-samples ut som lyd-samples inn, men siden innholdet av høye frekvenser vil være lite kan vi nokså trygt sample RMS-verdiene med en lavere frekvens uten at vi får aliasering.

- *Vektet gjennomsnitt* tillegger samplene forskjellig vekt i gjennomsnittet. Som oftest blir "gamle" samples dempet i forhold til "nye". Lavpassfiltre som demper på en meget lav frekvens (størrelsesorden 20 Hz) kan brukes til en slik gjennomsnitting. Envelope-følgeren som vi vanligvis bruker på IRCAM-maskinen fungerer på denne måten.

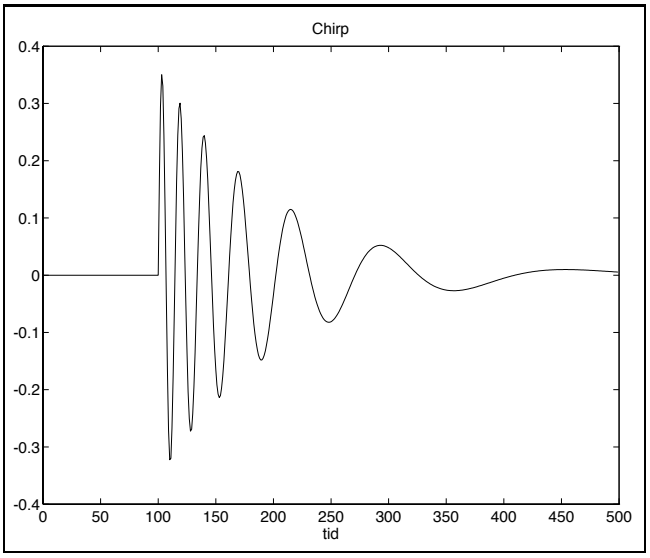
Integrasjonstiden T bestemmer hvor nøye envelopen skal tracke lyden. Vi ønsker ikke at trackingen skal være *for* nøyaktig, for da kan vi risikere å begynne å følge selve kurveformen ved lavfrekvente signaler. På den annen side må T være såpass liten at transienter ikke dempes for mye. Dette er en vanskelig balansegang, og T bør helst settes manuelt ettersom hvaslags lyd man har med å gjøre.

Figurene 5.9-5.11 viser tre eksempler på RMS-måling med løpende integrasjon, med ulike T. Testlyden er en sinustone med en skarp attack (figur 5.8). Tonen dør så ut samtidig med at frekvensen senkes. [4] Legg merke til hvordan transientresponsen, som observeres i begynnelsen av signalet, blir dårligere ettersom vi øker T. Men samtidig dempes selve signalet bedre ved de lave frekvensene på slutten. $T = 32$ er muligens et brukbart kompromiss.

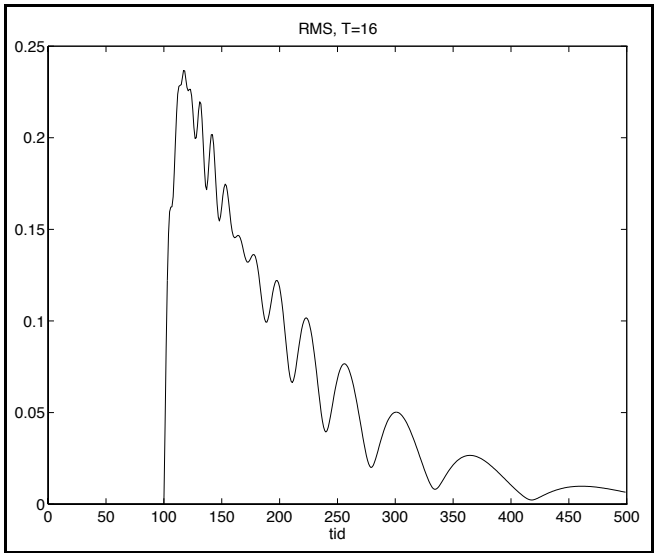
Vi kan prøve etpar andre metoder også. Hvis vi velger ut den *maksimale* verdien i de siste 64 samples, får vi resultatet i figur 5.12.

En annen teknikk er *peak-deteksjon*, det vil si at vi forsøker å finne toppene i kurven og trekker linjer mellom disse (figur 5.13).

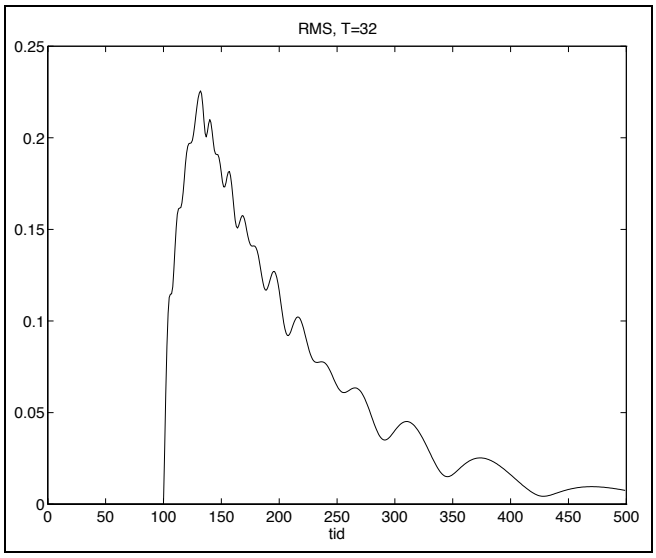
[4] Dette er et mye brukt test-signal, og kalles en *chirp*



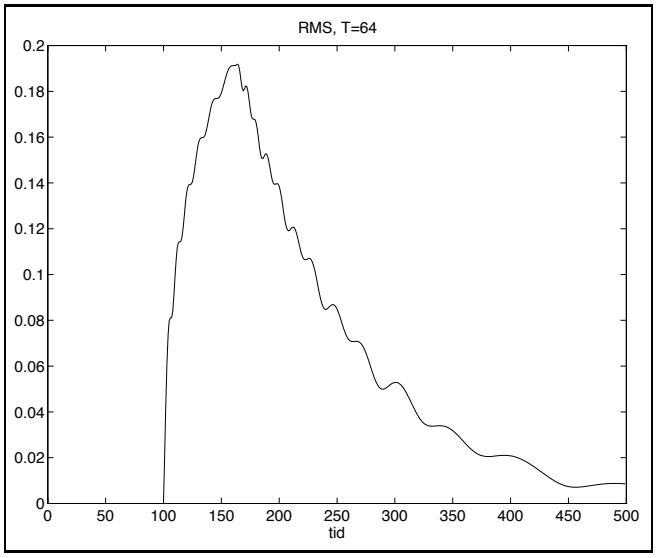
Figur 5.8: Testlyd (chirp)



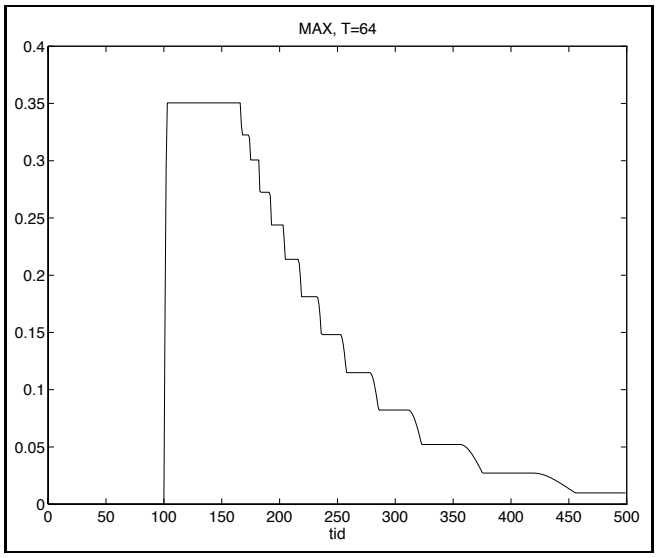
Figur 5.9: Glatting over 16 samples



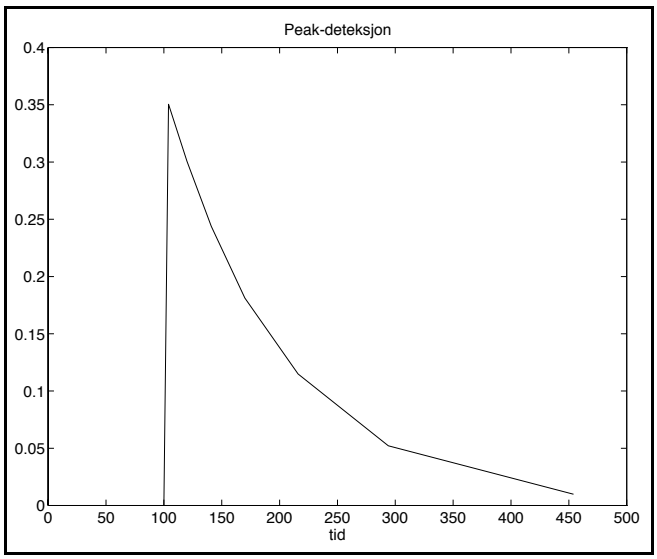
Figur 5.10: Glatting over 32 samples



Figur 5.11: Glatting over 64 samples



Figur 5.12: Envelope-tracking med max-filter



Figur 5.13: Peak-deteksjon

Akkurat for denne lyden fikk vi her et glimrende resultat, men generelt vil denne metoden gå i vasken når vi begynner å få inn overtoner som lager "småtopper" mellom hovedtoppene. Det finnes imidlertid endel triks for å forsøke å overse disse.

5.1.8 Pitch-følging

En annen viktig analyse-teknikk er *pitch-følging* eller F_0-*analyse*, der vi forsøker å plukke ut grunntonefrekvensen i et signal. Dette forutsetter naturligvis at vi har å gjøre med en monofon lyd med en veldefinert grunntone.

Å finne frekvensen til en grunntone er en meget komplisert affære, og det er ikke klart hvordan øret og hjernen løser denne oppgaven. Et stort antall metoder er pønsket ut for pitch-følging, men ingen av dem er ufeilbarlige. En vanlig feil er at analysatoren plukker ut frekvensen til en overtone, slik at vi kan få f.eks. en pitch som ligger en oktav for høyt.

En klassisk teknikk er peak-deteksjon, der vi teller antall bølgetopper i sekundet. For å unngå å telle små-topper som stammer fra overtoner, kan vi gjøre en kraftig lavpassfiltrering først. Dette lavpassfilteret bør følge grunntonefrekvensen gjennom en feedback- mekanisme.

En liknende metode er å telle nullgjennomganger, dvs. de punktene der kurven går fra negative til positive verdier. Igjen vil overtoner kunne lage problemer, og vi bør lavpassfiltrere signalet først.

En av de smartere metodene går ut på å benytte et *kamfilter* som slipper gjennom en viss grunntonefrekvens og harmoniske av denne, men demper alle andre frekvenser. Et slikt filter er lett å lage. Grunntonefrekvensen varieres så inntil vi finner et maksimum i signalstyrken ut fra filteret. Vi kan da regne med at toppene i filterkarakteristikken har truffet deltonene i det opprinnelige signalet. Ettersom pitch varierer, passer vi på å oppdatere filteret slik at signalstyrken forblir størst mulig. En slik algoritme kan bli lurt ved at den stabiliserer seg på en *subharmonisk*. Hvis vi har en grunntonefrekvens på 300 Hz, kan vi risikere at analysatoren finner et maksimum for 100 Hz, fordi topp nr. 3, 6, 9 etc. i kamfilteret treffer deltonene på 300 Hz, 600 Hz, 900 Hz etc. Man bør derfor forsøke å finne den *største* frekvensen som gir maksimum.

Det finnes også metoder som arbeider i frekvensdomenet, ved at man gjør en spektralanalyse og finner toppene i spekteret. Grunntonefrekvensen bestemmes så som største felles faktor av disse toppene.

5.2 Fysisk modellering

Fysikerne begynner etterhvert å få ganske god oversikt over hvilke prosesser som foregår i vibrerende musikkinstrumenter. Vi kan lage matematiske modeller for dette, og implementere dem på datamaskinen. Vi *simulerer* således de fysiske prosessene i musikkinstrumentet, og kan på den måten produsere lyd.

Denne syntesemetodikken har mange tiltalende sider. For det første kan man produsere lyd som låter svært naturlig og ekte. For det andre kan en musiker kommunisere med datamaskinen gjennom en fysisk begrepsverden, og f.eks. bygge opp et musikkinstrument på skjermen ved hjelp av objekter som strenger, resonanskasser, membraner, buer, hammere, plektre, rør etc. Man kan benytte hvilke byggematerialer man vil, og lage instrumenter av alle kategorier og størrelser. Det er også mulig å lage gradvise overganger fra ett instrument til et annet ("morfing"). Syntetiske instrumenter basert på fysisk modellering vil også reagere på en ordentlig

måte på forskjellige spilleteknikker. Hvis man f.eks. presser hardt på en virtuell fiolinbue, kan man forvente at modellen gir en styggere lyd på samme måte som en ekte fiolin gjør det.

Mange teknikker for fysisk modellering er svært beregningskrevende. Ofte dreier det seg om å løse såkalte *partielle differensiallikninger* numerisk. *Bølgelikningen*, som beskriver hvordan bølger forplanter seg, og *Navier-Stokes-likningen*, som beskriver turbulent luftstrømning, er to slike likninger som dukker opp i musikalske anvendelser. Å løse partielle differensiallikninger koker gjerne ned til å løse lineære likningssystemer med mange tusen likninger og tilsvarende antall ukjente. Dette tar tid.

I de senere år er det imidlertid kommet både maskiner som er raske nok til å gjøre slike ting på en rimelig tid, og nye, mer effektive algoritmer. Fysisk modellering står nå antagelig foran et gjennombrudd både i eksperimentell og kommersiell musikk.

5.2.1 En enkel fiolin-modell

La oss se på et enkelt eksempel hvor vi prøver å etterlikne en fiolinlyd ut fra fysiske betraktninger. I heftet om akustikk så vi at vi kan anvende en eksitator-resonator-modell på fiolinen. Eksitator består av en sagtann-liknende kurve, som stammer fra "slip and grip"-interaksjonen mellom bue og streng. En slik sagtann kan vi enkelt generere i en analog synthesizer eller i datamaskinen.

Resonator består av fiolinkassen og andre mekaniske elementer som filtrerer sagtannkurven fra strengen. Frekvensresponsen til fiolinkassen kan måles i laboratoriet, eller vi kan prøve å regne den ut på grunnlag av kassens form (det er ikke lett!). Vi lager oss så et filter med tilsvarende respons, og kjører sagtannkurven gjennom dette.

Dette er en enkel modell, der vi har utelatt et utall av subtile fysiske effekter. For eksempel vil ikke eksitator gi en ren sagtann-kurve, men en komplisert kurve som bl.a. avhenger av buetrykket.

Et annet problem er at da vi konstruerte resonatoren, tok vi utgangspunkt i fysiske målinger på en virkelig fiolin. Dette gjør modellen lite fleksibel.

5.2.2 Karplus-Strong-algoritmen

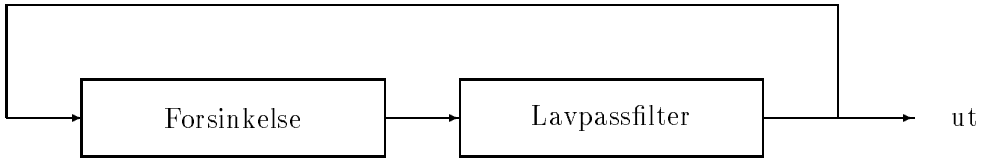
"The Karplus-Strong Plucked String Algorithm" er en spesialisert metode for å syntetisere gitar-liknende lyder. Denne algoritmen er kanskje lite fleksibel, men den gir glitrende god lyd og er ekstremt beregningseffektiv.

La oss si at vi sitter på et fast sted på en gitarstreng og observerer hva som farer forbi. På dette punktet har vi også plassert en mikrofon hvor vi plukker opp strengens utslag.

Når strengen klimpres på et punkt et stykke unna oss, vil strengens deformasjon forplante seg utover til begge sider. Vi kan regne med at denne deformasjonen har en nokså tilfeldig form, med et rikt spektrum. På veien mot oss vil imidlertid de høyeste frekvenskomponentene dempes mer enn de laveste. Dette har å gjøre med hvordan friksjonen behandler ulike frekvenser forskjellig. Strengen fungerer altså som et *lavpassfilter*.

Denne noe lavpassfiltrerte kurveformen farer forbi oss, og blir samtidig plukket opp av mikrofonen. Bølgen farer så videre mot enden av strengen, blir reflektert, lavpassfiltrert en gang til på veien, og kommer tilbake til oss. Slik pendler bølgebevegelsen fram og tilbake, samtidig med at den filtreres stadig mer. Bevegelsen dør da etterhvert ut, og de høyeste frekvensene forsvinner først.

La oss prøve å lage en modell for dette:



Forsøk å se sammenhengen mellom forklaringen over og hvordan signalet går i modellen!

Forsinkelsen implementeres som en digital delay. Grunntonefrekvensen har å gjøre med lengden på strengen, og dermed hvor lang tid det tar for bølgen å komme tilbake til vår tapping. Lengden på forsinkelsen vil dermed bestemme grunntonefrekvensen: 1 millisek vil gi 1000 Hz, 2 millisek 500 Hz etc.

Lavpassfilteret kan være av enkleste type. Det er mer enn godt nok å simpelthen ta gjennomsnittet av nåværende og foregående sample. Av dette ser vi at Karplus-Strong-algoritmen nesten ikke krever regnekapasitet i det hele tatt.

Vi ser også at KS-algoritmen i sin enkleste form kan implementeres ved hjelp av en vanlig digitaldelay (effektboks) som er innstilt på en meget liten forsinkelsestid, et lavpassfilter (f.eks. tonekontrollen på en mikser) og en feedback.

Vi starter en note ved å fylle digitaldelayen med en tilfeldig sekvens (hvit støy). Tilfeldigheten gjør at vi får en noe annerledes klang for hvert anslag, og dette øker naturligheten. Denne snutten vil så fare rundt i systemet, samtidig med at den lavpassfiltreres igjen og igjen. Vi ender etterhvert opp med bare grunntonen (en sinus), men den forsvinner også til slutt. Noen eksempler er vist i fig. 5.14-5.16, der vi har brukt identiske parametre hver gang. Tilfeldigheten lager likevel variasjon i resultatet.

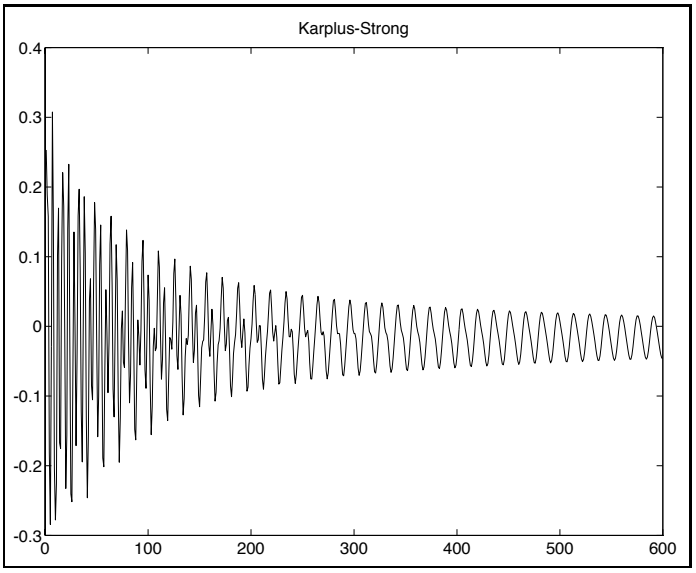
Ved å endre gain i feedback-løkken, kan vi styre hvor hurtig lyden skal dø ut. Vi kan si at vi da regulerer friksjonen i strengen.

Karplus-Strong-algoritmen kan utvides på mange måter. Vi kan benytte forskjellige slags filtre i feedback-løkken, filtrere sluttresultatet for å simulere gitarkasse, lage el-gitar-effekter som feedback og fuzz osv. En mykere attack kan oppnås ved å lavpassfiltrere hvitstøyen en gang før selve syntesen starter. Det går også an å putte naturlig lyd inn istedenfor hvit støy (dette er gjort i "Smalltalk" av Paul Lansky).

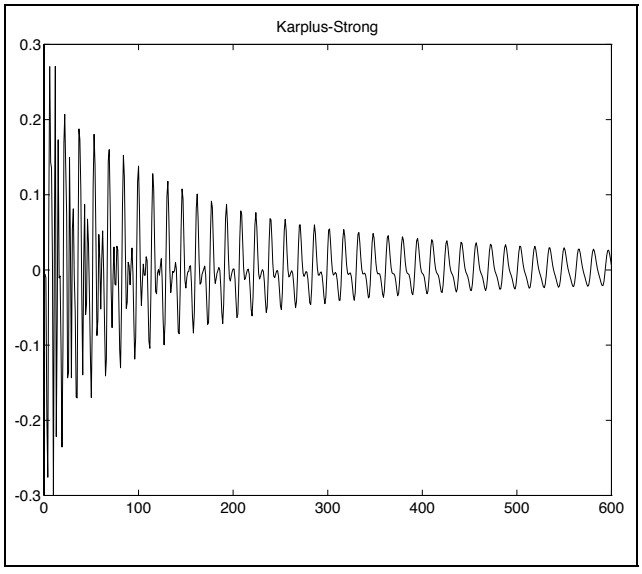
I det hele tatt er dette en morsom metode å arbeide med. En god demonstrasjon av Karplus-Strong-algoritmen er stykket "Silicon Valley Breakdown" av David Jaffe. Anders Vinjar i Norge har laget musikk som benytter utvidelser av Karplus-Strong-algoritmen. En implementasjon av Karplus-Strong finnes bl.a. i C-Sound.

5.2.3 Waveguides

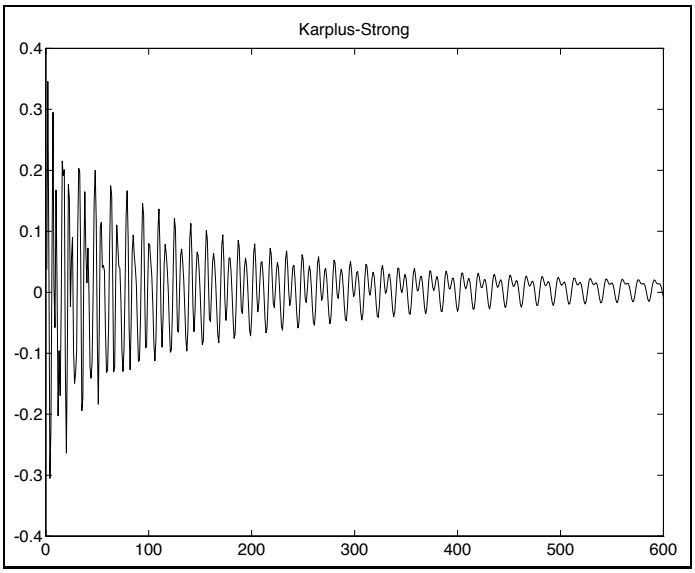
Karplus-Strong-algoritmen er et spesialtilfelle av en mye større klasse av syntesemetoder basert på fysisk modellering, nemlig syntese ved *waveguides*. Dette er en helt ny metodikk, som er utviklet av Julius O. Smith III ved CCRMA, Stanford. Som i Karplus-Strong- algoritmen,



Figur 5.14:



Figur 5.15:



Figur 5.16:

forsøker man å sette opp digitale forsinkelser som etterlikner mediet som lydbølgene forplanter seg i. Langs disse "bølge-lederne" kan man sette inn elementer som modifiserer signalet på sin vei, f.eks. lavpassfiltre. På den måten kan man løse bølgelikningen på en effektiv og fleksibel måte.

Yamaha har nå satt i produksjon en synthesizer som benytter denne metoden. Den har bare to stemmer og koster 6000 dollar, men vi må regne med at dette bare er begynnelsen på en viktig kommersiell teknologi.

Ved CCRMA har man også utviklet et program for syntese av tale og sang, kalt SPASM, som bruker waveguides. Kvaliteten og fleksibiliteten til dette systemet skal være meget god.

5.2.4 Modal syntese

Det går også an å tenke fysisk modellering mer i frekvensdomenet: Hvilke frekvenser vil kunne finnes i det fysiske systemet, og hvordan vil de utvikle seg? Siden vi da ser på egenskapene til de ulike svingningsmodi som er tilstede, kan vi kalle dette for *modal syntese*.

Programmet *Modalys*, som delvis er utviklet ved IRCAM, benytter modal syntese i et system som lar brukeren bygge opp sine egne instrumenter ved hjelp av forhåndsdefinerte virtuelle fysiske objekter.

I Norge har *Terje Syversen* utviklet teknikker og programvare for modal syntese. Han har også planlagt hardware som skal kunne utføre dette i sann tid. [5]

5.3 Lydbehandling og syntese i frekvensdomenet

Vi har hittil sett mest på lydbehandlingsteknikker som først og fremst betraktes i tidsdomenet. I dette kapittelet skal vi se på metoder der vi behandler lydens spektrum, slik at vi beveger oss i frekvensdomenet. Dette er en noe løs klassifisering, siden enhver operasjon i tidsdomenet gir

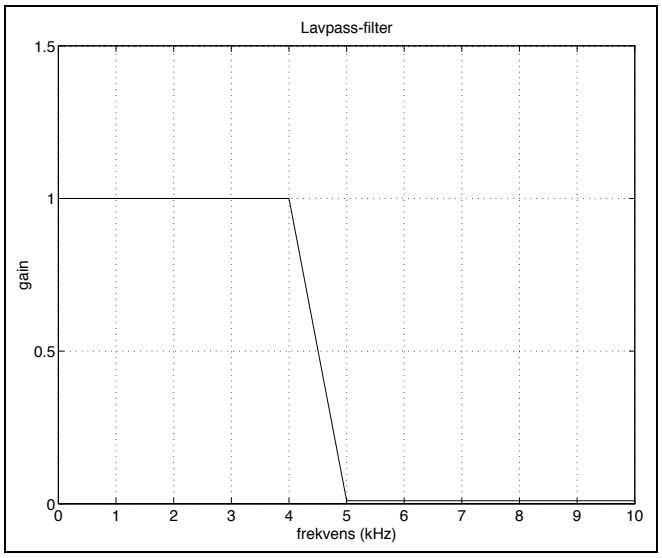
[5] Terje Syversen: Syntese av Musikalsk Lyd, hovedoppgave ved Institutt for Fysikk, Universitetet i Oslo

en endring i frekvensdomenet og omvendt. Det er, som vi har sett, bare snakk om to forskjellige måter å representere lydsignalet på.

5.3.1 Generelt om filtre

Et *filter* er en innretning som endrer et spektrum på en slik måte at det ikke kommer til nye frekvenskomponenter. For å være litt mer konkret, kan vi si at et filter generelt *demper* visse frekvenser og *forsterker* andre. Tradisjonelt betraktes følgende klasser av filterkarakteristikker som særlig viktige:

- *Lavpass*, der alle frekvenser under en viss frekvens (*cut-off* eller *klippefrekvens*) beholdes, mens høyere frekvenskomponenter dempes mest mulig (figur 5.17).



Figur 5.17: Lavpassfilter

  Et lavpassfilter kan dermed f.eks. brukes til å dempe støy på høye frekvenser. Diskantknappen på stereoanlegget kontrollerer et lavpassfilter.

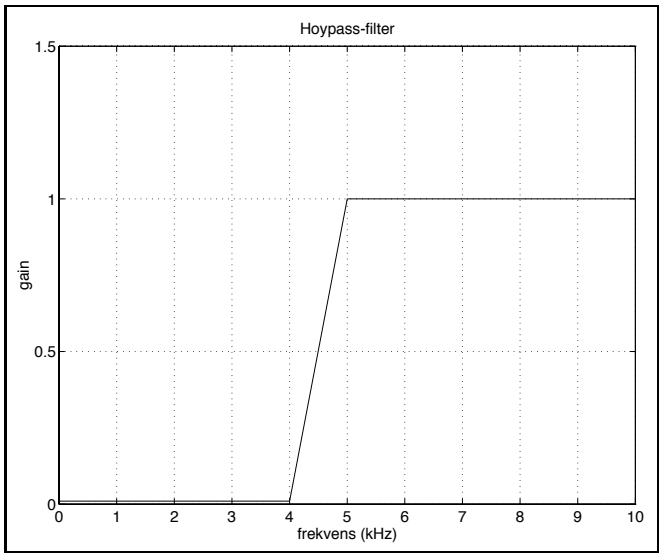
- *Høypass*, der alle frekvenser over klippefrekvensen beholdes, mens lavere frekvenskomponenter dempes mest mulig (figur 5.18).

  Bass-knappen på stereoanlegget kontrollerer et høypassfilter. Et høypassfilter og et lavpassfilter brukes i to-veis høyttalere for å styre lyd til henholdsvis diskant- og basselementet.

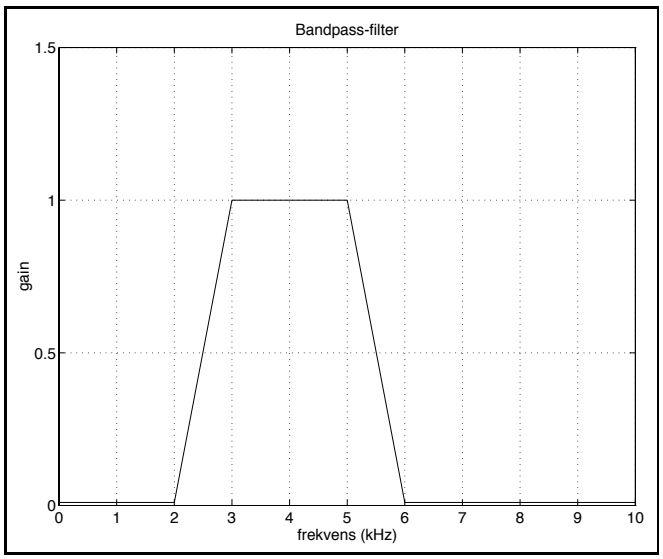
- *Båndpass*, der alle frekvenser innenfor et visst bånd beholdes, mens frekvenskomponenter utenfor båndet dempes mest mulig (figur 5.19).

  Et båndpassfilter kan karakteriseres ved en *senterfrekvens* (her 4 kHz) og en *båndbredde*. Ingeniører snakker gjerne om *Q-faktor* (godhetsfaktor), som er definert som senterfrekvens delt på båndbredde. Stor Q betyr dermed liten båndbredde.

  Et båndpassfilter med liten båndbredde kan "sveipes" over en lyd slik at bare en og en deltone slipper igjennom filteret av gangen.

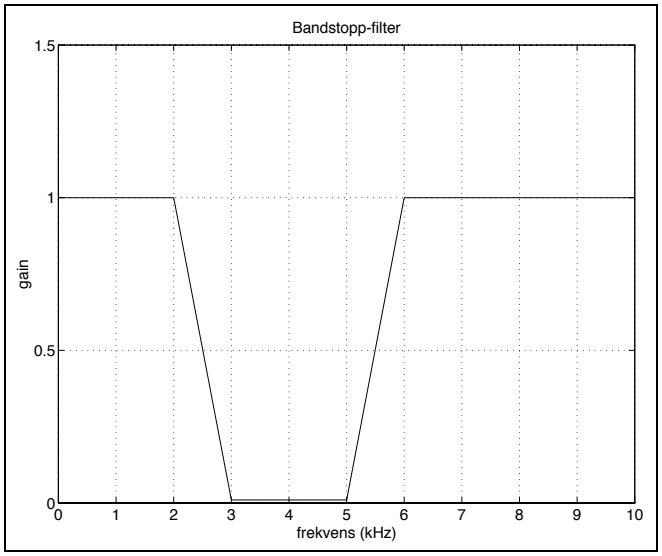


Figur 5.18: Høypassfilter



Figur 5.19: Båndpassfilter

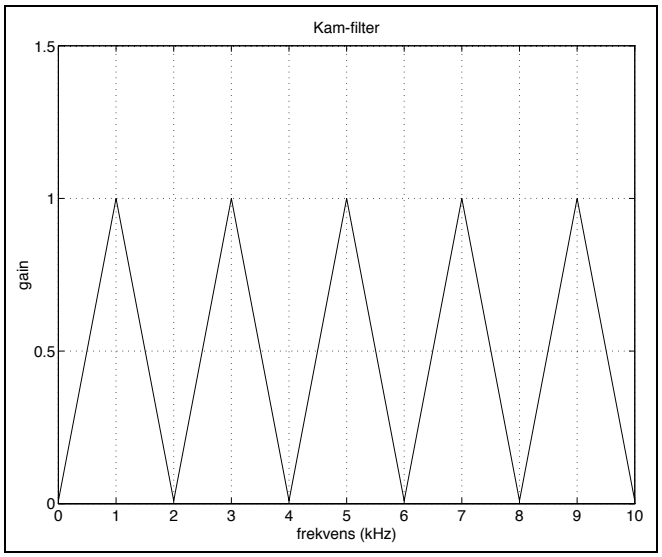
- *Båndstopp* eller *notch*, der alle frekvenser utenfor et visst bånd beholdes, mens frekvenskomponenter innenfor båndet dempes mest mulig (figur 5.20).



Figur 5.20: Båndstoppfilter

Et båndstoppfilter med liten båndbredde og senterfrekvens 50 Hz er fint til å fjerne nettbrum med.

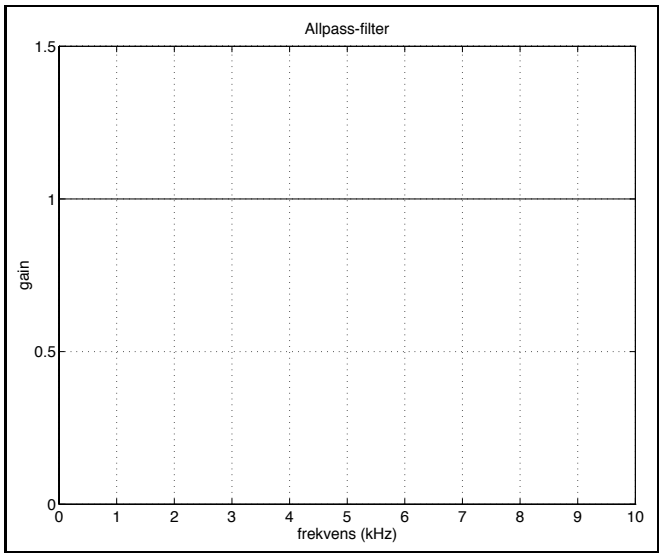
- *Kamfilter*, som vekselvis stanser og stopper frekvensene med jevne mellomrom oppover i spekteret (figur 5.21).



Figur 5.21: Kamfilter

Et kamfilter er fint til å slippe igjennom en grunntone med overtoner, men dempe alt annet. Vi så en anvendelse for dette i avsnittet om pitch-følging.

- *Allpass*, som slipper igjennom alle frekvenser uhindret (figur 5.22).



Figur 5.22: Allpassfilter

Hva i all verden er vitsen med det? Et allpassfilter gjør kanskje ingenting med *styrken* til de ulike deltonene, men det kan ha stor effekt på *fasen*. Dette betyr at man kan konstruere allpassfiltre som slipper lyden uhindret igjennom, bortsett fra at ulike frekvenser forsinkes ulikt og derfor kommer ut til forskjellig tid. Dette kalles *dispersjon*, og er analogt til spredningen av ulike frekvenser i lys gjennom et prisme.

Det er teoretisk umulig å konstruere et filter hvor flankene er helt loddrette. Det vil alltid finnes et overgangsområde mellom passbåndet og stoppbåndet. Generelt må et filter gjøres mer komplisert jo steilere flanker vi ønsker. De enkleste filtrene (av *første orden*) er lavpass- og høypassfiltre som demper 6 dB pr. oktav. Mer kompliserte filtre av annen orden kan dempe 12 dB pr. oktav. Svære filtre, ofte kalt *brick-wall-filtre*, kan dempe kanskje oppimot 100 dB pr. oktav, men da begynner man å få problemer med bl.a. stor forsinkelse gjennom filteret.

Vi kan lage mer kompliserte karakteristikker og brattere flanker ved å koble grunntypene over i serie (da vil karakteristikkene multipliseres) eller parallell (da vil karakteristikkene adderes).

5.3.2 Digital filtrering med differenslikninger

Filtrering kan implementeres meget effektivt på en datamaskin ved hjelp av *differenslikninger*. En slik likning angir ett nytt utgangs-sample som en vektet sum av tidligere inngangs-samples og utgangs-samples. Hvis x er inngangs-samples og y er utgangs-samples, kan vi skrive

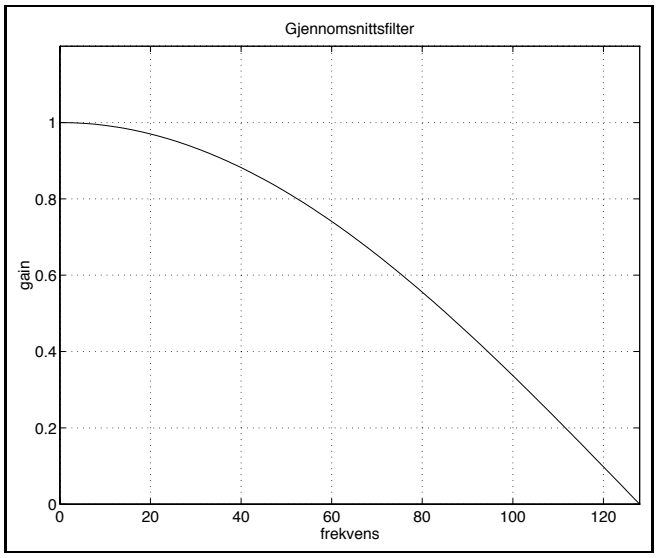
$$\begin{aligned} y_n = a_1 x_n &+ a_2 x_{n-1} + a_3 x_{n-2} + \cdots \\ &+ b_2 y_{n-1} + b_3 y_{n-2} + \cdots \end{aligned}$$

der n er nummeret til "nåværende" sample.

La oss se på et enkelt spesialtilfelle, der vi bare bruker tidligere inngangs-samples. Vi tar simpelthen gjennomsnittet av de to forrige inngangs-samplene, og presenterer dette som et nytt utgangs-sample:

$$y_n = 0.5 \cdot x_n + 0.5 \cdot x_{n-1}$$

Siden vi tar et løpende gjennomsnitt, vil hurtige variasjoner i sampleverdiene i en viss utstrekning bli glattet ut. Dette tilsvarer at høyfrekvente komponenter blir dempet. Det ser dermed ut til at vi får et slags lavpassfilter. Sender vi således en DC-spenning gjennom filteret, hvilket tilsvarer en frekvens på 0 Hz, vil denne passere uendret gjennom filteret. Vekselvis positive og negative verdier med like stor absoluttverdi, hvilket tilsvarer en komponent på Nyquist-frekvensen, vil bli dempet fullstendig. Filterkarakteristikken er vist i figur 5.23 (Nyquist-frekvensen er her 128 Hz).



Figur 5.23: Gjennomsnittsfilter

Det kan overlates som en oppgave til leseren å finne ut hvorfor vi får et høypassfilter hvis vi bytter ut pluss-tegnet i likningen over med et minus-tegn.

Hvis vi i tillegg tar med tidligere utgangs-samples i differenslikningen, introduserer vi en feedback i filteret. Vi får da et *rekursivt* filter.

Det er en stor vitenskap å bestemme koeffisientene i differenslikningen slik at vi får en ønsket filterkarakteristikk i frekvensdomenet. Vi kan ikke gå nærmere inn på dette. I praksis vil man benytte oppslagsverk og tabeller der man kan finne koeffisienter for forskjellige slags filtre, der ting som senterfrekvens og båndbredde inngår som parametre.

5.3.3 FFT, spektralanalyse

Hvis vi har lyden representert i frekvensdomenet, kan vi lage filtre ved å behandle spekteret direkte. Vi kan gå inn og klippe bort frekvenskomponenter innenfor visse områder, og forsterke i andre områder. Det er også mange andre morsomme operasjoner vi kan utføre i frekvensdomenet.

For å overføre en samplet lyd til frekvensdomenet, må vi gjøre en *frekvensanalyse* eller *Fourier-transformasjon*. Dette er i utgangspunktet en meget tung regneoperasjon. Det var derfor en viktig oppdagelse som ble gjort av Cooley og Tukey på 1960-tallet, da de fant en

svært effektiv algoritme for Fourier-transformasjon. Etterhvert har det kommet flere liknende algoritmer, som nå går under fellesbetegnelsen *Fast Fourier Transform* eller *FFT*.

FFT er en nokså innfløkt og smart algoritme, men for brukeren er resultatet greit nok: Man putter en sample-sekvens inn, og får et spektrum ut. Man kan gjerne gjøre en sammenhengende FFT på en hel lydfil, men da vil utviklingen over tid inngå i spektral- koeffisientene på en utydelig måte. Den svenske komponisten Paul Pignon har arbeidet med slik "mammut-FFT", hvor han transformerer en lydfil over i frekvensdomenet og spiller av *spekteret* som samples direkte (!). Denne sammenblandingen av tids- og frekvensdomenet er verdt å høre på, forunderlig nok.

En analysemetode som bedre følger hvordan øret og hjernen oppfatter lyd, er *korttids-FFT*. Man tar da en spektralanalyse av korte biter av lyden av gangen, slik at man får en rekke påfølgende øyeblikksbilder av spekteret som varierer over tid. Vi må da velge hvor mange *punkter* vi vil ha i FFT'en, dvs. hvor mange samples vi vil analysere av gangen. Antall punkter betegnes oftest med bokstaven N. De fleste FFT-algoritmer fungerer mest effektivt hvis N er en potens av 2. Typiske verdier som brukes er 256 (2^8), 512 (2^9), 1024 (2^{10}) og 2048 (2^{11}). Ut fra spektralanalysen vil vi da få $N/2$ verdier for amplityde og fase for de $N/2$ frekvensområdene (*kanalene*) som er jevnt spredt fra 0 Hz til Nyquist-frekvensen. Vi ser da at vi får N tall ut til sammen, og vi puttet N tall inn. Spektralanalysen har hverken skapt eller fjernet informasjon.

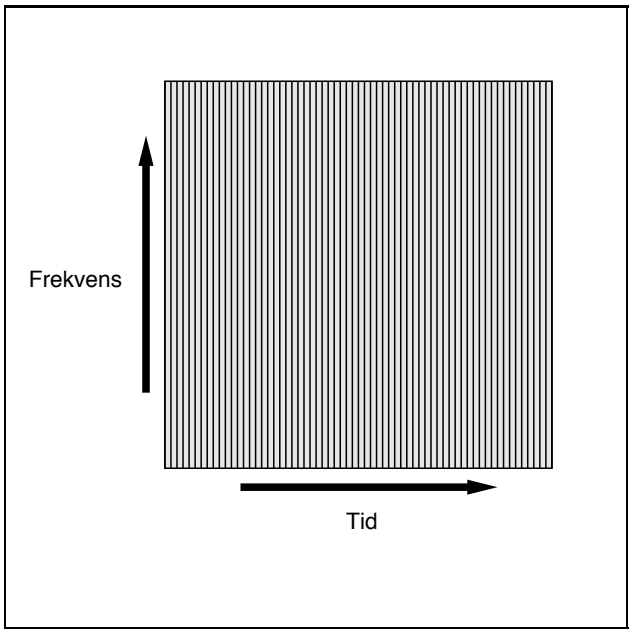
Vi ser at jo flere punkter vi har i FFT'en, jo større oppløsning får vi i frekvensdomenet (vi får flere "samples" i spekteret). Med en samplingfrekvens på 32 kHz, vil $N = 512$ gi en oppløsning på 64 Hz, mens $N = 1024$ vil gi en oppløsning på 32 Hz. Men når vi øker N må vi analysere stadig lengre lydbiter, og det gjør at vi får en dårligere oppløsning i tid. $F_s = 32$ kHz og $N = 512$ vil gi 64 spektre i sekundet (hver lydbit blir 16 ms lang), mens $N = 1024$ vil gi 32 spektre i sekundet, så hver lydbit blir 31 ms lang.

Vi får dermed en avveining mellom presisjon i frekvens og presisjon i tid: Jo høyere nøyaktighet vi måler frekvensen med, jo lavere nøyaktighet må vi nødvendigvis måle tiden med. I figurene under er det vist hvordan vi kan dele opp frekvens-tid-planet på ulike måter ved å justere N. Vi kan ikke plassere en hendelse i tid og frekvens med større nøyaktighet enn en boks. Disse boksene har alltid samme areal, slik at hvis vi øker nøyaktigheten i frekvens (boksene blir lavere) minsker nøyaktigheten i tid (boksene blir bredere)[6].

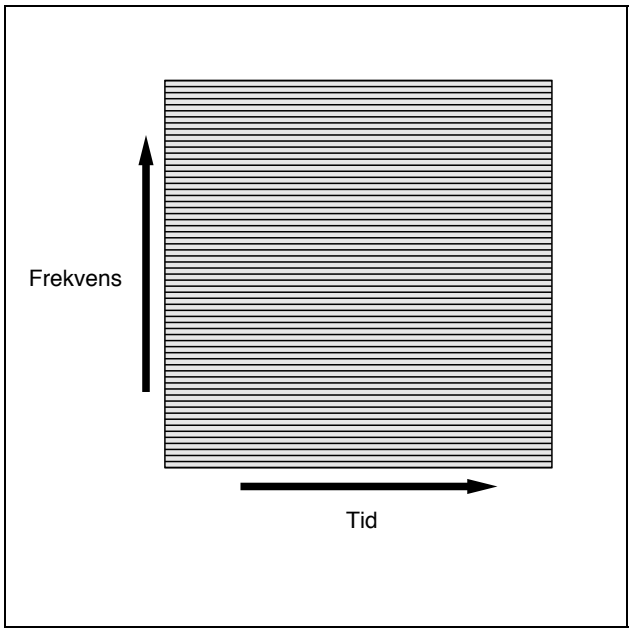
Bestemmelse av N bør gjøres i hvert enkelt tilfelle, avhengig av lyden vi arbeider med. Et stort antall tette frekvenskomponenter kombinert med lave krav til presisjon i tid (langsom dynamikk), vil typisk kreve store N. Færre frekvenskomponenter kombinert med hurtige attacks krever typisk små N. Vi ønsker generelt å sette N så liten som mulig for å få best mulig temporal oppløsning, men ikke så liten at frekvensområdene gaper over mer enn en deltone i lyden.

Vi vil imidlertid aldri *miste* informasjon i en FFT. Lyden kan derfor rekonstrueres perfekt uansett N, så lenge vi ikke har gjort noen endringer i spekteret.

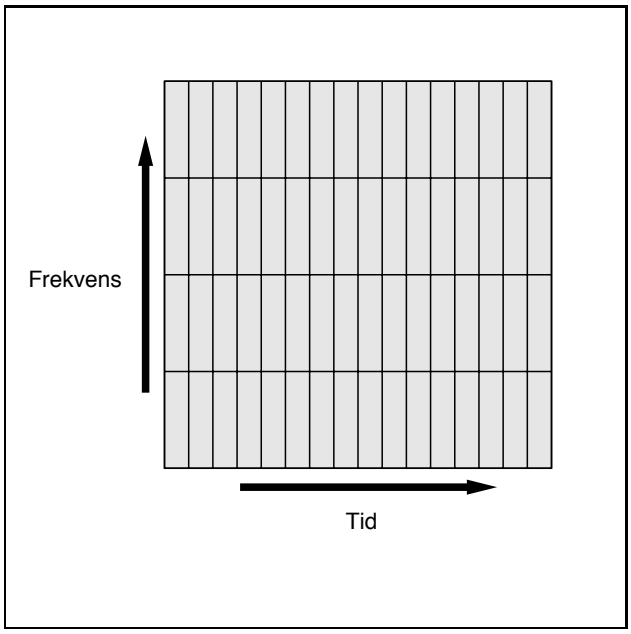
[6] I kvantemekanikken er dette kjent som *Heisenbergs usikkerhetsrelasjon*. Hvis vi betrakter partikler som bølger, vil frekvensen avhenge av partikkelens masse etter likningen $f = mc^2/h$ der f er frekvens, m er masse, c er lyshastigheten og h er Planck's konstant. Et elektron har meget liten masse og dermed meget liten frekvens. For å bestemme massen (les: frekvensen) må vi dermed betrakte elektronet over et langt tidsrom, slik vi så over. Siden elektronet beveger seg i løpet av dette tidsrommet, vil bestemmelse av posisjon og fart bli upresis. Vi kan derfor ikke angi en bestemt posisjon for elektronet, men bare plassere det innenfor et område som vi kaller en *elektronsky*. Usikkerhetsrelasjonen er gjenstand for mye mystisering og tåkelegging (New Age, Frithjof Capra etc.), men som vi ser dreier det seg om en høyst prosaisk og grei sammenheng.



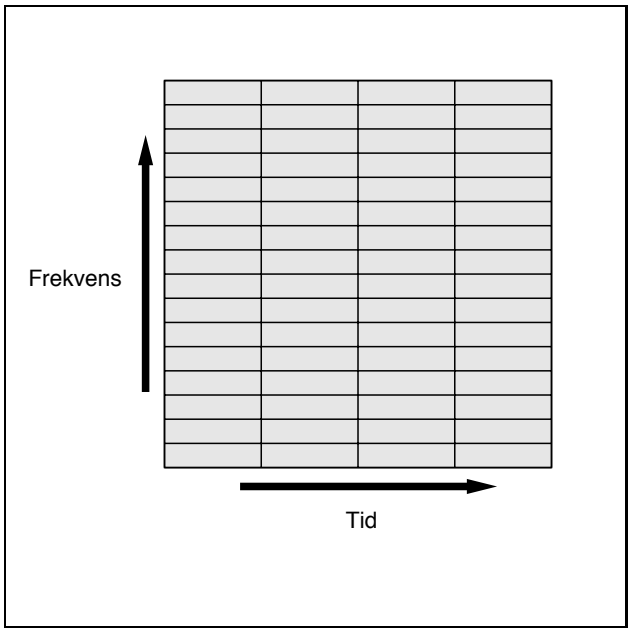
Figur 5.24: $N = 1$ gir best mulig presisjon i tid, men gir ingen frekvensinformasjon



Figur 5.25: $N = \infty$ gir best mulig presisjon i frekvens, men gir ingen plassering i tid



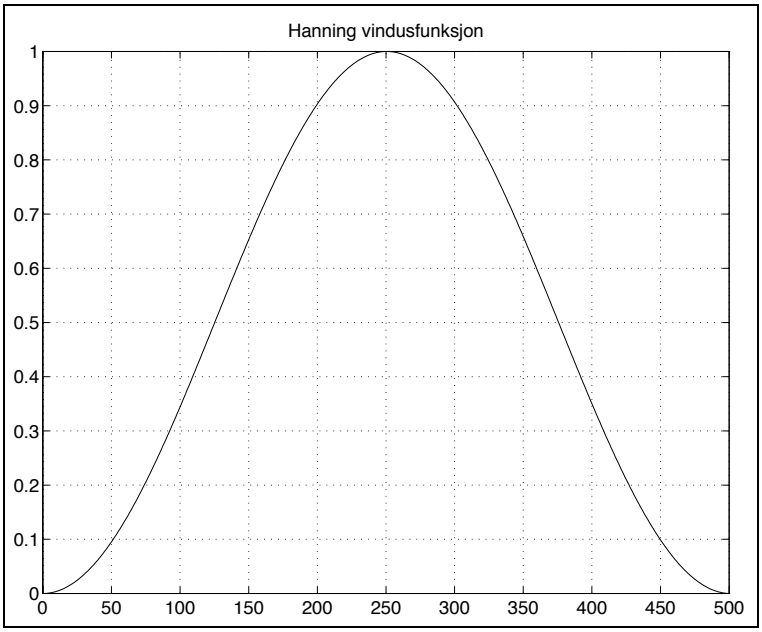
Figur 5.26: Liten N gir god presisjon i tid, men dårlig presisjon i frekvens



Figur 5.27: Stor N gir god presisjon i frekvens, men dårlig presisjon i tid

5.3.4 Vindusfunksjoner, overlapping

De små områdene vi analyserer av gangen i en korttids-FFT kalles *vinduer*. Av tekniske grunner (undertrykking av sidebånd etc.) vil man vanligvis legge på en liten envelope på hvert av disse vinduene før man utfører FFT'en. En slik envelope kalles en *vindusfunksjon*, og det finnes mange ulike slike funksjoner, med navn som Hamming, Hanning, Kaiser, Bartlett etc. Hver av disse har sine fordeler og ulemper.



Figur 5.28: Hanning vindusfunksjon

For ikke å miste informasjon i overgangen mellom vinduene, er det nødvendig å la dem *overlappe*. Denne overlappingen er gjerne ganske drastisk, og derfor vil vi få ut mange flere spektre pr. sekund enn vi så på over (uten at oppløsningen i tid dermed blir noe bedre, vi får bare en slags interpolering). Datamengden bli derfor stor ut fra en korttids-FFT.

5.3.5 Ting man kan gjøre med et spektrum

Når vi nå har representert lyden i frekvensdomenet, åpner det seg en verden av morsomme lydbehandlingsteknikker. Vi har allerede nevnt hvor lett det er å lage filtre. Noen andre eksempler:

- Hvis vi adderer en fast verdi til hver frekvens, vil vi få et *spektralskift* som vil introdusere inharmonisitet. Dette tilsvarer på et vis en ringmodulasjon, bortsett fra at vi får ett sidebånd, ikke to.

- Hvis vi multipliserer hver frekvens med en fast verdi, får vi et pitch-skift uten medfølgende hastighetsendring. Dette er en metode for harmonizing som gir meget god kvalitet.

- Vi kan snu spekteret opp-ned, slik at grunntonen blir den høyeste harmoniske i spekteret og omvendt, eller permutere deltonene på andre måter.

- Vi kan sortere deltonene etter amplityde, og beholde bare de kraftigste.
- Vi kan filtrere *spekteret*, enten langs frekvensaksen (vil framheve eller dempe bratte flanker i hvert korttidsspektrum) eller langs tidsaksen (vil framheve eller dempe endringer i klangen over tid).
- Vi kan analysere *to* lyder, og f.eks. multiplisere spektrene med hverandre eller lage glidende overganger mellom dem på forskjellige måter ("morfing").
- Vi kan tvinge deltonene henimot visse frekvenser, f.eks. en musikalsk skala.

Her er det fritt fram for fantasien! Parametrene til disse metodene kan selvsagt variere over tid.

5.3.6 IFFT, resyntese

Etter at vi har vrengt spekteret på en eller annen fantastisk måte, er vi nødt til å komme oss tilbake til tidsdomenet før vi kan høre lyden. Dette kalles *resyntese*, og er egentlig bare en additiv syntese. Denne resyntesen kan gjøres med den samme algoritmen som FFT, men med et par små endringer som gjør at den virker *baklengs*. Dette heter *invers* FFT (IFFT).

Enkelte ganger kan det være fordelaktig å resyntetisere lyden med "rå makt", dvs. addere et stort antall sinustoner direkte. Dette er langt mer tidkrevende enn IFFT.

Hvis vi strekker oppdateringen av parametrene i resyntesen utover i tid, får vi en time-stretch (kompresjon/dilatasjon) av høy kvalitet uten medfølgende pitch-endring.

5.3.7 Fase-vokoderen

Det ble nevnt at tallene som kommer ut av en FFT er amplityde og fase for den deltonen som ligger innenfor hvert frekvensområde. [7] Men musikere er nok mer opptatt av *frekvens* enn av fase. Derfor benyttes ofte en teknikk som heter *fase-vokoding*, der faseinformasjonen fra FFT'en brukes til å beregne frekvens. Dette baserer seg på at endringen i fase fra analysevindu til analysevindu er proporsjonal med deltonens frekvens. Vi får da ut eksakte frekvenser, ikke bare informasjon om at deltonen ligger innenfor en bestemt kanal.

Musikkprogrammet C-Sound har en fasevokoder, og den finnes også i SoundHack for Mac og flere andre systemer. Fasevokoderen er brukt i mange musikkstykker til mange forskjellige formål.

5.3.8 Lineær-prediktiv koding

Menneskets tale fungerer ved at stemmebåndene produserer et pulstog med et nokså flatt spektrum. Dette pulstoget filtreres videre av hals og munnhule. Forskjellige vokaler tilsvarer forskjellige frekvenskarakteristikker for dette filteret. Et typisk trekk ved disse karakteristikkene er to eller flere tydelige resonanser (formanter).

Ved lineær prediksjon forsøker vi å produsere et filter av differenslikning-typen, med koeffisienter som varierer over tid. Dette filteret skal etterlikne filterkarakteristikken til lyden vi analyserer. Signalet kan siden re-syntetiseres ved å eksitere filteret med et syntetisk pulstog som etterlikner stemmebåndene. Vi kan på den måten kode tale bare gjennom filterkoeffisientene,

[7] Vi håper at det er bare en slik sinusoid komponent i hver kanal, men dette vil ofte ikke slå til (f.eks. ved hvit støy)

som varierer forholdsvis langsomt, og dermed har vi en effektiv metode for data-reduksjon av talesignaler. Lineær-prediktiv koding (LPC) har oppnådd en posisjon innen talesyntese, selv om lydkvaliteten ofte er dårlig (dette bunner i bruk av for få filterkoeffisienter, og stiliseringen av eksitator). Den typiske "robot-stemmen" som var vanlig i tale-synthesizere for noen år siden, var produsert ved hjelp av LPC.

Som vi har sett, kan eksitator-resonator-modellen anvendes på de fleste musikkinstrumenter. LPC begrenser seg derfor ikke til bruk på tale.

LPC kan brukes til å vri lyden på mange rare måter. Vi kan f.eks. variere pitch ved å skru på frekvensen til eksitator. I vanlige harmonizere, og i pitch-skift med FFT, vil formantenes posisjon endre seg. Tale fungerer ikke på den måten, og LPC gir derfor et mer korrekt resultat, uten den typiske "Donald-stemmen".

Vi kan også kjøre andre lyder gjennom filteret vi har generert. På den måten kan vi gjøre kryss-syntese, f.eks. ved at vi simulerer at vi har et symfoniorkester plassert i strupehodet istedenfor stemmebånd. Vi får da et "talende orkester".

Det må innrømmes at LPC ofte ikke låter så bra. Det krever erfaring å utnytte LPC på den beste måten, bestemme hvor mange filterkoeffisienter vi skal ta med etc.

Guruen innenfor musikalske anvendelser av LPC er den amerikanske komponisten Paul Lansky. Musikkprogrammet C-Sound inneholder en implementasjon av LPC.

5.3.9 Filterbanker

Det finnes en annen metode for spektralanalyse som kanskje er lettere å begripe, men som ikke er så beregningseffektiv som FFT, nemlig bruk av en *filterbank*. En filterbank er en bunke med båndpassfiltre koblet i parallell, med senterfrekvenser som er spredt over hele frekvensområdet. Det gjøres en RMS-analyse eller annen type energi-analyse på output fra hvert filter, og dermed får vi informasjon om hvor mye energi vi har innenfor hvert frekvensbånd.

Filterbanker er historisk viktige, fordi de kan implementeres analogt. De fleste enkle spektralanalysatorer av den typen vi finner i lydstudioer, fancy stereoanlegg etc. benytter filterbanker. Den klassiske effektboksen som kalles "vokoder" har en filterbank koblet sammen med en oscillatorbank. Dette gir en enkel form for analyse-resyntese.

I de senere år har filterbanker fått en kraftig renessanse. Man har begynt å sette pris på *kontinuiteten* som filterbanker gir i forhold til den mer opphakkede, vindusbaserte FFT-metodikken. Samtidig har digitale signalprosessorer blitt raske nok til å benytte filterbanker i sann tid. De nye digitale kompakt-kassettene (DCC), og Minidisc, benytter filterbanker som en del av datakompresjonen.

5.4 Granulære metoder

I dette kapittelet skal vi se på en måte å betrakte lyd på som kan virke uvant etter alt snakket om kontinuerlige sinusoider og varierende spektre. Det går an å se på lyd som et aggregat av utallige små "korn" (grains) som er plassert rundt i frekvens og/eller tid. Vi kan syntetisere lyd ved å pøse på med slike korn, vi kan forsøke å finne ut hvilke korn av en valgt type vi må benytte for å bygge opp en lyd vi analyserer, eller vi kan klippe opp en lyd i små biter som vi behandler på ulike måter. Alt dette kan vi kalle *granulære* lydbehandlingsmetoder, etter det latinske ordet *granula* (korn). [8] Hva slags korn vi vil benytte i f.eks. granulær syntese

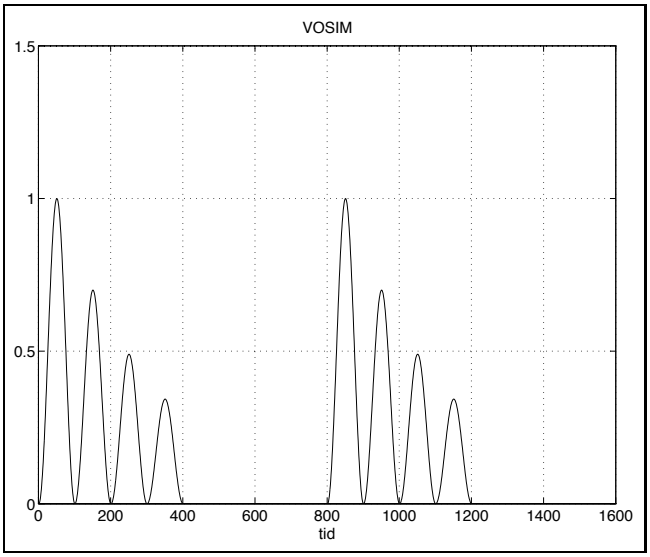
[8] Bergarten *granitt* består av bittesmå korn, og har fått sitt navn av den samme stammen

er igrunnen valgfritt. Akustisk vil det gjerne dreie seg om små "plopp" av type dryppende vannkran eller vibrafon. Disse ploppene kan f.eks. spres ved hjelp av tilfeldighetsgeneratorer i "skyer". Å bygge opp en lyd med slike korn har en parallell i *pointilismen*. Dette var en retning innen impresjonismen der man malte bilder ved å avsette tusener av små fargepunkter på lerretet.

5.4.1 VOSIM-syntese

VOSIM-syntese (for *VOice SIMulation*) er en granulær syntesemetode som opprinnelig ble laget for talesyntese. Kornene har her en spesiell form med parametre som vi kan velge. Hvert korn består av et lite pulstog med N klokkeformede pulser (kvadrert sinus er fint), hver med en pulsbredde w. Etter den første pulsen multipliseres påfølgende pulser med en dempningsfaktor $d < 1$. Hele pulstoget gis en amplityde a, og så gjentas kornene med en periode L som bestemmer grunntonefrekvensen ($f = 1/L$).

Figur 5.29 viser en bit av et VOSIM-signal, med to korn.



Figur 5.29: VOSIM-signal

Ved hjelp av parametrene kan vi styre pitch, amplityde og klangfarge. En enkelt VOSIM-generator lager et spektrum med en enkelt formant. For flere formanter, bruker vi flere VOSIM-generatorer.

VOSIM ble utviklet ved Institutt for Sonologi i Utrecht, Nederland. Metoden har ikke noen stor utbredelse i dag.

5.4.2 FOF-syntese

En *FOF* (Forme Onde Formantique), er et lite korn som består av en sinus med en eksponensiell envelope. På samme måte som i VOSIM, vil syntese med en enkelt FOF-generator gi et spektrum med en enkelt formant. Miksing av flere ulike FOF-generatorer gir et spektrum med flere formanter, som kan brukes til syntese av f.eks. tale og sang.

En FOF kan tolkes som en enkelt impuls fra stemmebåndene, som er filtrert av hals og munnhule. Hvis man trekker pusten inn, kan man klare å lage enkeltstående FOF'er som kan

høres separat [9]. En slik liten kurveform, som er filterets respons på en impuls, kalles naturlig nok filterets *impulsrespons*. Et filter er fullstendig karakterisert ved sin impulsrespons; ja faktisk er frekvenskarakteristikken ikke noe annet enn Fouriertransformasjonen til impulsresponsen. Ved invers Fourier-transformasjon kan vi dermed gå fra frekvensrespons til impulsrespons. Hvis vi sampler denne impulsresponsen, kan vi benytte samplene direkte som filterkoeffisienter i et ikke-rekursivt filter. (Dette avsnittet var en digresjon, men det ville være synd om ordet "impulsrespons" ikke ble nevnt i en tekst om digital lydbehandling).

Programmet *Chant* fra IRCAM benytter FOF-syntese. Det finnes en meget imponerende syntese av "Nattens Dronning" av Mozart som er laget med dette systemet. C-Sound inneholder også en FOF-generator.

I forbindelse med granulær syntese må vi også nevne amerikaneren Barry Truax og svensken Peter Lundén, som begge har laget stykker som illustrerer metoden godt. "Riverrun" av Truax anbefales spesielt.

5.4.3 Granulering med vindusfunksjoner

Vi kan "granulere" en lyd ved å klippe den opp i små biter med en varighet på typisk mellom 20 og 500 millisekunder. Disse bitene kan så komprimeres eller strekkes, forsterkes eller dempes, filtreres etc. uavhengig av hverandre, eller de kan omkastes (permuteres) på forskjellige måter. Det er ikke lurt å klippe av lydkornene brått. Hvis vi f.eks. bytter om rekkefølgen kan vi da risikere at kurven gjør sprang i overgangen mellom kornene. Slike diskontinuiteter gir klikk. Vi legger derfor på envelopes på hvert lille korn, som faller pent ned mot null i hver ende. Det benyttes samme type vindusfunksjoner som vi så på under diskusjonen om FFT (Hanning etc.). Det er vanlig å la vinduene overlappe.

Ved å strekke og komprimere kornene kan vi gjøre pitch-skift og time-stretch med relativt lav kvalitet. Nesten alle kommersielle harmonizere benytter granulering med vindusfunksjoner. Det begynner nå å komme effektbokser som utfører morsomme granuleringsmetoder med tilfeldighet etc. IRCAM-arbeidsstasjonen er også velegnet til granulering, og dette har vært brukt av bl.a. Philippe Manoury og Rolf Wallin.

5.4.4 Granulering med nullgjennomgangsanalyse

Hvis vi passer på å klippe opp lyden akkurat på de tidspunktene der kurven passerer null-nivået på vei oppover, kan vi unngå klikk uten å modulere med vindusfunksjoner. Dette medfører at kornene vil variere i lengde: I perioder med høye frekvenser eller støylyder får vi mange nullgjennomganger og små korn, mens i perioder med lave frekvenser, stillhet eller fattig spektrum får vi store korn.

En spesiell effekt får vi ved å avspille disse kornene i et spiralformet mønster på denne måten:

1 2 3 4 5, 3 4 5 6 7, 5 6 7 8 9, 7 8 9 10 11, etc.

Her er *inkrementet* satt til 2 nullgjennomganger, mens *vindus-størrelsen* er satt til 5 biter mellom påfølgende nullgjennomganger. Dette vil strekke lyden ut i tid, og gi artige effekter.

Det var den sveitsiske komponisten Rainer Boesch som gjorde oss i Norge oppmerksom på denne typen teknikker, og vi har siden benyttet dette flittig. I stykket "Strøk" av Rolf Wallin er hardingfelemusikk behandlet ved hjelp av granulering med nullgjennomgangsanalyse. Den norske lydbehandlingsmaskinen "Fredag" ble i sin tid konstruert med spesiell hardware og

[9] Det er den engelske komponisten Trevor Wishart som har pekt på dette

software som kunne (og forsåvidt fortsatt kan) utføre dette i sann tid med endel spesielle fiduser.

5.4.5 Granulering og FFT

Ved korttids-FFT betrakter man små biter av lyden av gangen. Hver av disse bitene blir dekomponert i sine enkelte sinusoide bestanddeler. Analysen gir oss derfor et stort antall små sinusoide biter som er plassert i tid og frekvens. Vi kan dermed betrakte korttids-FFT som en metode som finner ut hvilke sinusoide korn vi må bruke for å bygge opp lyden. Invers FFT kan da betraktes som en granulær syntese som resyntetiserer lyden ut fra informasjonen om alle disse små kornene.

Dette er en litt uvanlig måte å se på korttids-FFT på. Når vi tenker i granulære baner, blir det naturlig å gjøre ting som å plukke vekk korn, permutere korn i tid og frekvens etc. Vi kan også resyntetisere med andre typer korn enn sinusoider, f.eks små plinger generert av Karplus-strong-algoritmen.

NoTAM bedriver utvikling av et lite program for våre SGI-maskiner som retter seg inn mot å betrakte korttids-FFT som en granulær metode. Programmet heter *Ceres*, etter den romerske korn-gudinnen.

5.4.6 Wavelets

Det går an å forsøke å dekomponere en lyd i andre typer korn enn små sinusoider. En matematiker vil si at vi benytter en annen *basis* for dekomposisjonen. I signalbehandling og matematikk benyttes betegnelsen *wavelets* om våre "korn" ("wavelet" er diminutiv av "wave", og betyr altså "liten bølge"). Det finnes flere ulike klasser av wavelets i bruk. Det finnes også forskjellige algoritmer for å dekomponere en funksjon opp i enkelte wavelets, enkelte er basert på en matematisk operasjon som heter *korrelasjon*.

Wavelets har vært et hett tema i matematikk og signalbehandling i de senere år. Det viser seg at denne betraktningsmåten har mye for seg både teoretisk og i anvendelser. Støyreduksjon ved wavelet- transformasjon er nå en viktig teknikk.

Vi så i avsnittet om FFT at vi kan styre oppløsningen i tid og frekvens ved å justere N. Med wavelets har vi større frihet til å dele opp frekvens-tid-planet. For eksempel kan vi ha god oppløsning i frekvens ved lave frekvenser, mens vi kan minske frekvensoppløsningen oppover i spekteret slik at tidsbestemmelsen blir bedre (se figur 5.30). Det er slik øret virker.

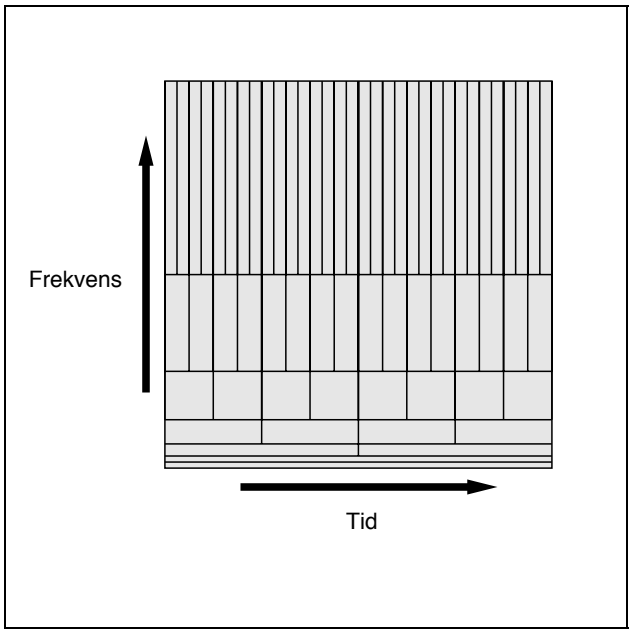
Syntese og analyse med generelle wavelets begynner også å få musikalske anvendelser.

5.5 Spatialisering

For å gi "liv" til en lyd ønsker vi gjerne å simulere at lydkilden er plassert et visst sted i et rom med en viss romklang. Dette heter *spatialisering*.

5.5.1 Ekko

Ekko simuleres ved hjelp av en *digital delay* som forsinker lyden. Den forsinkede lyden (ekkoet) mikses så med originalen. Dersom vi setter forsinkelsestiden til en liten verdi, vil vi få vekselvis konstruktiv (signalene er i fase) og destruktiv (signalene er i motfase) interferens mellom det opprinnelige og det forsinkede signalet på forskjellige frekvenser. Dette betyr at



Figur 5.30: Frekvensavhengig oppløsning ved dekomposisjon i en spesiell type wavelets

oppsettet fungerer som et *kamfilter*, ja mer enn det, det *er* ikke noe annet enn et kamfilter, med differenslikningen

$$y_n = x_n + x_{n-m}$$

der m er antall samples forsinkelse. Avstanden mellom toppene i filterkarakteristikken er gitt ved lengden på forsinkelsen.

Vi kan så multiplisere utgangssignalet med en dempningsfaktor $d < 1$, og sende dette tilbake i inngangen (feedback). Vi får da en effekt av typen Hei Hei Hei Hei Hei Hei. Hvis forsinkelsen er i størrelsesorden 50 ms, vil dette gi en metallisk og kunstig romklang. Dette er fremdeles et filter, nå av den rekursive typen, med differenslikning

$$y_n = x_n + x_{n-m} + d \cdot y_{n-m}.$$

5.5.2 Romklang

Ekkomaskinen med feedback fra forrige avsnitt kan brukes til å simulere romklang i et stort metallrør eller noe slikt. De fleste naturlige rom har en mye mer komplisert geometri, med forskjellige typer materialer som reflekterer lyden på ulik måte. Dette gjør at romklangen får et meget komplisert forløp.

Vi kan måle impulsresponsen til en konsertsal ved å fyre av en pistol, og skrive ut ekkoene som pulser som funksjon av tiden. Vi kan så bruke denne impulsresponsen til å konstruere et digitalt filter direkte. Vi sier da at vi *folder* eller *konvolverer* lyden med impulsresponsen. Hittil har slik "ekte" romklang vært altfor beregningskrevende for praktisk bruk. En etterklangstid på 10 sekunder vil f.eks. kreve en million addisjoner og multiplikasjoner for hvert eneste sample i lyden (ved 50 kHz samplingfrekvens og stereo). Ny teknologi som bl.a. er beregnet for Virtual Reality-anvendelser (f.eks. "Convolvotron") begynner imidlertid å gjøre slike ting mulig.

For å forenkle beregningene, stiliserer vi impulsresponsen endel. Som oftest vil impulsresponsen begynne med endel enkeltstående ekko ("early response") som tilsvarer de primære og sekundære refleksjonene fra vegger og tak. Disse ekkoene kan vi simulere med digitale delays. Etterhvert som klangen dør ut, blir impulsresponsen meget kompleks. Vi får da inn alle mulige kombinasjoner av tidligere ekko som reflekteres enda en gang. Det er vanlig å benytte fiffig oppsatte allpassfiltre for å simulere dette.

Hver gang lyden reflekteres eller passerer gjennom noen meter luft vil den miste diskant. Digitale delays med feedback og et lavpassfilter i tilbakekoblingen vil kunne simulere dette. [10]

5.5.3 Plassering av lydkilder i rommet

I heftet om akustikk så vi på hvordan øret og hjernen bestemmer retningen og avstanden til en lydkilde. For å simulere *avstand* bruker vi disse effektene:

- Endring av amplityde, ved at amplityden faller omvendt proporsjonalt med avstanden (effekten, som avhenger av kvadratet av amplityden, vil da falle omvendt proporsjonalt med kvadratet av avstanden).
- Endring av frekvensinnhold, ved at lyden mister mer og mer diskant ettersom avstanden øker (simuleres med et lavpassfilter)
- Doppler-effekten. Dette simuleres ved å sette opp en digital delay som simulerer tiden lyden bruker fra lydkilden til lytteren. Ettersom avstanden endres, vil da også delay-tiden endres, og Dopplerske pitch-skift oppstår da helt "av seg selv".
- Balanse mellom romklang og direkte signal. Lyden vil virke nærere jo tørrere den er.

For å simulerer *retning* bruker vi disse effektene:

- Amplityde-panorering mellom kanalene
- Faseforskjell og Haas-effekt (simuleres med digitale delays)
- Eventuelt simulering av ytterørets og hodets filtrerende effekt avhengig av retning

5.5.4 Andre effekter

Chorus kan simuleres ved å mikse et antall parallelle systemer der lyden gjennomgår en langsomt varierende, tilfeldig forsinkelse. Dette vil simulere både at de separate lydkildene er noe ute av fase, og at de har noe ulik pitch (på grunn av de resulterende Doppler- skiftene). I tillegg kan vi legge på en tilsvarende tilfeldig modulasjon av amplityde.

Flanger er ikke noe annet enn en langsom endring av delay-tid, gjerne modulert av en lavfrekvent sinus.

Fuzz lages ved å kjøre lyden gjennom en ulineær overføringsfunksjon (se avsnittet om ulineær forvrengning).

[10] Hvis delay-tiden settes meget liten, er vi tilbake i Karplus- Strong-algoritmen!

5.6 Analyse

Med digital lydbehandling kan vi trekke ut en rekke måltall som kan brukes til å analysere musikk. De fleste av disse metodene er beskrevet teknisk i tidligere avsnitt, men det kan være nyttig å se litt mer på hvordan de kan anvendes i en musikologisk sammenheng.

Sonogram

Et sonogram er et spektrogram der tiden går langs x-aksen, frekvens langs y-aksen og amplityder vises som gråtoner (se figur 3.10). I kombinasjon med et oscillogram som viser amplityde, er dette en meget nyttig og lettlest representasjonsform som egner seg godt til publikasjon og i undervisning.

Det finnes mange programmer som kan produsere sonogrammer, men ikke alle gir like pene resultater. På NoTAM har vi laget et lite program for SGI-maskinene som heter *Sono*. Vi bruker Sono bl.a. til å lage partiturer for lydteknikeren under framføring, som hjelpemiddel under produksjon av musikkvideoer, og som generell dokumentasjon.

Spektrale måltall

Et sonogram inneholder altfor mye informasjon til at det kan brukes direkte til f.eks. statistiske sammenlikninger. Det er endel måltall som vi kan bruke til å karakterisere visse egenskaper ved spekteret og dermed redusere datamengden.

For det første kan vi prøve å finne grunntonefrekvensen (pitch) ut fra deltonene i spekteret. Dette er pitch-tracking i frekvensdomenet. Slike metoder vil i utgangspunktet bare fungere for enkle, monofone lydkilder, og selv for ideelle lydkilder vil man kunne få feilaktige resultater. Imidlertid er det gjerne slik at dersom pitch-trackeren først har funnet riktig grunntone, vil den gi et meget nøyaktig resultat. Analyser med hensyn på f.eks. intonasjon og mikrotonalitet vil derfor kunne fungere meget bra.

Et annet nyttig måltall er det *spektrale tyngdepunktet* (centroiden). Hvis vi har N deltoner (eller kanaler i FFT-analysen) med frekvenser f_i og amplityder a_i, er det spektrale tyngdepunktet gitt ved en vektet og normalisert sum:

$$f = \frac{\sum_{i=1}^{N} a_i f_i}{\sum_{i=1}^{N} a_i}.$$

Tyngdepunktet gir et godt mål for hvor "lyst" spekteret låter, tildels uavhengig av eventuell pitch.

Spektral entropi er et måltall som sier noe om hvordan energien er spredt i spekteret. En enkel sinus gir meget lav entropi, fordi energien er konsentrert i ett lite bånd. Hvit støy gir meget høy entropi, fordi energien er spredt fullstendig ut. Entropien korrellerer med støyinnhold i lyden, og vi kan også si at den er et mål for *kompleksitet*. Spektral entropi kan måles med Shannons informasjonsindeks:

$$E = -\sum_{i=1}^{N} a_i \ln a_i,$$

der a_i er amplityder for de N FFT-kanalene. Disse amplitydene bør være normaliserte slik at $\sum a_i = 1$.

Alle disse måltallene, samt amplityden, kan presenteres sammen med et sonogram i et integrert diagram som gir et rimelig komplett bilde av lyden. På NoTAM har vi laget programmet *Bio* (opprinnelig ment for bioakustisk analyse) til dette formålet.

Vi vet ikke om noen robuste analysemetoder som kan finne starttidspunkter for lyder i et komplekst lydbilde. For analyser av rytmikk er det nok derfor foreløpig nødvendig å markere tidspunktene manuelt ved hjelp av f.eks. Bio. Ved å se på analysedataene og lytte samtidig, kan man bestemme tidspunktene forholdsvis nøyaktig og raskt.

Numeriske verdier for måltallene som er nevnt over kan behandles videre statistisk for f.eks. å finne korrelasjoner og grupperinger.

Kapittel 6

C-sound

Dette kapittelet gir en kortfattet innføring i Csound, som er et "klassisk" program for syntese og omforming av lyd. Csound ble opprinnelig skrevet av Barry Vercoe ved MIT, og føyer seg inn i rekken av beslektede systemer helt fra Max Mathews' *Music 4* på 1960-tallet, via *Music 4B*, *Music 4C*, *Music V*, *Music 11* og *Cmusic*. Csound brukes av svært mange komponister i dag, og er tilgjengelig gratis for UNIX-maskiner, Macintosh og IBM PC. En gjennomgang av Csound er obligatorisk i nesten alle kurs i computermusikk rundt om i verden.

Csound er ikke et generelt programmeringsspråk, men et språk for å definere oppbyggingen av instrumenter fra primitive byggeblokker. På mange måter kan Csound sammenliknes med *Max*, hvor man også kobler sammen byggeklosser til større enheter. Csound har imidlertid i utgangspunktet ikke noe grafisk brukergrensesnitt, så instrumentene må bygges opp tekstlig med en syntaks som nok dessverre virker litt kryptisk i begynnelsen.

Med Csound er det nesten ingen grenser for hvor komplekse ting man kan gjøre. Hvis man ønsker et FM-instrument med tusen operatorer, er det helt i orden. Vanligvis kjøres ikke Csound i sann tid, så arbeidsgangen er at man først lager den tekstlige beskrivelsen. Deretter starter man Csound, som sender den produserte lyden til en lydfil for senere avspilling.

Du må gi *to* tekstfiler som input til Csound. I den ene filen, *orkesterfilen*, ligger en beskrivelse av de forskjellige instrumentene, hvor du har angitt sammenkoblingen av byggeklosser som oscillatorer, filtre etc. I den andre filen, *partiturfilen* eller (*score file*), ligger en liste med noter, dvs. oppkall av instrumentene til ulike tider og med ulike parametre. Partiturfilen kan godt sammenliknes med en MIDI-fil, og det finnes programmer som kan konvertere mellom MIDI-filer og partiturfiler. I partiturfilen angis også de forskjellige *tabellene* som Csound skal bruke til kurveformer, enveloper etc. Hvis du kan programmere litt i C, kan du lage et C-program som skriver partiturfiler etter en eller annen algoritme som du finner på (algoritmisk komposisjon).

For å få noe særlig utbytte av Csound, må du ha en god forståelse av prinsippene for lydsyntese og lydbehandling. Du må vite hva FM-syntese, additiv syntese, filtrering etc. er, og du må vite ting som f.eks. hvilke matematiske operasjoner som benyttes for å mikse lyder eller legge en envelope på en lyd. Dette er gjennomgått i tidligere kapitler.

6.1 Et lite eksempel

For å lage en liten lyd med Csound, må vi først definere et instrument. Dette gjøres i orkesterfilen, som vi kan kalle `eksempel.orc`. Start opp en teksteditor med kommandoen `jot`

`eksempel.orc` på Indy. På Mac skulle *Word* eller noe slikt være brukbart.

Skriv inn følgende (forklaring under):

```
sr     =  16000
kr     =  1000
ksmps  =  16
nchnls =  1

instr 1
  a1    oscil  10000,440,1
  out   a1
endin
```

6.1.1 Header

Først i orkesterfilen må vi ha en *header*, der vi spesifiserer samplingfrekvens og et par andre rare ting. `sr = 16000` betyr at vi ønsker en samplingfrekvens på 16 kHz. Dette er ikke mye, men nok til dette enkle eksemplet (beregningene går fortere jo lavere samplingfrekvens vi bruker). Ulike maskiner har litt ulike sett av tillatte samplingfrekvenser, på Indy kan man velge mellom 8000, 11025, 16000, 22050, 32000, 44100 og 48000.

Et viktig poeng i Csound er skillet mellom *samplingrate* og *kontrollrate*. Samplingraten er antall samples i lyden pr. sekund, og vi kunne godt ha beregnet alle tallene i instrumentet med denne hastigheten. Ofte har man imidlertid parametre som endrer seg noe langsommere, f.eks. enveloper. For å minske beregningstiden, kan man bestemme at disse verdiene skal oppdateres med en lavere hastighet. Denne lavere "samplingfrekvensen" kalles kontrollrate, og vi setter den her til 1000 Hz med kommandoen `kr = 1000`. Vi må også angi forholdet mellom samplingraten og kontrollraten med kommandoen `ksmps = 16`. Dette er selvsagt idiotisk, maskinen burde kunne klare denne divisjonen selv.

Kommandoen `nchnls = 1` angir at vi skal lage en mono-lydfil.

6.1.2 Instrumentet

Man kan ha mange forskjellige instrumenter i et orkester, og disse nummereres i orkesterfilen. Dette gjøres i innledningen til instrumentdefinisjonen, her med kommandoen `instr 1`. Instrumentet avsluttes med kommandoen `endin`.

"Boksene" som vi skal koble sammen i et instrument angis på separate linjer. Hver linje begynner med et *variabelnavn* som brukes som referanse til output fra boksen. Så følger boksens *type*, med eventuelle typespesifikke *parametre*. De forskjellige boks-typene og deres parametre er beskrevet i Csound-manualen.

`a1 oscil 10000,440,1` betyr således at vi setter opp en boks av type *oscil*, som er en generell oscillator. Den tar tre parametre, nemlig amplityde, frekvens og kurveform. Amplityden angis som et heltall, der 32767 er fullt utslag. Frekvensen oppgis i Hz. Kurveformen oppgis med et tabellnummer som refererer til en tabell som vi senere skal definere i partiturfilen.

Output fra oscillatoren sendes til variabelen `a1`. Ved å la variabelnavnet begynne med en *a*, angir vi at variabelen skal oppdateres for hvert sample (*"audio rate"*). Vi kunne dermed godt ha kalt variabelen "asparges", men ikke "potet". Hvis navnet hadde begynt med *k*, ville den blitt oppdatert med kontrollhastighet, dvs. her 1000 ganger i sekundet.

`out a1` betyr at output fra instrumentet skal hentes fra variabelen `a1`.

Vi er nå ferdig med orkesterfilen.

6.1.3 Partituret

Så må vi lage partiturfilen. Vi kan kalle den `eksempel.sco`. Skriv inn følgende:

```
f1  0  4096  10  1
i1  0  4
e
```

`f1` betyr at vi skal definere funksjonstabell nummer 1, som vi refererte til i orkesterfilen. Tabellen skal opprettes ved tidspunkt 0 sek, og den skal inneholde 4096 samples (må være en potens av 2). Vi skal bruke en subrutine som heter `GEN10`, derfor tallet 10. `GEN10` tar som parametre de relative amplitydene til en overtonerekke; her spesifiserer vi bare en grunntone, slik at tabellen fylles med en sinusperiode.

Så følger alle "notene", dvs. oppkall av instrumenter til forskjellige tidspunkter. Her har vi bare en note; `i1` betyr at vi skal bruke instrument nummer 1. Noten skal starte ved tidspunkt 0 sek, og skal vare i 4 sekunder.

`e` angir slutten på partiturfilen.

Vi kan nå starte Csound, på Indy gjøres det med kommandoen `csound -Ao eksempel eksempel.orc eksempel.sco`. Dette er litt mer inngående forklart i heftet *Lokal guide til lydbehandling i UNIX*. Csound skal nå produsere en lydfil med navnet *eksempel*. Spill lyden med f.eks. `sfplay eksempel`.

Som man ser, er syntaksen i Csound meget kortfattet, konsis og kryptisk. Det tar tid å venne seg til dette, men fordelen er at det blir mindre å skrive.

6.2 Flere triks

6.2.1 P-felter

Instrumentet i forrige kapittel kunne bare spille en fast frekvens med en fast lydstyrke. Det må vi fikse. Parametre i instrumentet kan styres fra verdier som vi oppgir i partiturfilen. Parametrene til `i`-kommandoene i partiturfilen heter `p1`, `p2` osv. De tre første *p-feltene* har en fast betydning:

$$
\begin{aligned}
p1 &= instrumentnummer \\
p2 &= starttidspunkt \\
p3 &= varighet
\end{aligned}
$$

(se eksemplet i forrige avsnitt).

De resterende p-feltene (p4, p5 etc.) kan vi benytte som vi vil. Vi kan la p4 styre frekvens og p5 amplityde. Det finnes en funksjon som heter `cpspch` som gjør om fra formatet *oktav.pitchklasse* til en frekvens i Hz. Funksjonen `ampdb` gjør om fra en dB-verdi til en amplityde.

Orkesterfilen kan da skrives slik:

```
sr     =  16000
kr     =  1000
ksmps  =  16
nchnls =  1

instr 1
  a1 oscil ampdb(p5),cpspch(p4),1
  out a1
endin
```

Og partiturfilen kan se slik ut (legg merke til at semikolon markerer en kommentar):

```
f1  0  4096  10  1
;   p2 (start)    p3 (varighet)    p4 (frekvens)    p5 (styrke)
i1  0             0.1              8.00             50
i1  0.2           0.1              8.00             55
i1  0.4           0.1              8.00             60
i1  0.8           0.1              8.04             65
i1  1.2           0.1              8.07             70
i1  1.6           1                9.00             70
i1  1.6           1                9.04             70
i1  1.6           1                9.07             70
e
```

6.2.2 Envelopes, GENs, carry

Vi fortsetter å bygge ut instrumentet fra forrige avsnitt. For det første vil vi gjerne legge en *omhyllingskurve* (envelope) på hver tone. Vi blar litt i Csound-manualen, og finner "boksen" (enhetsgeneratoren) `linen`. Formatet er definert slik:

```
kr linen kamp,irise,idur,idec
ar linen xamp,irise,idur,idec
```

Typisk Csound-kryptisk! Hva betyr dette? Det man forsøker å få fram er at `linen` kan produsere sin output enten med "kontrollrate" eller med "audio rate" (samplingfrekvensen). Attack-tiden ("rise") kan oppdateres med "initialiseringsrate", dvs. hver gang instrumentet kalles opp fra en `i`-kommando i partituret. Det sammet gjelder "dur" og "dec" (release-tiden). Amplityden kan oppdateres med kontrollrate eller langsommere dersom output oppdateres med kontrollrate. Hvis output oppdateres med audio rate, kan amplityden oppdateres med alle slags hastigheter (angitt med "x").

Vi velger k-versjonen, og sender output fra `linen` inn i amplityde-parameteren til oscillatoren. I tillegg lar vi kurveform-parameteren styres fra et nytt p-felt, slik at vi kan endre klangfargen for hver note:

```
sr     =  16000
kr     =  4000
ksmps  =  4
nchnls =  1
```

```
instr 1
  k1    linen   ampdb(p5),0.05,p3,p3*0.4
  a1    oscil   k1,cpspch(p4),p6
        out     a1
endin
```

I partiturfilen definerer vi en funksjonstabell nummer 2 ved hjelp av `GEN10`, med et rikere spektrum (8 overtoner), og legger til et nytt p-felt p6. I tillegg utnytter vi en nyttig finesse i syntaksen, nemlig *carry-tegnet* punktum. Et punktum betyr at vi skal bruke samme verdi som i forrige linje. De siste punktum på en linje kan utelates. Det finnes flere andre carry-tegn, bl.a. *pluss*, som betyr at vi skal bruke p2+p3 fra forrige linje, dvs. tidspunktet for forrige notes slutt (brukes ikke her).

```
f1  0  4096  10  1
f2  0  4096  10  1 .6 .3 .1 .3 .6 .3 .1
;   p2 (start)  p3 (varighet)  p4 (frekvens)  p5 (styrke)  p6 (kurve)
i1  0           0.1            8.00           50           1
i1  0.2         .              .              55
i1  0.4         .              .              60
i1  0.8         .              8.04           65
i1  1.2         .              8.07           70
i1  1.6         1              9.00           .            2
i1  .           .              9.04
i1  .           .              9.07
e
```

6.3 Synteseteknikker

Dette kapittelet presenterer endel enkle Csound-instrumenter som demonstrerer ulike syntesemetoder.

6.3.1 Ringmodulering

I dette eksemplet ser vi hvordan man kan gjøre *ringmodulering*, dvs. at vi multipliserer to lyder med hverandre. Vi velger å modulere en lyd med et rikt spektrum (carrier) med en sinustone hvor frekvensen varierer (modulator):

```
sr     =  32000
kr     =  2000
ksmps  =  16
nchnls =  1

instr 1
  kmodf linen   40,p3*0.4,p3,p3*0.4      ; Styrer frekvensen til modulator
  kcara linen   10000,p3*0.1,p3,p3*0.1   ; Styrer amplityden til carrier
  amod  oscil   10000,kmodf,1            ; Modulator
```

```
  acar  oscil   kcara,100,2              ; Carrier
        out     amod*acar/10000
endin
```

(Legg merke til at vi må skalere output ved å dividere med 10000, for å unngå klipping).

```
f1 0 4096 10 1
f2 0 4096 10 1 0.6 0.3 0.1 0.1 0.3 0.6 0.1
i1 0 20
e
```

6.3.2 FM-syntese

FM er lett å få til i Csound, f.eks. slik:

```
sr     =  32000
kr     =  2000
ksmps  =  16
nchnls =  1

instr 1
  kmodf linen   400,p3*.05,p3,p3*.95     ; Styrer amplityden til modulator
  kcara linen   10000,p3*.05,p3,p3*.95   ; Styrer amplityden til carrier
  amod  oscili  kmodf,100,1
  acarr oscili  kcara,100+amod,2
        out     acarr
endin
```

Vi bruker her `oscili` istedenfor `oscil`, fordi dette minsker interpolasjonsstøyen.
Partiturfilen er den samme som i forrige eksempel.

6.3.3 Subtraktiv syntese

Subtraktiv syntese betyr filtrering av en lyd med et rikt spektrum. I orkesterfilen under bruker vi `reson`, som er et båndpassfilter med styrbar senterfrekvens og båndbredde.

For å holde lydstyrken på det opprinnelige nivået selv etter filtreringen, bruker vi `balance`, som her sørger for at `afilt` holdes på samme nivå som `abuzz`.

```
sr     =  16000
kr     =  2000
ksmps  =  8
nchnls =  1

instr 1
  kbuzz linen   10000,p3*.1,p3,p3*.1         ; Styrer amplityden
  kfilt linseg  2000,p3*0.5,0,p3*0.5,2000    ; Styrer senterfrekvens
  abuzz oscili  kbuzz,100,1                  ; Tonegenerator
  afilt reson   abuzz,kfilt,100              ; Filter
```

```
  abal  balance afilt,abuzz                  ; Automatisk gain-kontroll
        out     abal
endin
```

For å få et rikt spektrum på den opprinnelige lyden, bruker vi `GEN11` i partiturfilen. `GEN11` gir et flatt spektrum, og vi ber om å få med 50 harmoniske:

```
f1 0 8192 11 50
i1 0 20
e
```

Lydkilden kan byttes ut med en hvitstøygenerator, slik:

```
sr     =  16000
kr     =  2000
ksmps  =  8
nchnls =  1

instr 1
  knois linen   10000,p3*.1,p3,p3*.1         ; Styrer amplityden
  kfilt linseg  2000,p3*0.5,0,p3*0.5,2000    ; Styrer senterfrekvens
  anois rand    knois                        ; Tonegenerator
  afilt reson   anois,kfilt,100              ; Filter
  abal  balance afilt,anois                  ; Automatisk gain-kontroll
        out     abal
endin
```

6.3.4 Waveshaping

Følgende er basert på et instrument laget av Rob Waring. `expseg` lager en stykkevis eksponensiell funksjon. Parametrene til `expseg` er startverdi, tid til verdi 2, verdi 2, tid til verdi 3, etc.

`tablei` er en boks for tabelloppslag; overføringsfunksjonen er angitt i p6. Siden `aosc` i utgangspunktet varierer mellom -250 og +250, har vi brukt en *variabeltilordning* for å addere 256. På den måten får vi en peker inn i tabellen, som er 513 samples lang.

```
;waveshaping instrument

sr=32000
kr=3200
ksmps=10
nchnls=1

instr   1

kenv    linen   p4,.1,p3,.1
aenv    expseg  1,p3*p7,250,p3*p8,150,p3*p9,p10,p3*.5,.1
aosc    oscili  aenv,p5,1
```

```
aosc    =       aosc+256
atab    tablei  aosc,p6
        out     atab*kenv
endin
```

I partiturfilen har vi brukt `GEN08` for å fylle tabellen. `GEN08` lager en glatt, fin interpolering (spline) mellom de oppgitte punktene.

```
f1 0 2048 10 1
f2 0 513 8 -1 89 -.5 89 -.6 156 .6 89 .5 89 1
f3 0 513 8 -1 100 0 100 -.4 112 .4 100 0 100 1

i1    0      6     10000    275    3    .1     .05   .3   20
i1    0      6      5000    350    3    .1     .05   .3   20
i1    0      6      3300    425    3    .1     .05   .3   20
i1    0      9      2500    475    2    .06    .04   .4   70
i1    0      9      2000    525    2    .06    .04   .4   70
i1    0      9      1600    600    2    .06    .04   .4   70
i1    0      9      1400    675    2    .06    .04   .4   70
```

6.3.5 FOF

Det finnes en egen boks for FOF-syntese. `fof` er en ganske kompleks enhetsgenerator, og den har fått et helt appendix i Csound-manualen. Her er et instrument man kan prøve:

```
      sr = 32000
      kr = 2000
      ksmps = 160

      instr    1
k1    oscil    2, 4, 1
k2    linseg   0, p3/2, 0, p3/2, 0.8
k3    expseg   1, p3/3, 1, p3, 0.001
a1    fof      5000, p4+k1,  500*k3, k2, 120, .003, .017, .005, 10, 1, 2, p3
a2    fof      4000, p4+k1,  850*k3, k2, 120, .003, .017, .005, 10, 1, 2, p3
a3    fof      3000, p4+k1, 2500*k3, k2, 120, .003, .017, .005, 10, 1, 2, p3
a4    fof      2000, p4+k1, 3500*k3, k2, 120, .003, .017, .005, 10, 1, 2, p3
a5    =        a1 + a2 + a3 + a4
      out      a5
      endin

f1 0 4096  9 1 1 0
f2 0 1024 19 .5 .5 270 .5
i1 0 12 100
i1 1 11 133
i1 2 10 150
e
```

6.3.6 Karplus-Strong

Karplus-Strong-algoritmen har også en egen boks, som heter `pluck`. Se Csound-manualen for detaljene, men `pluck` er enkel å bruke:

```
      sr = 32000
      kr = 200
      ksmps = 160

      instr    1
a1    pluck    ampdb(p5),cpspch(p4),cpspch(p4),0,1
a2    linen    a1,.01,p3,p3*0.3
      out      a2
      endin
```

I partiturfilen bruker vi en ny fidus, nemlig *tempo*-kommandoen `t` som justerer alle tidspunkter og varigheter. `t 0 90` betyr at tempo skal settes til 90 beats pr. sek. istedenfor de vanlige 60, fra tidspunkt 0.

```
t 0 90
i1 0   0.25 6.04 70
i1 0.6 .    6.07
i1 1.2 0.9  6.09
i1 +   0.25 6.04
i1 2.7 .    6.07
i1 3.3 0.3  6.10
i1 +   1.0  6.09
i1 4.8 0.25 6.04
i1 5.4 .    6.07
i1 6.0 0.9  6.09
i1 +   .    6.07 75
i1 7.8 4.   6.04 70
i1 .   .    5.04
i1 .   .    5.09
e
```

6.4 Lydbehandling

Det finnes bokser i Csound som leser fra forhåndsproduserte lydfiler. Dette gjør at Csound er like velegnet til lydbehandling som til lydsyntese.

6.4.1 Kor

I instrumentet under bruker vi `soundin`, som leser en angitt lydfil. Nytt er også `delayw` som skriver inn i en delay, `delayr` som angir en fast delaytid, og `deltapi` som er en variabel tapning av delayen. Tre tapninger beveger seg her langsomt fram og tilbake, ute av takt med hverandre:

```
    sr=32000
    kr=32000
    ksmps=1

instr 1
ainnlyd   soundin   "Oyvind/testlyd"
adummy    delayr    1
amod1     oscili    .05,.1,1
atap1     deltapi   amod1+.05
amod2     oscili    .05,.11,1
atap2     deltapi   amod2+.05
amod3     oscili    .05,.12,1
atap3     deltapi   amod3+.05
          delayw    ainnlyd
          out       (atap1+atap2+atap3)/3;
endin

f1 0 8192 10 1
i1 0 45
e
```

6.4.2 Global romklang

Ofte vil man legge en effekt på mange instrumenter samtidig, eller man vil la en varierende kontroll affisere et antall instrumenter. For å oppnå dette, bruker man *globale variable*. Globale variable begynner med bokstaven *g*, etterfulgt av en bokstav etter det vanlige systemet (a, k, i). Det er lurt å initialisere globale variable i orkester-headeren, med kommandoen `init`.

Eksemplet under viser hvordan romklangen kan kjøre helt uavhengig av andre instrumenter. Etter at instrumentene har sendt sine outputs til den globale variabelen, kan den brukes som input i det separate romklangsinstrumentet. Siden instrumentene adderer inn i den globale variabelen, må vi også passe på å nulle den etter at den er brukt i romklangsinstrumentet (dette kan kanskje virke litt pussig).

```
    sr    = 32000
    kr    = 1000
    ksmps = 32
    garev init 0

instr 1                         ; Hvitstoy
  anois rand    5000
  garev =       garev+anois     ; Send til reverb
        out     anois           ; og direkte ut
endin

instr 2                         ; 220 Hz pip
  abeep oscil   10000,220,1
  garev =       garev+abeep
```

```
        out     abeep
endin

instr 99                        ; Romklang
  asig  reverb  garev*.1,2      ; Input fra garev, 2 sek. klangtid.
  out   asig
  garev =       0
endin


f1  0  4096  10  1
i99 0  5                        ; Skru paa klangen i 5 sek.
i1  0  .1
i2  1  .1
i1  2  .1
i2  2  .1
e
```

6.4.3 Fasevokoding

Det separate programmet `pvanal` kan gjøre en fasevokodingsanalyse av en lyd. Analysedataene sendes til en egen fil, som siden kan leses inn i et Csound-instrument for resyntese med enhetsgeneratoren `pvoc`.

`pvanal` krever en mono AIFF-fil (ikke AIFC), og programmet kjøres med UNIX-kommandoen "`pvanal innfil analysefil`".

`pvoc` leser fra analysefilen med en varierende *peker*. Denne pekeren kan bevege seg hurtig eller langsomt gjennom lyden, gå baklengs etc. uten at pitch endres. På denne måten kan man f.eks. lage time-stretch. I eksemplet strekkes lyden med en faktor 2, samtidig som vi transponerer en kvint opp:

```
sr = 44100
kr = 441
ksmps = 100

instr  1
ktime   line   0, p3, p3/2
aout    pvoc   ktime, 1.5, "pvfil"
out    aout
endin

i1 0.0 30.0
e
```

6.4.4 Ceres

Ceres er et annet fasevokodingsprogram med en kobling mot Csound. Ceres er menystyrt, og relativt enkelt å bruke. Med *Sieve*-funksjonen kan man plukke ut bare de sterkeste deltonene

på ethvert tidspunkt, og disse små "ploppene" kan betraktes som egne noter. Ceres kan skrive en Csound partiturfil med alle disse notene.

6.4.5 Lineær-prediktiv koding

Det separate programmet `lpcanal` kan gjøre en lineær-prediktiv analyse av en lyd. Analysedataene sendes til en egen fil, som siden kan leses inn i et Csound-instrument for resyntese med enhetsgeneratoren `lpread`. `lpcanal` har problemer med høye samplingrater (44.1 og 48 kHz).

`lpcanal` kjøres med UNIX-kommandoen "`lpcanal innfil analysefil`". Med på lasset følger en (upålitelig) pitch-tracker og en RMS-analyse. Se Csound-manualen for detaljene. Analysefilen leses inn i Csound f.eks. slik:

```
sr = 32000
kr = 1000
ksmps = 32

instr   1
ktime                   line     0, p3, p3
krmsr,krmso,kerr,kcps   lpread   ktime, "analyse.lpc"
kcps                    =        (kcps == 0 ? 220 : kcps)
avoice                  buzz     krmso, kcps, int(sr/880), 1
aunvoc                  rand     krmso
asig                    =        (kerr < .1 ? avoice : aunvoc)
aout                    lpreson  asig
                        out      aout
endin
```

Her var det mye rart! For det første bruker `lpread` en tidspeker, her kalt `ktime`, akkurat som `pvoc`. Slik kan man lage hastighetsendringer etc. `lpread` oppdaterer variablene `krmsr` (RMS av residualen, brukes sjelden), `krmso` (RMS av originalsignalet), `kerr` (andelen støy), og `kcps` (pitch-tracking). Dessuten oppdateres alle filterkoeffisientene som senere brukes av filteret `lpreson`.

I tilordningene brukes en syntaks som er kjent fra programmeringsspråket C. Den første tilordningen skal tolkes slik: "Hvis `kcps` er 0 så sett `kcps` til standardfrekvensen 220, ellers sett den tilbake til `kcps`". Den andre tilordningen tester på `kerr` for å bestemme om eksitasjonssignalet skal tas fra `buzz` eller `rand`. Dette tilsvarer omtrent en `switch` i Max.

LPC er en kraftig og morsom teknikk. Prøv f.eks. å la pitch være konstant (robotstemme), eller la pitch styres fra RMS-verdiene! Effekter som "snakkende orkester" etc. er også mulige. Ha imidlertid ikke for store forhåpninger til lydkvaliteten når du bruker LPC.

6.4.6 Deltoneanalyse/additiv resyntese

Additiv syntese kan gjøres med vanlige oscillatorer og kontrollfunksjoner som `line`, men det finnes en mer praktisk metode for komplekse tilfeller. Enhetsgeneratoren `adsyn` leser en egen kontrollfil, der amplityde og frekvens til opptil 50 deltoner spesifiseres til diskrete tidspunkter. `adsyn` interpolerer lineært mellom disse. Et eget grafisk brukergrensesnitt eksisterer for å editere `adsyn`-kontrollfiler. Start dette programmet med UNIX-kommandoen `adsyn`.

Kontrollfilen kan senere leses inn i et Csound-instrument, slik:

```
sr=32000
kr=32
ksmps=1000

instr 1
aout  adsyn  1,1,1,"test.ads"
      out aout/2
endin
```

Divisjonen med 2 er bare for å unngå klipping. Man må kanskje justere litt her. Det er en feil i `adsyn` som gjør at den er ganske hårsår når det gjelder kontrollraten. `ksmps=1000` later til å fungere.

Amplityder, frekvenser og tidspunkter kan skaleres med de tre første parametrene til `adsyn` (her er de alle skalert med identiteten).

Det separate programmet `hetro` kan gjøre en filterbankanalyse av en lyd der amplityde og frekvens til et antall deltoner spores og lagres i en `adsyn` kontrollfil. `hetro` krever en mono AIFF-fil (ikke AIFC), og programmet kjøres med UNIX-kommandoen "`hetro innfil analysefil`". Det er endel nyttige flagg til denne kommandoen, som er beskrevet i Csound-manualen. `hetro` er beryktet for sine klikk og annen støy.

6.4.7 Videre

Nå skulle du være i stand til å fortsette utforskingen av Csound på egenhånd, ved hjelp av Csound-manualen. Gå gjerne gjennom "tutorialene" til Richard Boulanger, appendix 2 i manualen. Filene finnes i `/local/b/csound/tutorfiles`.