Lyd på datamaskiner

Knut Mørken

November 17, 2008

1 Digital lyd

Svært mye av informasjonen som omgir oss i dag er lagret digitalt og blir overført digitalt. Vi har digital lyd på CD-plater, på internet og på kassettbånd, vi har digitale kameraer og digitale videokameraer, vi har digitale telefoner, vi har DVD-spillere der film er lagret digitalt, vi har bøker lagret digitalt på CD-rom plater, TV-signaler blir sendt digitalt via satelitt og så videre. I denne seksjonen skal vi se litt nærmere på hva ordet 'digitalt' betyr i sammenheng med lyd.

Fenomenet lyd er det vi oppfatter når lufttrykket ved trommehinnene i ørene varierer på bestemte måter. Nærmere bestemt må lufttrykket ossillere mellom 20 og 20 000 ganger i sekundet for at vi skal oppfatte ossillasjonene som lyd. Grensene på 20 og 20 000 er ytterpunktene — for de fleste ligger grensene litt over 20 og litt under 20 000. I forhold til det totale lufttrykket er svingningene vi oppfatter som lyd svært små, men øret er et følsomt organ som reagerer på mye mindre energimengder enn for eksempel øyet.

Mesteparten av lydene vi hører inneholder en ujevn blanding av mange ulike trykkvariasjoner slik at det er umulig å si at variasjonene ligger fast på for eksempel 1000 ossillasjoner i sekundet. Hvis ossillasjonene er jevne vil vi oppfatte lyden som om den ligger i en bestemt tonehøyde, slik som når et musikkinstrument spiller en enkelt tone. Det er antall ossillasjoner pr. sekund i slike 'rene' toner som må ligge innenfor 20 og 20 000, og vi refererer til antall ossillasjoner pr. sekund som *frekvensen* til tonen. Frekvens måles i Hz[1] slik at for eksempel en tonehøyde som svarer til 100 ossillasjoner pr. sekund angis som 100 Hz.

Det følger fra et grunnleggende resultat i en del av matematikken som kalles *Fourier-analyse* at enhver lyd kan skrives som en passende sum av 'rene' lyder med en veldefinert frekvens. Siden vårt øre bare kan oppfatte frekvenser mellom 20 Hz og 20 000 Hz er det nok å ta med rene lyder med en frekvens som ligger

[1] Uttales hertz etter den tyske fysikeren Heinrich Hertz (1857–1894).

i dette hørbare området når vi spalter opp en lyd som en sum av rene lyder på denne måten.

For å lagre lyd må vi lagre størrelsen på trykkvariasjonene som den aktuelle lyden produserer. I utgangspunktet varierer lufttrykket kontinuerlig i tid slik at vi for hvert tidspunkt bør lagre hva lufttrykket skal være. Dette var tanken bak gamle vinylplater der trykkvariasjonene var kodet som små ujevnheter i vinylen som kunne plukkes opp av stiften på platespilleren.

På CD-plater lagres lyden *digitalt*. Dette betyr at lufttrykket måles med jevne mellomrom og at hver av disse målingene lagres — vi sier at lyden *samples* og kaller målingene for *sampler*. Når lyden så skal spilles av igjen må avspillingsutstyret 'fylle inn' den informasjonen som skal ligge mellom samplene ved hjelp av en passende matematisk funksjon. For å få best mulig lydkvalitet er det viktig at tidsrommet mellom samplene er kort, og på ei CD-plate er det lagret 44 100 målinger pr. sekund, noe vi referer til ved å si at *samplingsraten* er 44100. Det viser seg at hvis vi bruker f sampler pr. sekund så kan vi ikke få med oss høyere lydfrekvenser enn $f/2$ Hz. For å være sikre på å få med oss frekvenser opp til 20 000 Hz må vi derfor ha minst 40 000 sampler pr. sekund, og med 44 100 sampler pr. sekund har vi da litt å gå på.

Det at lyd lagres digitalt innebærer også et annet aspekt, nemlig det at lydsamplene *kvantiseres*. I utgangspunktet burde vi egentlig lagre størrelsen på ossilasjonene med uendelige mange siffers nøyaktighet, men det lar seg selvsagt ikke gjøre. På CD-plater er hvert sample av lyden lagret med 16 bits (verdiene er lagret binært, som et tall i to-tallssystemet) hvilket betyr at de lagrede måleverdiene bare kan anta $2^{16} = 65536$ forskjellige verdier.[2] Typisk vil da verdiene kunne variere mellom -2^{15} og $2^{15} - 1$, og vi kan bruke datatypen `short` hvis vi vil arbeide med slike data i et Java-program. Vi må dessuten huske at lyden er lagret i stereo slik at hver gang musikken måles får vi 2 verdier som hver krever 16 bits. Dette betyr at det på en musikk-CD er lagret $2 \cdot 16 \cdot 44100 = 1411200$ bits pr. sekund som svarer til 176400 bytes pr. sekund (en byte er 8 bits). På en typisk CD med 1 times spilletid ligger det altså en informasjonsmengde på ca. 600 Mb (1 Mb er $1024 \cdot 1024$ bytes = 1048576 bytes). CD'er brukes også som lagringsmedium for andre typer informasjon enn musikk og da er kapasiteten 650 Mb.

Mer moderne lydformater utnytter forskjellige teknikker for å komprimere datamengden slik at musikk som opptar 600 Mb på en CD kan reduseres til så

[2] Det er 16 bits som er den effektive informasjonsmengden for hvert sample, men i tillegg er det for hver måleverdi lagret 33 bits med ekstra informasjon som brukes til *feilkorreksjon*. Ved hjelp av denne informasjonen kan CD-spilleren sjekke om den avleste måleverdien er riktig og eventuelt korrigere den. Dette gjør CD´er forholdsvis robuste overfor riper og fettmerker, men hvis mye av feilkorreksjonsinformasjonen også er ødelagt vil vi allikevel kunne få støy under avspilling.

lite som 50 Mb i MP3-format (vanlig format på Internett).

Til telefoni er det ikke bruk for den samme kvaliteten som til musikk, så både ISDN- og GSM-telefoner kommuniserer ved hjelp av digital lyd som er samplet 8000 ganger i sekundet. Dessuten brukes bare 8 bits til å representere hvert sample (egentlig 12 bit som komprimeres til 8). Til sammen gir dette en informasjonsstrøm på 64000 bits pr. sekund hver vei under en vanlig telefonsamtale. Med et vanlig ISDN-abonnement får vi tilgang på to slike linjer slik at vi kan overføre to samtaler samtidig eller 128000 bits pr. sekund. Har vi et ISDN-modem kan vi altså i beste fall overføre 128000 bits pr. sekund hvis vi bruker begge linjene samtidig.

1.1 Lyd på datamaskin

Ut fra vår diskusjon så langt ser vi at musikken på en CD kan betraktes som to endelige tallfølger $\{a_n\}$ og $\{b_n\}$, en følge for hver kanal (musikken er i stereo). For å illustrere de enkleste prinsippene bak digital lydbehandling er det tilstrekkelig å se på en kanal, så vi vil anta at et digitalt lydsignal er gitt ved en endelig følge av N tall $\{a_n\}_{n=1}^N$, og i denne sammenhengen refererer vi ofte til en følge som et *signal*. Hvis lyden er hentet fra en CD-plate kan det være naturlig å se på hver a_n som et 16 bits heltall i intervallet -2^{15} og $2^{15}-1$. Men ved å dividere alle tallene i følgen med 2^{15} vil vi få konvertert til flyttall som ligger i intervallet $[-1,1]$, noe som kan være mer hendig ved beregninger.

Veien fra en gitt tallfølge $\{a_n\}$ til lyd er kort. Så og si alle moderne datamaskiner er utstyrt med elektronikk som kan behandle lyd (et lydkort) og høytalere. Vi bør derfor velge et programmeringsspråk som gir adgang til lydkortet via en eller annen funksjon, la oss kalle den `Play`. Når vi kaller opp denne funksjonen med følgen $\{a_n\}$ som parameter, vil lydkortet omdanne følgen til et elektrisk signal som varierer på samme måte som følgen. Dette signalet sendes til maskinens høytalere der det setter høytalermembranene i svingninger svarende til signalet. Disse svingningene setter lufta i bevegelse med det resultat at vi hører den aktuelle lyden.

Denne beskrivelsen av veien fra følge til lyd får det hele til å høres ganske greit ut, men fra et matematisk synspunkt er det i allefall en stor utfordring. Følgen vår $\{a_n\}$ består av tall som angir styrken på lydsignalet ved en del tidspunkter som er adskilt med et fast tidsintervall. Har vi for eksempel en samplingsrate på 8000 er avstanden mellom hver verdi $1/8000s = 0.000125s$. På den annen side er lyd et fenomen som er kontinuerlig i tid slik at lufttrykket varierer hele tiden, og ikke bare ved isolerte tidspunkter. På en eller annen måte må vi derfor 'fylle inn' verdier for lydsignalet mellom de verdiene som følgen gir. Dette kan gjøres på mange måter, og kvaliteten på lyden vi hører i høytalerene er svært avhengig

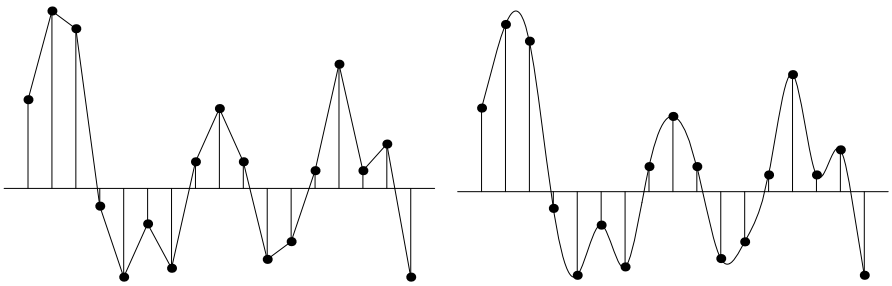


Figure 1: To forskjellige måter å konvertere et digitalt signal til et analogt. I (a) har vi trukket rette linjer mellom hver måleverdi, mens vi i (b) brukt tredjegrads polynomer mellom hver måleverdi.

av hvor godt vi gjetter på hvordan lyden er mellom verdiene vi har fra $\{a_n\}$. På den annen side er det ikke så kritisk hvordan dette gjøres hvis vi har høy samplingsrate siden to naboverdier da som regel er ganske lite. I en slik situasjon vil de ulike metodene som regel gi omtrent samme svar. I figur 1 har vi vist to måter å fylle inn mellomliggende verdier på.

Kontinuerlige lydsignaler kalles ofte *analoge* signaler i motsetning til digitale signaler, og det å fylle inn mellomliggende verdier kalles ofte digital-til-analog konvertering eller *DA-konvertering*. Den motsatte prosessen, det å danne et digitalt signal fra et analogt, er noe enklere, og ikke overaskende kalles dette ofte *AD-konvertering* (analog-til-digital konvertering).

Som regel er det vanskelig å kontrollere DA-konverteringen — den er implementert i elektronikken og vi må ta det vi får. Det vi har til rådighet er en funksjon som `Play`, og ikke minst den generelle regnekraften i datamaskinen.

Før vi kan avspille lyden gitt ved følgen $\{a_n\}$ må vi bestemme oss for en samplingsrate. Hvis vi for eksempel har 16000 verdier kan dette svare til en lyd som varer i 2 sekunder hvis samplingsraten er 8000, mens lyden bare vil vare i 1 sekund hvis samplingsraten er 16000. Samplingsraten er en parameter som vi må gi til funksjonen `Play` og som vi har full kontroll over. Vi kan derfor spille av våre 16000 verdier med ulike samplingsrater. Hvis lyden er generert med en samplingsrate på 8000 vil det høres riktig ut om vi avspiller med den samme samplingsraten. Hvis vi derimot spiller av med en samplingsrate på 16000 når lyden ble generert med en samplingsrate på 8000 vil alle ossillasjoner bli dobbelt så raske som opprinnelig, slik at tonehøyden vil fordobles ved avspilling. En lyd med en frekvens på 1000 Hz vil derfor bli til en lyd på 2000 Hz. På samme måte vil frekvensen bli redusert når samplingsraten reduseres.

Som vi så tidligere er samplingsraten på CD´er 44100, mens den for digital telefoni er 8000. For eksperimenter er det som regel lurt å bruke en samplingsrate på 8000 så slipper vi å arbeide med så store datamengder.

1.2 Filtrering av lyd

La oss nå anta at vi har et lydsignal $\{a_n\}_{n=0}^{N-1}$ som vi har lagret i en tabell av lengde N. Det som gjør kombinasjonen digital lyd og datamaskiner så spennende er at vi nå kan kombinere matematikk med datamaskinens regnekraft til å endre eller *filtrere* lyden. La oss for eksempel tenke oss at vi har fått tilgang på en digital utgave av et gammelt opptak med en artist som levde på første halvdel av 1900-tallet. Vi kan da forsøke å forbedre lydkvaliteten ved å fjerne noe av støyen som et slikt gammelt opptak uvegerlig vil være befengt med.

En annen utfordrende oppgave er å komprimere følgen slik at informasjonsmengden blir redusert til 1/12, slik som det gjøres i MP3-formatet, men uten at lydkvaliteten blir hørbart dårligere.

Her skal vi ikke forsøke oss på slike krevende oppgaver. I stedet skal vi se på noen enkle operasjoner vi kan gjøre med lyden.

Avspille lyden med forskjellige samplingsrater. Kanskje den enkleste måten å endre lyden på er å endre samplingsraten. Hvis vi har et lydsignal der samplingsraten var s ved opptak kan vi ved avspilling forsøke samplingsratene $s/2$, s og $2s$. Det vil endre en tone med frekvens f til en tone med frekvens lik henholdsvis $f/2$, f og $2f$.

Spille av lyden baklengs. Opp gjennom årene har det gått rykter om skjulte budskap på en del rockeplater. Ved å spille platen baklengs skal budskapene komme fram. Slikt er lett å sjekke hvis vi har en digital versjon av plata.

For å spille av lyden $\{a_n\}_{n=0}^{N-1}$ baklengs danner vi lyden $\{b_n\}_{n=0}^{N-1}$ der b_n er gitt ved

$$b_n = a_{N-1-n}, \qquad \text{for } n = 0, 1, \ldots, N-1.$$

Vi ser da at $b_0 = a_{N-1}$, $b_1 = a_{N-2}$ og så videre fram til $b_{N-2} = a_1$ og $b_{N-1} = a_0$. Lyden gitt ved følgen $\{b_n\}$ vil derfor være lyden gitt ved $\{a_n\}$ spilt baklengs.

Legge til støy. Vi skal ikke her gå inn på hvordan en kan fjerne støy fra en følge, men det å legge til støy er ikke så vanskelig. Nå er ikke støy noe entydig begrep, vi bruker det om all mulig uønsket lyd. Her mener vi med støy tilfeldig sus uten noen spesiell struktur. Det viser seg at denne typen støy kan vi få fram ved hjelp av tilfeldige tall.

La oss anta at vi har lydsignalet $\{a_n\}$, og at verdiene er flyttall i intervallet $[-1,1]$. For å legge til støy kan vi legge et tilfeldig tall i intervallet $[-0.1,0.1]$ til hver verdi a_n. Dette oppnår vi ved å danne en ny følge $\{b_n\}$ der b_n er gitt ved (i et Javalignende språk)

```
b[n] = a[n] + 0.2*(random()-0.5)
```

Siden `random()` gir et tilfeldig tall mellom 0 og 1 får vi et tilfeldig tall mellom -0.5 og 0.5 ved å trekke fra 0.5, og ved å multiplisere resultatet med 0.2 får vi et tilfeldig tall mellom -0.1 og 0.1. Vi kan selvsagt gjøre støyen sterkere eller svakere ved å endre faktoren 0.2 over.

Legge til ekko. Et ekko er bare en svakere kopi av den opprinnelige lyden med litt forsinkelse. Ved å variere tidsforsinkelsen kan vi få fram forskjellige effekter. Svært kort forsinkelse vil ikke oppfattes som ekko, men som en litt 'mykere' variant av den opprinnelige lyden, mens litt større forsinkelse (ca. 2 ms[3]) gir et naturlig ekko. Hvis vi skal måle forsinkelsen i millisekunder må vi kjenne samplingsraten. Tidsintervallet mellom to sampler er 0.125 ms ved en samplingsrate på 8000. En forsinkelse på 2.5 ms svarer derfor til en forsinkelse på 20 tidsintervaller. Dette oppnår vi ved å danne et nytt lydsignal $\{b_n\}$ ved

$$b_n = a_n + d\,a_{n-20}, \tag{1}$$

der d er en dempningsfaktor, for eksempel $d = 0.5$. Legg merke til at denne definisjonen av $\{b_n\}$ skaper problemer ved begynnelsen av følgen siden $n-20$ er negativ når $n < 20$, og vi har antatt at signalet $\{a_n\}$ starter med a_0. Mer presist bør vi derfor definere $\{b_n\}$ ved

$$b_n = \begin{cases} a_n, & \text{for } 0 \le n \le 19, \\ a_n + d\,a_{n-20}, & \text{for } 20 \le n \le N-1. \end{cases}$$

Hvis vi vil ha mer eller mindre forsinkelse, kan vi erstatte 20 med et annet tall. Vi må også huske på at samplingsraten påvirker forsinkelsen. Med en samplingsrate på 44100 må forsinkelsen være omtrent 110 sampler for å få en tidsforsinkelse på omtrent 2.5 ms.

Ved å variere tidsforsinkelsen kan vi få fram ulike effekter. Vi kan for eksempel la forsinkelsen variere periodisk mellom 10 og 30 ved å beregne $\{b_n\}$ ved

$$b_n = a_n + d\,a_{n-g_n}$$

[3] 1 ms står for 1 millisekund som er det samme som 0.001 s.

der g_n er gitt ved

$$g_n = 10\big(2 + \sin(\mu n)\big). \tag{2}$$

Vær oppmerksom på at g_n vanligvis ikke blir noe heltall, så vi bør runde av høyresiden til det nærmeste heltallet. Faktoren 10 kan selvsagt endres etter behov, og som nevnt over bør den økes hvis samplingsraten økes.

Parameteren μ i (2) er en konstant som bør velges nokså liten, ellers blir variasjonene for voldsomme. Hvis vi ønsker å få inn 5 svingninger pr. sekund og samplingsraten er 8000 så må μn variere over et intervall på 10π når n varierer over et intervall på 8000. Altså må vi ha $8000\mu = 10\pi$ eller

$$\mu = \frac{10\pi}{8000}.$$

Mer generelt ser vi at hvis vi vil ha ν svingninger pr. sekund når samplingsraten er s må vi velge

$$\mu = \frac{2\pi\nu}{s}.$$

Dempe diskanten. Fra stereoanlegg er vi vant til å ha muligheten til å kunne justere bass og diskant i musikken. For å gjøre dette ordentlig trenger vi en presis definisjon av frekvensbegrepet og en måte å måle frekvensinnholdet i et signal. Det skal vi ikke forsøke oss på her, men vi kan få til slike effekter med en intuitiv tilnærming til problemet. Vi begynner med å se hvordan vi kan dempe de høye frekvensene og dermed markere bassen bedre.

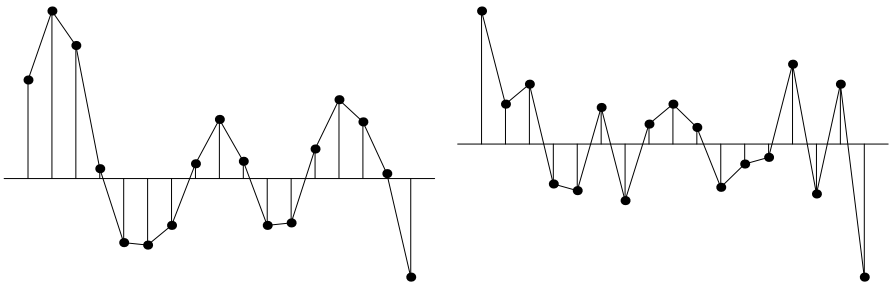
De høye frekvensene kommer fra de raskeste svingningene i lydsignalet. Om vi derfor gjør en operasjon på signalet som gjør svingningene mindre så får vi redusert innholdet av høye frekvenser og dermed diskanten i lyden. Et slikt filter kalles et *lavpassfilter*. En enkel måte å redusere svingningene på er danne et nytt signal $\{b_n\}$ ved å ta *gjennomsnitt* av noen nabosampler. For eksempel kan vi definere $\{b_n\}$ ved

$$b_n = \frac{a_{n-1} + a_n + a_{n+1}}{3}.$$

Som for ekko får vi problemer i endene av signalet, men det kan vi ordne med å bruke det opprinnelige signalet i endene,

$$b_n = \begin{cases} a_n, & \text{for } n = 0, \\ (a_{n-1} + a_n + a_{n+1})/3, & \text{for } 1 \le n \le N-2, \\ a_n, & \text{for } n = N-1. \end{cases}$$

Vi erstatter altså a_n med gjennomsnittet av a_n og de to naboverdiene og gir de tre verdiene like stor vekt i gjennomsnittet. Siden gjennomsnittet skal settes inn



(a) (b)

Figure 2: Glatting og framheving av kanter i et signal. I (a) har vi erstattet signalet i figur 1 med gjennomsnittssignalet gitt ved (3), mens vi i (b) har erstattet signalet i figur 1 med signalet gitt ved (4) som framhever kantene i signalet.

på plassen til a_n er det ikke urimelig å la a_n telle mer enn de to naboene. Det kan vi oppnå ved å bruke filteret

$$b_n = \begin{cases} a_n, & \text{for } n = 0, \\ (a_{n-1} + 2a_n + a_{n+1})/4, & \text{for } 1 \le n \le N-2, \\ a_n, & \text{for } n = N-1. \end{cases} \tag{3}$$

Effekten av dette på signalet i figur 1 er vist grafisk i figur 2, og vi ser ganske tydelig at kantene har blitt glattet ut litt.

For å få en penere dempning av diskanten kan vi bruke lengre gjennomsnittsfiltre, for eksempel

$$b_n = \begin{cases} a_n, & \text{for } 0 \le n \le 1, \\ (a_{n-2} + 4a_{n-1} + 6a_n + 4a_{n+1} + a_{n+2})/16, & \text{for } 2 \le n \le N-3, \\ a_n, & \text{for } N-2 \le n \le N-1. \end{cases}$$

Legg merke til at koeffisientene foran a'ene er hentet fra rad 4 i Pascals trekant, mens tallet vi dividerer med er summen av disse koeffisientene. Det viser seg at det å plukke koeffisienter fra en rad i Pascals trekant med like nummer generelt er en god måte å dempe diskanten på.

Dempe bassen. På samme måte som vi kan dempe diskanten kan vi dempe bassen og dermed framheve diskanten. En enkel måte å gjøre dette på er å snu annenhvert fortegn i koeffisientene vi brukte ved glatting. Hvis vi endrer filteret

i (3) på denne måten og anvender det på $\{a_n\}$ får vi det nye signalet $\{b_n\}$ gitt ved

$$b_n = \begin{cases} a_n, & \text{for } n = 0, \\ (-a_{n-1} + 2a_n - a_{n+1})/4, & \text{for } 1 \le n \le N-2, \\ a_n, & \text{for } n = N-1. \end{cases} \tag{4}$$

Figur 2 (b) viser resultatet av å gjøre denne operasjonen på signalet i figur 1. Vi ser at dette signalet er mer kantete enn både det opprinnelige signalet og det glattede signalet i figur 2 (a). Denne 'kantetheten' er det som representerer de høye frekvensene, eller diskanten, i det opprinnelige signalet. Et filter av denne typen, som bevarer de høye frekvensene, kalles et *høypassfilter*. Vi kan danne andre høypassfiltere ved å endre passende glattingsfiltere på samme måten.

1.3 Signalbehandling.

Her har vi skissert hvordan vi ved hjelp av litt matematikk kan gjøre operasjoner på lydsignaler. Slik filtrering er en del av feltet *signalbehandling*, og er en viktig del av grunnlaget for moderne teknologi. Selv om vi enkelt kan se prinsippene bak signalbehandling, er det å lage gode filtere som gjør akkurat det de skal en omfattende prosess som ofte involverer både matematikk og statistikk. I tillegg er det viktig med en god forståelse for hvordan vi oppfatter lyd. Ved for eksempel kompresjon av lyd utnyttes det faktum at dersom vi hører en lyd med en dominerende frekvens som samtidig inneholder en nærliggende frekvens som ikke er så kraftig, så registrerer ikke øret denne nabofrekvensen. Denne kan derfor fjernes fra signalet uten at vi hører nevneverdig forskjell, og det viser seg at det nye signalet kan lagres på en mer kompakt form en det opprinnelige.

Her har vi fokusert på lydsignaler, men det er svært mange typer signaler (følger) som kan behandles på tilsvarende måte. Noen eksempler er radiosignaler, radarsignaler, ultralyd og digitale bilder.

2 Lyd fra funksjoner

I seksjon 1 så vi på de grunnleggende prinsippene for digital lydbehandling, og i denne seksjonen skal vi se hvordan vi kan utnytte de velkjente matematiske funksjoner til å generere lyd. Kontinuerlige lydsignaler (i motsetning til digitale signaler) kalles ofte *analoge signaler*, og det å behandle slike signaler kalles ofte *analog signalbehandling*.

Vi husker at en følge som ossilerer vil generere lyd når den avspilles. En måte å generere slike ossilerende følger er å plukke verdier fra en ossilerende funksjon, og prototypen på ossilerende funksjoner er $\sin x$ (eller $\cos x$). I figur 3 har vi vist

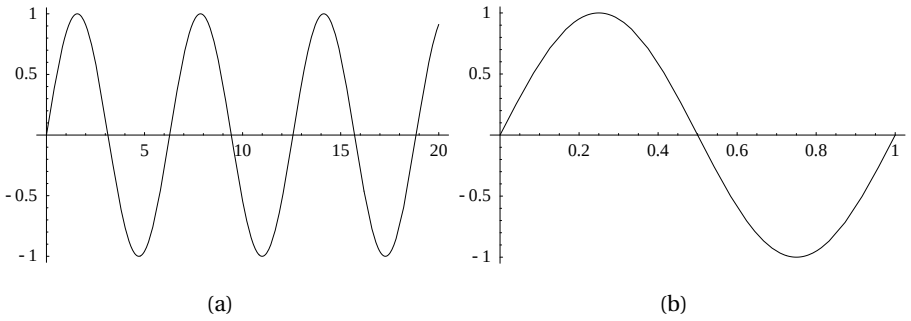


Figure 3: I (a) vises funksjonen $\sin t$ på intervallet $[0,20]$ mens (b) viser funksjonen $\sin 2\pi t$ på intervallet $[0,1]$.

et utsnitt av $\sin x$ hentet fra intervallet $[0,20]$ (husk at x er målt i radianer). Som vi vet ossilerer $\sin x$ jevnt mellom -1 og 1 og vi vet også at når vi har beveget oss over en intervallbredde på 2π vil $\sin x$ gjenta seg — den er en *periodisk* funksjon med *periode* 2π. I figur 3 ser vi at vi har fått med litt over 3 perioder av $\sin x$ siden $3 \cdot 2\pi \approx 18.9$ og vi har en intervallbredde på 20.

For å få lyd ut av en sinusfunksjon må vi ha kontroll på hvor mange ganger den ossilerer pr. sekund. Siden tiden nå skal løpe langs x-aksen erstatter vi x med t. Vi legger merke til at funksjonen $\sin 2\pi t$ inneholder en full periode av sinusfunksjonen når t varierer over intervallet $[0,1]$, se figur 3 (b). Hvis vi betrakter denne funksjonen over intervallet $[0,440]$ vil vi derfor få med oss 440 perioder av sinusfunksjonen. Disse ossilasjonene kan vi få inn på intervallet $[0,1]$ hvis vi bruker funksjonen $\sin 2\pi 440t$. For når t nå varierer over intervallet $[0,1]$ vil $440t$ variere over intervallet $[0,440]$ slik at vi får med oss 440 perioder av sinusfunksjonen. Siden vi tenker oss at t måles i sekunder har vi nå en funksjon som ossilerer 440 ganger pr. sekund, så hvis vi kan omdanne dette til lyd burde vi kunne høre denne funksjonen siden vi oppfatter som lyd alt mellom 20 og 20000 ossilasjoner pr. sekund.

En datamaskin kan bare håndtere digital lyd, så vi må konstruere en følge fra funksjonen vår. Men det er enkelt. La oss anta at vi bruker en samplingsrate på 8000. Da plukker vi bare 8000 jevnt fordelte verdier fra funksjonen $\sin 2\pi 440t$ og gir disse til `Play`-funksjonen i vår programmeringsomgivelse. Hvis vi kaller disse verdiene $\{a_i\}$ kan de beregnes ved

$$a_i = \sin(2\pi 440i/8000)$$

for $i = 0, 1, \ldots, 7999$. Når denne følgen avspilles med samplingsrate 8000 vil vi høre en lyd med frekvens 440 Hz. Denne tonen kalles *kammertonen* i musikk og

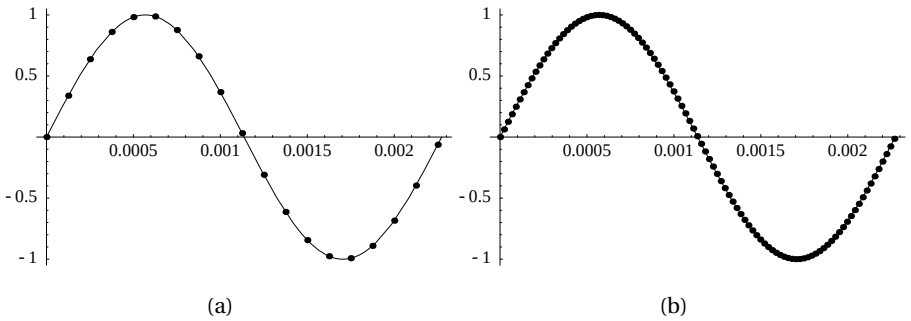


Figure 4: I (a) vises en periode av funksjonen $\sin 2\pi 440t$ samplet med 8000 verdier pr. sekund, og i (b) samples det med 44100 verdier pr. sekund.

har samme tonehøyde som summetonen i telefonen.

Hvis vi ønsker en høyere samplingsrate må vi plukke flere verdier fra hver periode. Med for eksempel en samplingsrate på 44100 må vi plukke 44100 verdier pr. sekund. I vårt tilfelle gir det en følge $\{\hat{a}_i\}$ der a_i er definert ved

$$\hat{a}_i = \sin(2\pi 440 i/44100)$$

for $i = 0, 1, \ldots, 44099$. Effekten av å øke samplingsraten er altså at vi tar med flere verdier og på den måten får en bedre representasjon av funksjonen. I figur 4 har vi vist hvor tett samplene ligger på en periode av funksjonen når samplingsraten er 8000 (i (a)) og 44100 (i (b)).

Hvis vi mer generelt ønsker en lyd med frekvens f oppnår vi det ved å plukke verdier fra funksjonen $\sin 2\pi f t$. Som nevnt over er det slik at hvis vi bruker samplingsrate s får vi ikke med oss høyere frekvenser enn $s/2$. Frekvensen f bør derfor ikke overstige $s/2$, men det kan være morsomt å eksperimentere med å overstige denne grensen.

De trigonometriske funksjonene er prototypene på ossilerende funksjoner, og definisjonen på en 'ren' tone med frekvens f er den som genereres av funksjonen $\sin 2\pi f t$ (eller $\cos 2\pi f t$; begge genererer den samme lyden). Men det er andre funksjoner som også kan generere lyder med gitt frekvens, selv om vi da ikke får 'rene' toner. For eksempel kan vi bruke en 'firkantpuls' der en periode ser ut som i figur 5 (a). Denne funksjonen er definert ved

$$P_0(t) = \begin{cases} 1, & \text{når } 0 \le t < 1/2, \\ -1, & \text{når } 1/2 \le t < 1. \end{cases}$$

Funksjonen er altså diskontinuerlig, men det er ikke tvil om at den ossilerer en gang. For å lage en lyd med frekvens 440 Hz fra denne funksjonen bruker vi

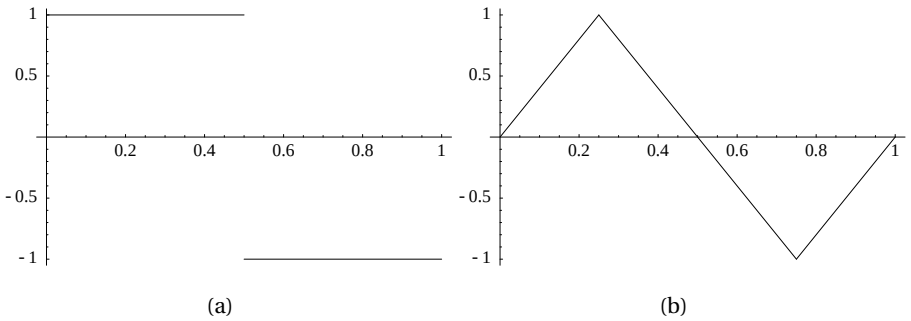


Figure 5: Plott av firkantpuls (a) og trekantpuls (b).

samme oppskrift som over og presser 440 perioder inn på et sekund. Det oppnår vi ved å bruke funksjonen $P_0(440t)$. Spiller vi av denne er det ikke tvil om at den har en grunnfrekvens på 440 Hz, men i tillegg hører vi noe de fleste vil synes er ubehagelig lyd, på grensen til støy. Matematisk kommer dette av at diskontinuitetene gir funksjonen noen kraftige, ekstra frekvenser, i tillegg til grunntonen på 440 Hz.

I figur 5 (b) har vi en annen periodisk funksjon som kan brukes som byggekloss for å lage lyd. Denne kalles en trekantpuls og er gitt ved

$$P_1(t) = \begin{cases} 4t, & \text{når } 0 \le t < 1/4, \\ 2-4t, & \text{når } 1/4 \le t < 3/4, \\ 4t-4, & \text{når } 3/4 \le t < 1. \end{cases}$$

Igjen kan vi danne en lyd med frekvens f ved å sample fra funksjonen $P_1(ft)$. Lyder generert fra P_1 er mer behagelige enn de som produseres fra P_0 siden vi ikke har diskontinuiteter. Men P_1 er ikke så glatt og pen som sinusfunksjonen (den deriverte av P_1 er diskontinuerlig) så den oppfattes av de fleste som noe skarpere.

Ved å gjenta andre funksjoner f ganger pr. sekund kan vi få fram andre lydkvaliteter med frekvens f og i prinsippet lyden fra alle mulige musikkinstrumenter. Men merk at dette er bare prinisipielt, i praksis vil det være vanskelig å modellere musikkinstrumenter godt på denne måten.

2.1 Modulasjon

De funksjonene vi har sett på så langt genererer en fast tone med en gitt frekvens, men med forskjellig kvalitet, omtrent slik forskjellige musikkinstrumenter kan produsere den samme tonen. For å generere lyd som inneholder informasjon

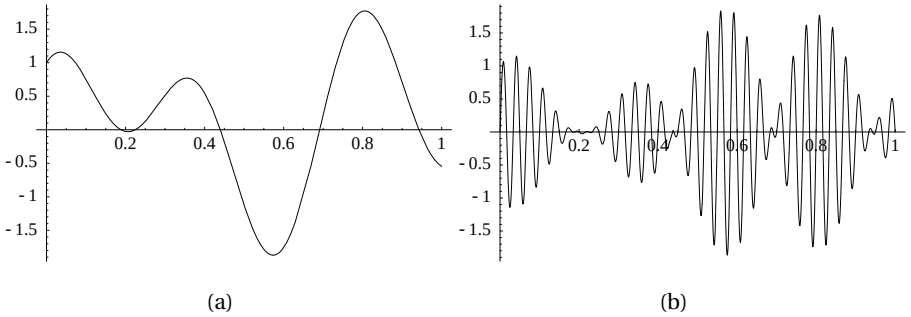


Figure 6: Funksjonen i (a) er i (b) multiplisert med $\sin 2\pi 30t$ for å illustrere amplitudemodulasjon.

må en bruke mer sofistikerte teknikker. De fleste bruker fremdeles analoge radioer der lydsignalene blir overført som kontinuerlige funksjoner. Utfordringen består i å få kodet et gitt lydsignal ved hjelp av kontinuerlige funksjoner slik at det kan overføres ved hjelp av elektromagnetiske bølger, siden slike bølger kan spres over store avstander før de plukkes opp av en antenne. (Det fungerer jo ikke så bra å sette en høytaler på en fjelltopp for å spre lyd!) FM-radio er basert på såkalt *frekvensmodulasjon* mens AM-radio er basert på *amplitudemodulasjon*.

Ved amplitudemodulasjon gis hver radiostasjon en fast sinusfunksjon med frekvens i området 150 kHz til 1500 kHz.[4] Dette svarer til senderfrekvensen, og det elektromagnetiske signalet som sendes ut har denne grunnfrekvensen. Et lydsignal som skal overføres legges inn ved å multiplisere denne grunnleggende sinusfunksjonen med en enkel variant av signalet som skal overføres. Hvis for eksempel lyden er gitt ved en funksjon $g(t)$ som sier hvordan lufttrykket skal variere og senderfrekvensen er 600 kHz vil signalet som sendes ut være

$$G(t) = C_1\big(1 + C_2 g(t)\big)\sin(2\pi 600000t), \tag{5}$$

der C_1 og C_2 er to konstanter som tilpasses signalet og sendertypen. I utgangspunktet svinger sinusfunksjonen mellom -1 og 1 — vi sier at utslaget eller *amplituden* er 1. Vi ser at i forhold til dette er amplituden i funksjonen $G(t)$ i (5) justert med funksjonen $C_1\big(1 + C_2 g(t)\big)$ som involverer lydsignalet som skal overføres. Med andre ord vil amplituden variere med tiden og med $g(t)$ — det er dette som har gitt opphav til begrepet amplitudemodulasjon. For å plukke opp lydsignalet må mottageren inneholde elektronikk for å skille $g(t)$ fra $G(t)$.

Figur 6 (a) viser et plott av et lite utsnitt av et amplitudemodulert signal. For lettere å vise hva som skjer når en sinusfunksjon multipliseres med en annen

[4] 1 kHz er det samme som 1000 Hz.

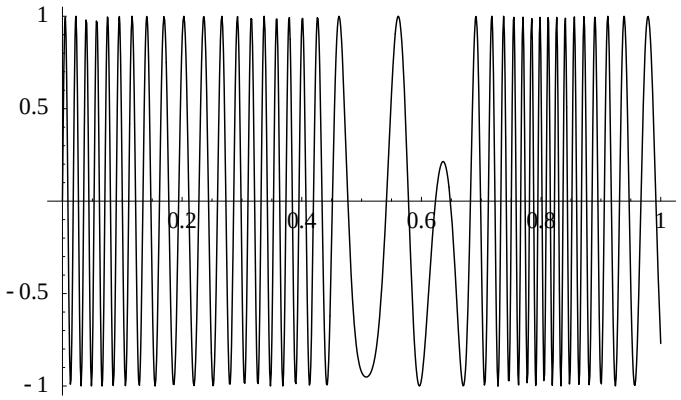


Figure 7: Funksjonen $g(t)$ fra figur 6 (a) kodet ved hjelp av frekvensmodulasjon.

funksjon er funksjonen som vises i (b) bare produktet av funksjonen i (a) og $\sin 2\pi 30t$.

FM-radio bruker et litt annet prinsipp for å overføre lyd. Igjen gis hver radiostasjon en fast frekvens ω, men nå i området 87.5 MHz til 108 MHz.[5] Lydsignalet $g(t)$ overføres nå ved å overføre funksjonen

$$G(t) = A\sin\Big(2\pi\omega t + 2\pi k\int_0^t g(s)\,ds\Big), \tag{6}$$

der A og k er passende konstanter. Signalet $g(t)$ brukes med andre ord til å justere frekvensen i steden for amplituden, derav navnet frekvensmodulasjon.

AM- og FM-modulasjon kan brukes til å lage spennende lyder, og mange av lydeffektene i synthesizere er basert på modulasjon. Mye av dette kan vi også ganske enkelt få til på en datamaskin. For eksempel kan vi mulitplisere $\sin 2\pi 440t$ med funksjonen e^{-t} eller mer generelt e^{-at} der a er en positiv konstant for å dempe lyden. Lydsignalet

$$e^{-t}\sin 2\pi 440t$$

vil derfor være en ren tone med frekvens 440 Hz som dør ut med tiden.

Vi kan også danne lyd ved hjelp av frekvensmodulasjon. Funksjonen

$$\sin(2\pi 440 + 20\sin 5t) \tag{7}$$

gir en lyd med en frekvens som varierer rundt 440 Hz ±20 Hz. Funksjonen

$$b(t) = e^{-\alpha t}\sin\big(2\pi f_c t + be^{-\beta t}\sin(2\pi f_m t)\big) \tag{8}$$

[5] 1 MHz er det samme som 10^6 Hz.

er basert på en kombinasjon av amplitude og frekvensmodulasjon. Ved å velge konstantene som $f_c = 80$ Hz, $f_m = 112$ Hz, $\alpha = 0.06$, $\beta = 0.09$ og $b = 4$, får vi en klokkelignende lyd. Når denne lyden avspilles bør tida løpe en stund, i allefall 10 sekunder og gjerne mer. Ved å eksperimentere med parametrene i (8) er det mulig å få fram et vidt spekter av lyder.

2.2 Musikkskalaer og matematikk

Vi har nå tilgjengelig funksjoner som gir oss toner med forskjellig frekvens, både rene toner og toner med litt 'skarpere kanter'. Hvordan kan disse kombineres for å produsere musikk?

I musikk opererer en ikke med toner med vilkårlig frekvens. For at musikken skal høres 'pen' ut må de ulike tonene ha frekvenser som står i et visst forhold til hverandre. Disse forholdene varierer innen ulike kulturer, men i vår vestlige verden er tolvtoneskalaen dominerende. Utgangspunktet for en slik skala er at når en tone svinger dobbelt så fort som en annen så høres de to tonene like ut selv om de er forskjellige — vi sier at de ligger en *oktav* fra hverandre. På et standard piano har den laveste tonen (A'en lengst til venstre) en frekvens på 27.5 Hz. Når vi beveger oss mot høyre ligger hver A en oktav over den foregående, og totalt er det på et standard piano åtte A´er med frekvenser (målt i Hz),

$$27.5,\ 55,\ 110,\ 220,\ 440,\ 880,\ 1760,\ 3520.$$

Den siste A'en er ikke den høyeste tonen på pianoet, det ligger 3 tangenter til høyre for denne slik at den høyeste tonen er en C. Tilsammen gir dette 88 tangenter som også er det vanlige på andre tangentinstrumenter av full størrelse. Mellom to A'er som er naboer ligger det 11 tangenter (både hvite og svarte). Hver av disse har igjen slektninger som ligger en oktav over eller under i frekvens. Innenfor en oktav er det derfor 12 forskjellige toner, og frekvensen til disse er jevnt fordelt i den forstand at forholdet i frekvens mellom to nabotangenter er konstant.

Hvis vi oversetter dette til matematikk så ser vi at de to funksjonene $\sin 2\pi f t$ og $\sin 2\pi 2 f t$ skiller seg med en oktav. Vi har totalt 12 toner i en oktav, og den trettende tonen ligger altså en oktav over den første og har dobbel frekvens av denne. La oss betegne de 13 frekvensene med f_0, f_1, f_2, ..., f_{12}, der $f_{12} = 2f_0$. De mellomliggende tonene skal ha frekvenser som er jevnt fordelt mellom f_0 og f_{12} slik at forholdet mellom nabotoner skal være konstant. Hvis vi kaller dette

forholdet c så skal vi altså ha

$$\begin{aligned} f_1 &= c f_0, \\ f_2 &= c f_1 = c^2 f_0, \\ f_3 &= c f_2 = c^3 f_0, \\ &\vdots \\ f_i &= c f_{i-1} = c^i f_0, \\ &\vdots \\ f_{12} &= c f_{11} = c^{12} f_0. \end{aligned}$$

Siden vi i tillegg har $f_{12} = 2f_0$ så ser vi fra den siste av disse ligningene at $c^{12} = 2$ eller $c = 2^{1/12} \approx 1.0594$. Tar vi utgangspunkt i A´en med frekvens 440 Hz kan vi derfor representere alle tonene på pianoet ved funksjonene

$$\sin 2\pi 440 c^i t, \qquad i = -48, -47, \ldots, 38, 39.$$

Spesielt ser vi at tonene innenfor oktaven som går fra 220 Hz til 440 Hz har frekvensene

$$\begin{gathered} 220, 233.08, 246.94, 261.63, 277.18, 293.67, \\ 311.13, 329.63, 349.23, 369.99, 391.00, 415.31, 440. \end{gathered} \tag{9}$$

For å spille av en av disse tonene trenger vi bare sample funksjonen $\sin 2\pi f_i t$ der f_i er en av frekvensene over, og sende resultatet gjennom `Play`. Hvis vi ønsker en annen type lyd kan vi bruke funksjonene P_0 eller P_1 over, eller eventuelt en annen funksjon.

Frekvensene listet opp i (9) er alle tonene mellom de to A'ene, og ikke alle disse hører hjemme i tonearten A-dur. A-dur inneholder frekvensene (navnet på tonene over)

A	H	C♯	D	E	F♯	G♯	A
f_0	f_2	f_4	f_5	f_7	f_9	f_{11}	f_{12}

Hvis disse frekvensene avspilles i rekkefølge er resultatet en pen A-dur skala. Vi har også A-moll skalaen som er gitt ved

A	H	C	D	E	F	G	A
f_0	f_2	f_3	f_5	f_7	f_8	f_{10}	f_{12}

(det fins også andre typer moll-skalaer).

Ved å begynne på andre frekvenser enn f_0, men bruke samme sprangfaktor mellom frekvensene, får vi fram de andre dur- og moll-skalaene. Begynner vi for eksempel på f_3 får vi henholdsvis C-dur og C-moll.

A-dur skalaen over er et eksempel på en *temperert* skala og er et kompromiss for å få det hele til å gå opp slik at vi for eksempel på et piano kan spille ikke bare i A-dur, men også i C-dur og F-dur. Frekvensene til den midterste oktaven i den tempererte A-dur skalaen er altså

A	H	C$^\sharp$	D	E	F$^\sharp$	G$^\sharp$	A
220	246.94	277.18	293.67	329.63	369.99	415.31	440

Det fins også andre skalaer der forholdet mellom frekvensene er basert på at det innen en oktav skal være tolv toner med et fast forhold mellom frekvensene. En *renstemt* dur-skala er, som navnet tilsier, ingen kompromisskala, og er basert på at frekvensene til de 8 tonene i skalaen har følgende forhold til den første tonen,

$$1,\ \frac{9}{8},\ \frac{5}{4},\ \frac{4}{3},\ \frac{3}{2},\ \frac{5}{3},\ \frac{15}{8},\ 2.$$

Dette vil gi en A-dur skala med frekvenser

A	H	C$^\sharp$	D	E	F$^\sharp$	G$^\sharp$	A
220	247.5	275	293.33	330	366.67	412.5	440

Som vi ser er ikke forskjellen så stor mellom den renstemte og den tempererte skalaen, men forskjellen er mer enn stor nok til at to instrumenter som er stemt etter hver sin skala ikke vil lyde bra sammen.

Forholdene som danner grunnlaget for den renstemte skalaen har sitt utspring i den *pytagoreiske* skalaen som ble utviklet av pytagorerne. Den pytagoreiske skalaen ble senere justert litt av Ptolemeus og ble dermed til den renstemte skalaen over. Forholdene 1, 9/8, 4/3, 3/2 og 2 er felles for begge de to skalaene og er 'naturlige' i den forstand at disse tonene enkelt kan produseres på et enstrenget instrument.

Problemet med den renstemte skalaen er at den er 'helt riktig' i A-dur (eller den duren som renstemmes), men ikke så god i andre tonearter. Siden det tar såpass lang tid å stemme et piano er det vanligvis stemt temperert siden det gir et godt kompromiss for alle tonearter. Strykeinstrumenter kan derimot godt bruke renstemte skalaer siden de er raske å stemme og musikeren selv har kontroll på nøyaktig hvilken tonehøyde som skal brukes.

I musikk er det sjelden at enkelttoner lyder alene. Som regel settes flere toner sammen slik at vi får *harmonier* eller *akkorder*. Teorien for dette kalles *harmonilære* og denne læren kan også studeres ved hjelp av matematikk. Dette skal vi ikke gå inn på her, men vi skal vise hvordan vi kan generere en A-dur akkord.

En slik akkord får vi ved å spille de tre tonene A, C$^\sharp$ og E samtidig. Hvis vi tar utgangspunkt i A´en med frekvens $f_0 = 220$ Hz skal vi altså spille de tre tonene med frekvens f_0, $c^4 f_0$ og $c^7 f_0$. Dette oppnår vi ved å avspille funksjonen

$$\sin 2\pi f_0 t + \sin 2\pi c^4 f_0 t + \sin 2\pi c^7 f_0 t.$$

Ved å variere f_0 får vi fram andre dur akkorder. Moll-akkorder får vi når den midterste frekvensen endres fra $c^4 f_0$ til $c^3 f_0$.

Oppgaver

Oppgave 1. Gjør eksperimenter med de ulike typene filtrering som er beskrevet i seksjon 1.2. Bruk både musikk, tale og evetuelt kunstige lyder i eksperimentene.

Oppgave 2. I denne oppgaven skal vi se på en enkel differensligning for å generere lyd som minner om et strengeinstrument. Metoden kalles Karplus-Strong algoritmen og den ble oppdaget i 1979 av en forsker, Kevin Karplus, og en student, Alexander Strong, på Stanford University i USA.

Metoden produserer særlig gode gitar-liknende lyder. Til å være så enkel og beregningseffektiv gir den utrolig god lyd. Differenseligningen er gitt ved

$$x_n - \frac{1}{2}(x_{n-p} + x_{n-p-1}) = 0. \tag{10}$$

Her er p et positivt heltall som bestemmer frekvensen til lyden. Hvis vi for eksempel bruker en samplingsrate på 44100 og en ønsker en tone med frekvens 440 Hz, må P være $P = 44100/440 = 100,22 \approx 100$. (På grunn av avrunding til heltall får vi altså en frekvens på 441 Hz steden for 440 Hz, men det fins en variant av algoritmen som forbedrer dette.) Vi legger ellers merke til at for $p = 1$ så er ligning (10) identisk med ligningen (**??**) som vi simulerte i seksjon **??**.

Vi ser at differensligningen (10) er av $p+1$te orden så vi trenger $p+1$ startverdier x_0, x_1, x_2, ..., x_p. Når disse er gitt kan vi generere nye verdier fra formelen

$$x_n = \frac{1}{2}(x_{n-p} + x_{n-p-1}).$$

For å få fram ulike toner bruker vi forskjellige startverdier.

a) Velg en passende samplingsrate og bruk verdiene $x_0 = 1$ og $x_1 = x_2 = \cdots = x_p = 0$ som startverdier og generer lyden med forskjellig frekvens. Husk at frekvensen er gitt ved s/p der s er samplingsraten.

b) Bruk tilfeldige tall mellom $[-1,1]$ som startverdier i steden og gjenta forsøket. Bruken av tilfeldige tall som startverdi har den fordelen at lyden blir mer realistisk siden den ikke er akkurat den samme hver gang.

c) Prøv andre startverdier som kan gi spennende lyder. Husk at startverdiene må ossilere for at det skal bli noe lyd.

d) Forsøk å endre differensligningen litt og se hvordan det påvirker lyden.

Oppgave 3. I denne oppgaven skal vi se litt på differensligninger der den karakteristiske ligningen har to komplekse røtter.

a) Gjør en numerisk simulering av ligningen

$$x_n = \frac{x_{n-1}}{2} - x_{n-2}, \qquad x_0 = 0,\, x_1 = 1.$$

Ser løsningen ut til å gå mot 0 eller ∞ når n blir stor eller forblir løsningen begrenset, men ulik 0?

Spill av lyden som løsningen representerer (generer nok verdier til ett sekund med lyd).

b) Verifiser at observasjonen du gjorde i (a) om hva som skjer med x_n når n blir stor faktisk er riktig.

c) La oss nå se på den generelle ligningen

$$x_n + bx_{n-1} + cx_{n-2} = 0 \tag{11}$$

der b og c er reelle tall, men vi antar at de er valgt slik at begge røttene i den karakteristiske ligningen er komplekse.

Finn et uttrykk for tallverdien og argumentet til røttene i den karakteristiske ligningen.

Som startverdier bruker vi i resten av oppgaven $x_0 = 0$ og $x_1 = 1$, slik som i (a).

d) Velg verdier av b og c slik at løsningen av (11) gir en lyd med fast volum og høy frekvens.

e) Velg verdier av b og c slik at løsningen av (11) gir en lyd med fast volum og lav frekvens.

f) Velg verdier av b og c slik at løsningen representerer en lyd med avtagende volum. Forsøk å få til både lav og høy frekvens.

Oppgave 4.

a) Skriv et program som genererer en lyd på 440 Hz med utgangspunkt i de tre funksjonene P_0, P_1 og sin, slik som beskrevet i teksten over. (Funksjonene P_0 og P_1 kan programmeres ved hjelp av `if`-tester.)

b) Bruk samplingsrate 8000 og eksperimenter med å spille av lyder med frekvens i nærheten av 4000, altså halvparten av samplingsraten. Forsøk å danne deg et bilde av hva som skjer i det du passerer grensen på 4000.

c) Finn andre funksjoner enn P_0, P_1 og sin som kan brukes til å generere spennende lyder.

Oppgave 5. Ta utgangspunkt i funksjonen gitt ved (8) og varier parametrene slik at du får fram ulike 'pene' lyder.

Oppgave 6.

a) Skriv et program som spiller av de 8 tonene i A-dur skalaen. Bruk en samplingsrate på 8000 og la hver tone vare i 0.4 s med en pause på 0.1 s mellom hver tone.

b) Skriv et program som lar f_0 gjennomløpe de 13 frekvensene svarende til tonene i oktaven fra fra 220 Hz opp til 440 Hz, og så for hver verdi av f_0 spiller de 8 tonene i den tilsvarende dur-skalaen.

c) Skriv et program som først spiller en A-dur akkord i den tempererte skalaen og deretter en A-dur akkord i den renstemte skalaen.

d) Skriv et program som først spiller en A-moll akkord i den tempererte skalaen og deretter en A-moll akkord i den renstemte skalaen.