

Dækker Spor 3 i LYD-Kit materialet.

Slide stak er tænkt som inspiration til lærer – ikke som et færdigt forløb overfor eleverne.

Knytter an til bølgelære i fysik (Ex. Kapitel 3 om bølger og Kapitel 4 om lyd i Orbit B STX, 2017 1. udgave)

Er tænkt som et supplement til teori og eksisterende forsøg med rør og strenge i den fysiske verden.

Et digitalt supplement, som man kan bygge videre med i Spor 4.

Undervejs i slide stak er de flere steder med klikbare links på diverse billeder. Det er så nævnt i noterne. Links er også samlet sidst i stakken, så man kan dele dem på anden vis – ex. som hjemmeopgaver/forberedelse.

Version 15-Mar-2022:

Rettet til frigivelse.

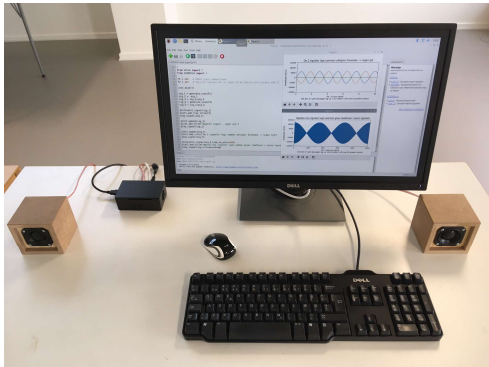
Version 09-Mar-2022:

Baggrundsmateriale og beskrivelser fra OneNote Notesbog ført over i slide stak, så det lettere kan deles/distribueres.

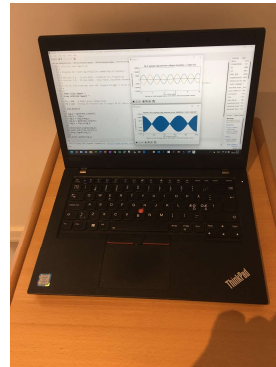
lettere kan deles/distribueres.

Spor 3: Intro – setup

Projekt støttet af:
midt
regionmidtjylland



Eller



2 Højtalere
som kan
flyttes.

Det væsentlige materiale ligger i kodeeksemplerne til Spor3.

Koden kan køre på både PC/MAC og Raspberry PI (sammen med forstærker board og højtalere).

Bruger man PC/MAC skal man sikre sig at man har nogle højtalere der fungerer i forhold til at demonstrere eksempelvis destruktiv interferens. Det vigtige her er at kunne placere dem tæt, og det vil hjælpe på effekten hvis der kun er én driverenhed. Men prøv det af inden det bruges for at se om effekten er tydelig nok. Interferens til eksempelvis stødtoner fra 2 kanaler fungerer fint i et normalt stereo setup – så her kan fast monterede højtalere sagtens bruges.

Spor 3 indeholder basale kode eksempler på hvordan man laver sinus kurver til lyd og plots, så eleverne kan eksperimentere med at se og høre dem.

De kan lave flere og addere dem - og dermed få nye lyde - og herigennem få en basis forståelse for additiv syntese. Eksemplerne kan også bruges til at forklare om fase og forskellen på at addere 2 kurver i modfase i mono (nul) og at lade 2 kanaler (stereo) afspille i modfase og så lade additionen ske i luften (mediet). Tilsvarende med stødtoner (beats).

De kan lave en stribe lyde med overtoner og bygges mere komplekse lyde. Med alle

overtoner (partialtoner) henholdsvis kun de ulige kan det kobles til fysik om stående bølger i strenge og åbne og halvåbne rør, hvilket er basis for additiv syntese.

Man kan lave de samme ting i andre værktøjer, men ideen her er at det så leder videre til Spor 4, hvor eleverne får mulighed for at sensor input og den basale viden om additiv syntese sammen med basal læring omkring musik og dermed kan bygge et digitalt musik instrument som de kan interagere med.

Alle Spor i LYD-Kit benytter Thonny som udviklingsmiljø (IDE) og Python som kode, hvilket samlet giver eleverne en platform til at 'bygge ting' med kode, sensorer (Micro:Bit) og eventuelt et fysisk forstærker modul og højtalere. Python koden kan afvikles i andre IDE'er end Thonny og fungerer på tværs af operativsystemer og maskiner (PC/Mac/Raspberry PI). Tilhørende kodeeksempler ligger i folderstruktur sammen med slide stak. Til Raspberry PI findes et image med alle koderne lagt ind på forhånd.

Spor 3: Intro - struktur

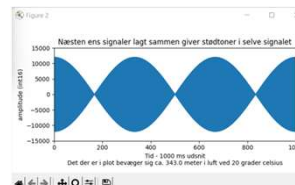
Lukket bog: En række program eksempler, hvor man ved at ændre i parametre kan lave eksperimenter med lyd.

```
from LKlib import *
from LKSELib import *

init_mixer()

sig_1 = generate_sine(440)
sig_2 = generate_sine(443)
sig_3 = sig_1+sig_2

plot_signal(sig_3,time_to_plot=1000)
play_signal(sig_3,forever=True)
```



Åben bog: Et funktionsbibliotek "LKlib.py" man kan kikke ind i for at dykke dybere i den digitale side.

```
def generate_sine(freq,amp_max=6000):
    t = np.arange(0,DURATION,1/SAMPLERATE)
    sine = (amp_max*np.sin(freq*2*np.pi*t)).astype(np.int16)
    return sine

def play_signal(mono_signal,vol_l=1,vol_r=1,forever=False):
    sound = generate_stereo_sound(mono_signal)
    channel = play_sound(sound,vol_l,vol_r,forever)
    if forever == False:
        time.sleep(DURATION+0.2)
    return channel
```

```
# -----
# Funktion der genererer og returnerer et sinus signal med den frekvens der angives som argument.
# Ex genereres ren sinus tone med frekvensen 440 (kammertonen) med:
#   kammertonen = generate_sine(440)
# hvor kammertonen så kan bruges i play_signal(kammertonen) efterfølgende for at høre den afspillet.
#
# Parametre:
# - freq      : Den frekvens man ønsker signal lavet i.
# - amp_max   : Amplituden. Den største værdi signalet skal have.
#               Kan udelades og så vil den bruge default værdi på 6000. Men man kan give et argument som har en anden værdi
#               for højere eller lavere amplitude (signal højde - volume/loudness - relativt til de 16bit som lydkortet kan håndtere)
#               Hvis man efterfølgende begynder at lægge signaler sammen skal man sikre sig at man ikke får værdier
#               der overstiger +/- 32,767. (signed 16 bit integer - som er det lydkortet kan håndtere)
#               Der er IKKE noget der går i 'stykker' hvis det sker men det kommer til at lyde mærkeligt. Signalet bliver
#               digitalt klippet af - men prøv det endelig for at høre effekten.
#
```

Den kode som følger med Spor 3 er organiseret sådan at man kan bruge dem 'Lukket bog' og 'Åben bog'

Lukket bog (blackbox):

Bruge eksemplerne til at demonstrere/eksperimentere med lyd fænomener som knytter an til fysik. Man kommer langt med at ændre i parametre i eksisterende eksempler. Kan derfor bruges i fysik til at supplere andre forsøg omkring lyd.

I 'lukket bog' delen er fokus på at kunne generere simple signaler med en enkelt linie kode, som gør det let at kunne addere flere signaler. Det at kunne addere signaler bruges til at sammensætte og efterligne overtoner etc.

Åben bog (whitebox):

Kikke dybere ind i koden og eksempelvis se hvordan den digitale sinus genereres og afspilles. Der kan man koble til diskret matematik. Man kan koble til funktioner i matematik/fysik. Man kan koble til digital lyd/el-lære omkring A/D konvertering, interpolation etc.

Koden/materialet giver mulighed for at lave tværfaglige projekter, hvor man for eksempel inddrager informatik/programmerings lærer og kan vise kobling til både matematik og fysik.

BEMÆRK:

Lklib.py og LKSElib.py skal ligge i direktoriet, hvor programeksemplerne/de nye programmer man evt. laver ligger.

Man kunne placere det i biblioteks Path under Python som et rigtigt bibliotek, men det er valgt at gøre det sådan her da det understøtter åben bog tilgangen, hvor man kan kikke ind i funktionerne hvis man ønsker.

Forklaring af de enkelte funktioners parametre er beskrevet i filen som kommentarer.

Åbn Lklib.py i Thonny for at være orienteret. Der er også en række funktioner der bruges i Spor 2 og 4 – til at kommunikere med Micro:Bitten på den serielle kanal.

I Spor 3 er der IKKE koblet til Micro:Bit og sensorer. Spor 3 er alene fokuseret på at generere lyde med sinus.

Spor 3: Intro – filoversigt

📄 LKlib.py
📄 LKS3-enhedscirkel-plot.py
📄 LKS3-Ex1-sum-signal.py
📄 LKS3-Ex2-interferens-luft.py
📄 LKS3-Ex3-Overtoner-ulige.py
📄 LKS3-Ex4-Overtoner-alle.py
📄 LKS3-Ex5-Overtoner.py
📄 LKS3-Ex6-Obo.py
📄 LKS3-Ex7-FormEnvelope.py
📄 LKS3-Ex8-Sweep.py
📄 LKS3-Ex9-Freq-Modulation.py
📄 LKS3-Ex10-Amplitude-Modulation.py
📄 LKS3-Ex11-Freq-og-Amp-Modulation.py
📄 LKS3-simple-7key-piano.py
📄 LKSEElib.py

Lklib og LKSEElib er biblioteksfunktioner der bruges af de øvrige eksempler.

LKS3-enhedscirkel-plot.py er til at forklare sinus i det digitale univers og til at bygge bro til en åben bog snak om grundfunktioner i LKLib.

Resten er eksempler til at lave eksperimenter/demonstrationer ud fra.

LKS3-simple-7key-piano.py er eksempel hvor man kan lave et simpelt klaver på tasterne a,s,d,f,g,h,j og selv bestemme/generere lydene rent digitalt.

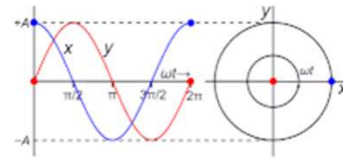
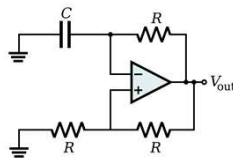
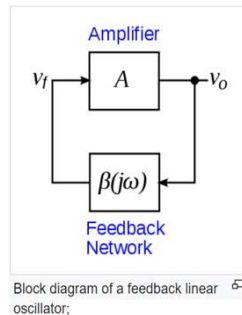
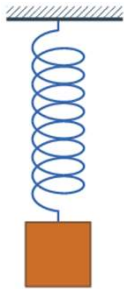
Oversigt over filerne. Eleverne skal kun have de relevante filer i forhold til det forløb man laver.

Obo eksemplet skal ikke udleveres, men tjener som inspiration til at eleverne selv kan bygge en syntetisk lyd ud fra en frekvens analyse på en 'rigtig' lyd.

BEMÆRK: At det kun er en simpel summering af grundtone og overtoner der benyttes her. Det resulterer i en 'kedelig' og monoton lyd. Det kan bruges til at tage en snak om:

- Det ikke er helt ligetil at efterligne den virkelige verden.
- I den virkelige verden er der varians over tid. Frekvens og amplitude varierer lidt..
- Man kan gøre noget med modulation og filtre – signal behandling, som vi ikke går dybt ind i her; men som er et interessant felt at studere dybere. Og i øvrigt benyttes i mange andre sammenhænge end lyd.

Spor 3: Harmonisk svingning - sinus



$$y(t) = A \sin(2\pi ft + \varphi) = A \sin(\omega t + \varphi)$$

Harmonisk svingning (Oscillator)
Mekanisk, elektrisk, matematisk, grafisk.

Harmonisk svingning (Oscillation)

Mekanisk (uden dæmpning). Pendulet laver ikke lyd, men det er stadig en harmonisk svingning, så det er mere universelt, men vi dykker bare ned i det med lyd som genstandsfelt.

Elektrisk – analog elektronik der kan generere en harmonisk svingning med spænding. Kræver et oscilloskop for at kunne ses.

Med passende forstærkning kan den bevæge membranen på en højttaler enhed ind og ud med den givne frekvens og dermed generere lyd.

Igen så bruges svingning i elektriske kredsløb til andet end at lave lyd.

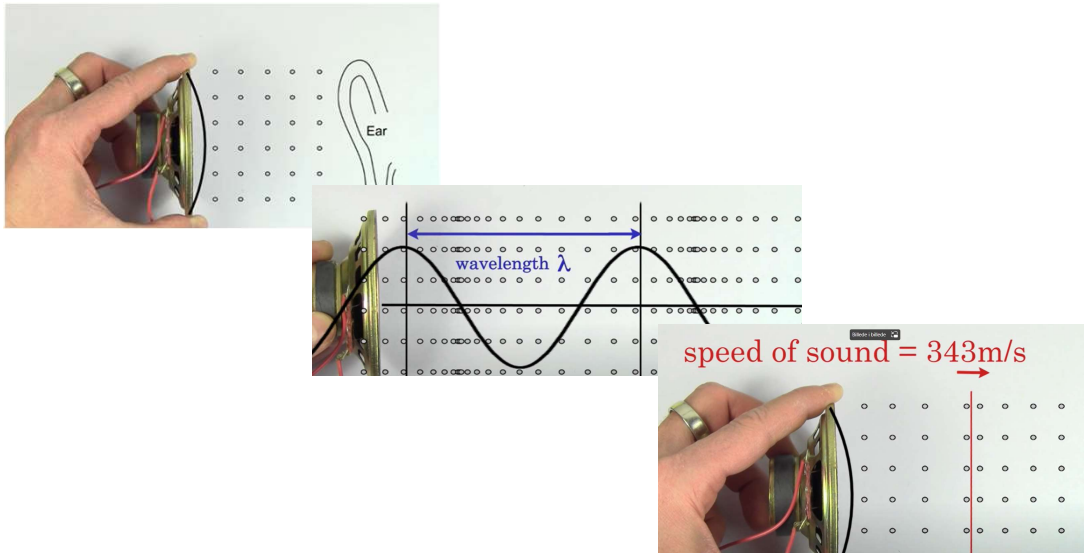
Matematisk funktion der beskriver svingningen.

Grafisk repræsentation

Både sinus og cosinus funktionen giver en sinus kurve, bare faseforskudt. Sinus går gennem 0 ved start (tiden nul).

Spor 3: Lyd-bølge (svingende luftmolekyler)

Projekt støttet af:
midt
regionmidtjylland



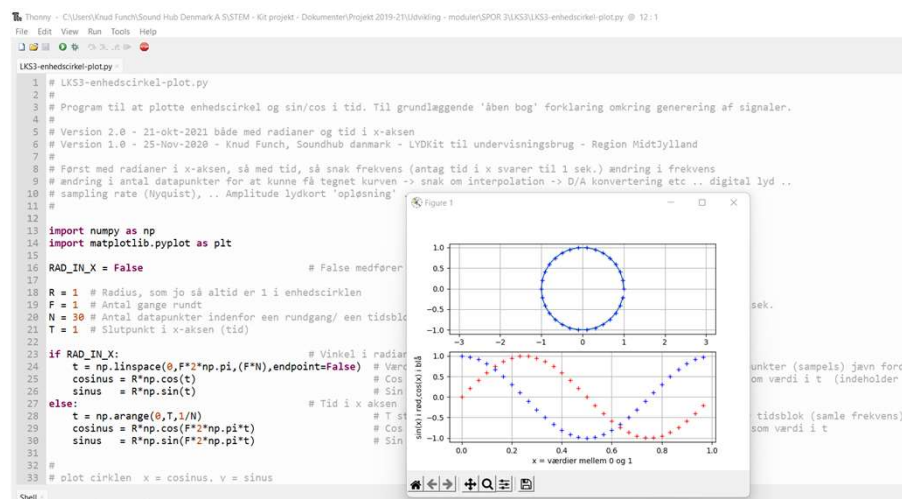
En god video forklaring/illustration af hvordan en svingende membran får luft molekylerne til at svinge.

Klikbare links på billederne til videoerne. Bemærk 3 forskellige!

Kan evt. gives som hjemmeopgave, som supplement til lærebog.

Hørbar lyd siger man ligger i området 20-20.000Hz. Det svarer til bølgelængder på mellem 17.2meter og 17.2millimeter!

Spor 3: Åben bog – Sinus repræsenteret digitalt.



Den matematiske beskrivelse af en harmonisk svingning kan repræsenteres digitalt i et program.

”LKS3-enhedscirkel-plot.py” er et eksempel på det.

Kan bruges til en åben bog introduktion til hvordan man kommer fra matematikken til en digital repræsentation, som så igen i anden række kan bruges til at generere Lyd. I LYD-Kit via funktionerne i funktionsbiblioteket LKlib.py.

Man behøver ikke at tage denne del med – man kan bare benytte funktionerne `generate_sine()`, `play_signal()` og evt. `plot_signal()` for at generere og høre lyden!

Plot i 2 udgaver styret af om konstanten `RAD_IN_X` er True eller False.

Med True er plottet med radianer i x-aksen, og med False er det tid (1 sekund) og programteknisk er der vist 2 forskellige måder at lave et array på til at holde datapunkterne.

Hvis vi tager 30 datapunkter på en enkelt omgang i enhedscirklen kan vi plote funktionerne

Lader vi én omgang svarer til 1 sekund har vi -> 1 Hz ikke hørbart.

Vi skal rundt mere end 20 gange pr. sekund (også mere med den forstærker og højttaler vi benytter – typisk omkring 100)

For at kunne se en sinus (at interpolere mellem punkterne og få en pæn/glat kurve) skal der så også langt flere datapunkter til.

Med perfekt hørelse kan man høre lyd der har frekvenser på op til 20.000Hz (grænsen er dog typisk tættere på de 16000Hz og desuden er der også grænser for hvad højttaler kan reproducere). Men det kræver så igen at man skal 20000 gange rundt i enhedscirklen på 1 sek. Og for at kunne 'se' en pæn kurve have mange flere datapunkter.

I 40'erne faldt nogle forskere (Shannon-Nyquist) ud af at man kan reproducere et kontinuerligt signal ved 'bare' at have det dobbelte antal datapunkter ift. den maksimale frekvens signalet indeholder. Det førte til at man i 70'erne kunne lave digital lyd (CD'en som eksempel) og i den forbindelse typisk taler om sampling frekvens på 44.100 ifm digital lyd. Et godt eksempel på hvordan Science, Matematik og Engineering spiller sammen.

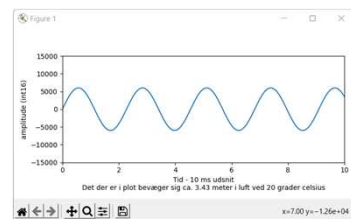
En sidste ting ifm at lave sinus kurven om til lyd er at det skal igennem et lydkort, en forstærkning og en højttaler. Det digitale signal (talværdier) skal laves om til en kontinuerlig spændings kurve der kan drive en højttaler. Første trin i det er at sende værdierne til et lydkort med en digital til analog konverter (D/A konverter). Det lydkort der typisk findes i computere har en opløsning på 16-bit. Dvs. kan repræsentere 65.536 forskellige lyd niveauer $(-32.768 - 32.767)$, som så er den repræsentation der er af Amplituden i signalet. (Dertil kommer også at selve forstærkeren kan have en volumen regulering).

Spør 3: Generer lyde med LKlib.

```
from LKlib import *  
  
init_mixer()  
  
signal = generate_sine(440)  
play_signal(signal)
```

```
from LKlib import *  
  
init_mixer()  
  
signal = generate_sine(440)  
play_signal(signal, forever=True)
```

```
from LKlib import *  
from LKSEELib import *  
  
init_mixer()  
  
signal = generate_sine(440)  
plot_signal(signal)  
play_signal(signal)
```



Basale eksempler på at generere lyd (og plot) med Lklib.py lukket bog.
HUSK Lklib.py i samme direktorie som man gemmer sine programeksempler i.

Desuden kan man bare tage udgangspunkt i de eksempler som ligger og bare ændre argumenter.

Fra venstre mod højre:

- Laver et sinus signal med frekvensen 440, og afspiller det (varighed 1 sekund, som er default værdien)
- Ditto, men bliver ved med at afspille lyden kontinuerligt indtil program stoppes.
- Som det første men med et plot af signalet tilføjet. Udsnit i plot er på 10 msek.

Rene signaler uden nogen form for varians – ligger op ad den rene matematiske model.

Spor 3: Alternativ måde at generere lyd med bestemt frekvens.

Projekt støttet af:
midt
regionmidtjylland



Til at give lidt perspektivering.

Fra 6 minutter inde til 7:30 i video (TV serie) fra 1969 med Prof. Julius Sumner Miller.

Generere lyd mekanisk

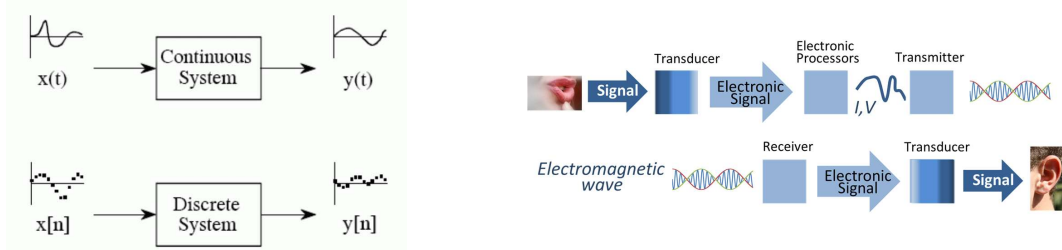
Luft og skive med huller der roterer -> frekvenser

Her indeholder lyden mere end den rene frekvens! Det gør den mere interessant.

Musik instrumenter er igen endnu mere interessante i deres lyd!

Klikbar link på billedet i slide starter 6 min inde

Spor 3: Interferens – signaler og systemer.



Kan bruges til at lave en overordnet forklaring på og skelnen mellem signal og system.

En perspektivering om hvad viden om interferens kan bruges til.

- At system påvirker signal og hvordan det opleves/måles hos modtageren.
- At flere signaler påvirker hinanden.
- At modeller i fysik (faget) typisk udtrykt med matematiske formler er under idealiserede forhold.
- Ex. med lyd så tænker man en punkt formet lyd giver og modtager i et rum uden refleksioner.

Virkeligheden er så meget mere kompleks.

- En lyd giver er typisk ikke punkt formet.
- Lyd ved stemmebåndet formes (moduleres) af hals og munden form – Prøv bare med 'aa' og 'iie' lyd
- Det kan påvirkes af bevægelse foran munden – prøv bare indianer kaldet.
- Det kan ændres af refleksioner – Prøv bare med hænderne som tragt, eller gå ind i et hjørne.

I et elektromekanisk lyd setup (et musikanlæg) kan der være flere højtalere (stereo eller

sourround) og der kan også være flere enheder i hver højtaler. Højtalerne er i et kabinet som kan give refleksioner. Rummet (væg, loft og gulv) giver også refleksioner. Alt sammen påvirker signalet undervejs til modtageren. (Lyd signaler spreder sig i alle retninger og bliver som sådan til flere signaler inden det rammer øret)

Som modtager er man heller ikke punktformet. Man har 2 ører, som også påvirker oplevelsen. Hjælper med til at kunne lokalisere ting.

Viden om det kan omvendt også bruges til at lave ændringer. Hvis man for eksempel har et kendt signal ind og så måler på hvordan signalet ser ud på et bestemt sted kan man se hvor meget signalet er ændret. Hvis ændringerne ikke er hensigtsmæssige kan man arbejde på at lave om i 'systemet'. Det kan være at ændre på rummets karakteristik (akustisk). Det kan være på et højtaler kabinets udformning. Det kan være på at lave selve signalet om så man kompenserer på systemets påvirkning.

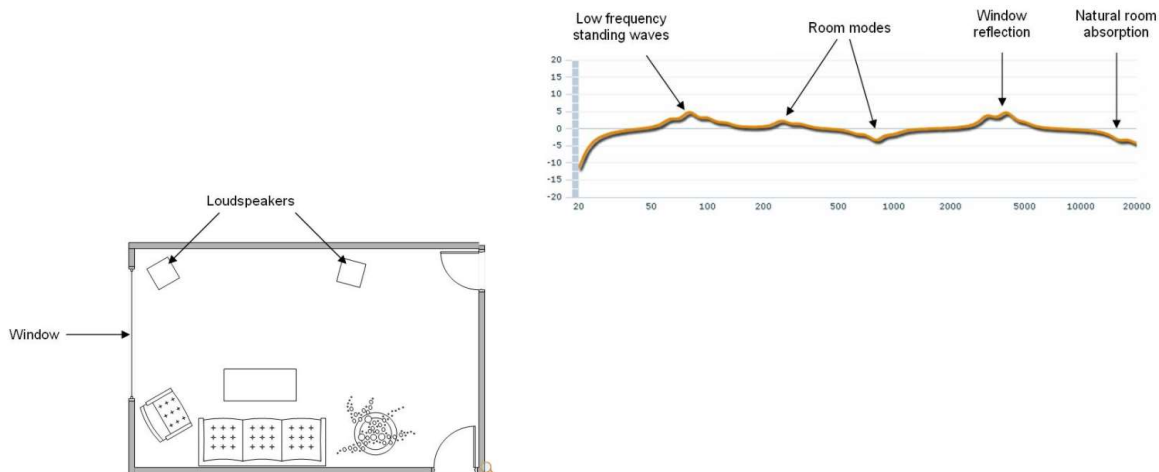
I den virkelige verden er signaler typisk kontinuerlige.

I den digitale er de diskrete. (samplet i tidsintervaller).

I det digitale domæne kan man behandle signaler digitalt.

Spor 3: Interferens – signaler og systemer. Eksempel

Projekt støttet af:
midt
regionmidtjylland



Et eksempel.

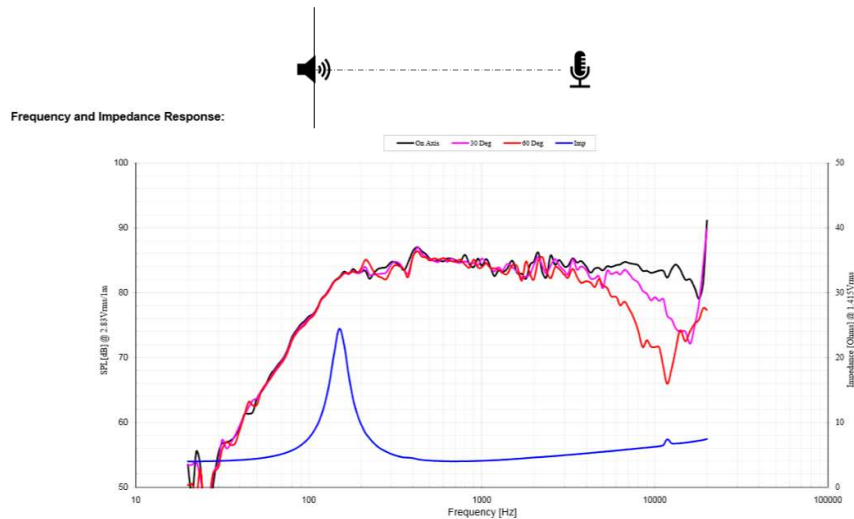
Ved at sende et signal ud med fast intensitet og med et sweep hen over de hørbare frekvenser og med en måle mikrofon placeret i eksempelvis sofaen kan man få en måling på hvordan rummet påvirker signalet.

Man kan så kompensere – enten ved at lave om i rummet (hvad måske ikke altid er en mulighed)

Men også ved at lave signalbehandling på oprindelig lyd inden den kommer ud af højttaleren ved at dæmpe signal der hvor rummet forstærker det og omvendt styrke det hvor rummet dæmper det.

Klikbar link på billede til venstre med kort video om reflekteret lyd i rum.

Spor 3: Interferens – signaler og systemer. Eksempel



Et andet eksempel

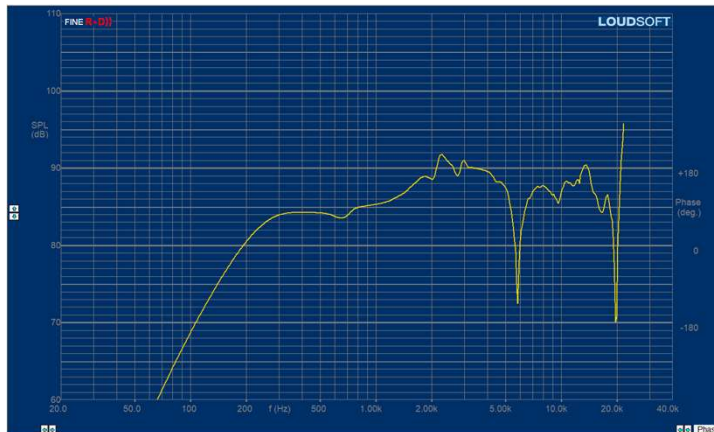
Her er en måling af den højtalerenhed som er brugt sammen med LYD-Kit.

Ideelt set skal frekvensresponsen være 'flad'. Den sorte graf er en måling (on axis) lige udfor enheden (placeret i en plade med 'uendelig' udstrækning i et stort rum – så der ikke er refleksioner)

Den viser at enheden har en fin respons fra ca. 100Hz til noget over 10.000Hz (rimelig flad).

På næste slide ser man en måling (on Axis) hvor enheden er i kabinettet, der er brugt til LYD-Kit til sammenligning.

Spør 3: Interferens – signaler og systemer. Eksempel



Et eksempel

Ved 5000 Hz er bølgelængden = 343m/s divideret med 5000Hz = 0.068m = 6.8 cm. En halv bølgelængde (pi eller 180 grader faseforskydning) er altså 3.4cm som forklarer at der er et dyk ved de ca. 5000Hz.

Det er fordi lyd fra højttaler driver og fra højttaler kassens kant rammer mikrofonen samtidigt med en halv bølgelængdes forskydning som medfører destruktiv interferens.

Igen er virkeligheden mere kompleks end teorien.

Lyden kommer ikke fra et punkt men fra flere steder på højttaler membranen, og da kanten ikke er rund er der forskellig afstand til den etc.

Skal man forbedre højttaleren kan man bringe højttaler enheden ud til kanten, men så bliver den til gengæld mere sårbar rent fysisk, da kanten beskytter enheden. Sådan er det ved produktudvikling og Engeneering. Der er flere ting at tage højde for.

Spor 3: Det simple eksempel med stødtone i signal.

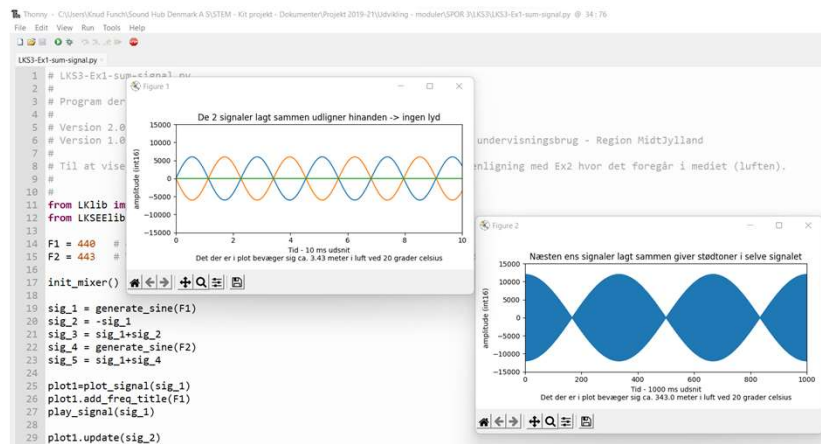
```
from LKlib import *  
  
init_mixer()  
  
sig_1 = generate_sine(440)  
sig_2 = generate_sine(443)  
signal = sig_1+sig_2  
play_signal(signal, forever=True)
```

"LKS3-Ex1-sum-signal.py" kan bruges til at demonstrere interferens, men der er flere ting i spil i programmet, som vist i næste slide.

Derfor blot det helt basale eksempel på kode til at generere 2 signaler som lægges sammen til et inden det afspilles.

Laver et signal der er kombineret af 2 på hhv. 440 og 443 ved simpel addition af de 2, og afspiller det. (stødtone i signalet)

Spør 3: Interferens. I signalet.



Brug "LKS3-Ex1-sum-signal.py"

Til at illustrere interferens i selve signalet. Den negative udgave af signal 1 svarer til et signal 180 grader (pi radianer) faseforskudt og når de 2 lægges sammen bliver det til ingenting. Det kan både ses og høres.

Der er også eksempel med stødtone i selve signal når 2 signaler der er næsten ens lægges sammen.

440 og 443 -> stødtone på 3 Hz som kan høres! 3 toppe i plot med 1 sek.
Det kan høres selv i en enkelt højttaler, da det er i selve signalet det sker.

Det svarer til den ideelle situation. Den enkle model. Den enkle ligning. Den rene matematik.


Til sammenligning med LKS3-Ex2-interferens-luft.py – se næste slide.

Spor 3: Interferens. I mediet (luften).



```
from LKLib import *
init_mixer()

sig_1 = generate_sine(440)
sig_2 = -sig_1
sig_3 = sig_1+sig_2
sig_4 = generate_sine(443)
```

	<code>play_signal(sig_1,1,0,forever=True)</code> <code>play_signal(sig_1,0,0,forever=True)</code>	Signal i én højttaler med én intensitet
	<code>play_signal(sig_1,1,0,forever=True)</code> <code>play_signal(sig_1,0,1,forever=True)</code>	Samme signal i 2 højttalere, hver med samme intensitet som før. Opleves højere.
	<code>play_signal(sig_1,1,0,forever=True)</code> <code>play_signal(sig_2,0,1,forever=True)</code>	Modsat signal i 2 højttalere, hver med samme intensitet som før. Opleves lavere. Hvorfor forsvinder lyden ikke helt?
	<code>play_signal(sig_1,1,0,forever=True)</code> <code>play_signal(sig_4,0,1,forever=True)</code>	2 rene signaler i hver højttaler med 3Hz forskel, og samme intensitet. Der opleves stødtoner, men det lyder lidt anderledes end hvor det var i selv signalet.

Brug "LKS3-Ex2-interferens-luft.py" som udgangspunkt til at demonstrere interferens i luften.

Brug de 2 kald til "play_signal()" med argumenterne der styrer lydintensitet i henholdsvis højre og venstre kanal (0 = ingenting, 1= max styrke) til at demonstrere konstruktiv og destruktiv interferens. Man skal bare ændre i parametrene (signal navn og 0,1 for styrke i kanalerne)

Bruger man det til forsøg skal man være forberedt på at det ikke nødvendigvis rammer 100% i forhold til teorien, da virkeligheden netop er mere kompleks end den enkle model.

Det betyder omvendt ikke at den enkle model ikke er værdifuld! Den forklarer jo det grundlæggende i fænomenet!

Sæt højttalerne tæt sammen for at få maksimal effekt ifm med/mod fase demonstration.

Kan også lave setup med højttalere med afstand og så bevæge sig rundt. Her skal man dog være opmærksom på det med at der er flere ting som spiller ind.
Man kan tydeligt opleve interferens fænomener.

Kan kobles til teori og regneopgave omkring intensitet hhv. afstande som vist i video'er

på næste slide.

Spor 3: Interferens. Teori. Mulige opgaver.

PHYSICS - MECHANICS - SOUND AND SOUND WAVES (1) DOUBLING SOUND INTENSITY

$I = 1 \times 10^{-4} \text{ W/m}^2 = 80 \text{ dB}$
 $I = 1 \times 10^{-4} \text{ W/m}^2 = 80 \text{ dB}$
 $I_T = I + I = 2 \times 10^{-4} \text{ W/m}^2$
 $\text{indB} = 10 \log \frac{I}{I_0} = 10 \log \frac{2 \times 10^{-4} \text{ W/m}^2}{1 \times 10^{-12} \text{ W/m}^2} = 10 \log 80 = 83 \text{ dB}$

$I = 1 \text{ W/m}^2 = 120 \text{ dB}$
 $I = 1 \times 10^{-4} \text{ W/m}^2 = 80 \text{ dB}$
 $I = 1 \times 10^{-4} \text{ W/m}^2 = 80 \text{ dB}$
 $I = 1 \times 10^{-4} \text{ W/m}^2 = 80 \text{ dB}$

PHYSICS - MECHANICS - SOUND AND SOUND WAVES (5) SOUND INTERFERENCE (WAVE INTERFERENCE)

COHERENT $f = 367 \text{ Hz}$ $v_{\text{SOUND}} = 345 \text{ m/s}$
 $\lambda = \frac{v}{f} = \frac{345 \text{ m/s}}{367 \text{ Hz}} = 0.94 \text{ m}$
 $\text{X DIST} = 4.47 \text{ m} - 4 \text{ m} = 0.47 \text{ m}$
 $\text{X DIST} = \lambda \left(\frac{0.47 \text{ m}}{0.94 \text{ m}} \right) = \frac{1}{2} \lambda$
 DESTRUCTIVE INTERFERENCE

Klikbare links på billeder i slide

Kan gives som hjemmeopgaver. Kan bruges som udgangspunkt for Opgaver.

Spor 3: Interferens. En anden måde at vise stødtoner.

Projekt støttet af:
midt
regionmidtjylland



Ud over den klassiske med stemmegaflerne kan man også demonstrere stødtoner som Julius Sumner Miller i TV serie fra 1969.

Det rent mekaniske fungerer fortsat. Det digitale er bare kommet til som en ekstra mulighed.

Fra 1:59 minutter inde til 4:10 i video (TV serie) fra 1969 (prof. Julius Sumner Miller)

Lyd genereres mekanisk (med luft i rør)

Luft og rør med forskellig længde -> stødtoner

Luft og rør i samme længde men forskellig luft temperatur inde i rør -> stødtoner

Klikbar link på billedet i slide starter 1:59 min inde

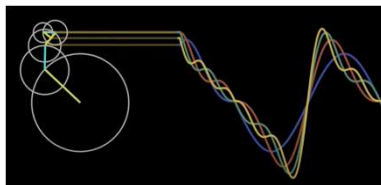
Spor 3: Partialtoner -> Fourier serier.

```
from Lklib import *
init_mixer()
bf = 440
ba = 8000

f1 = generate_sine(bf,ba) # fundamental frekvens

# n'th overtone med n*(fundamental frekvens) og 1/n * (fundamental amplitude)
o2 = generate_sine(2*bf,1/2*ba)
o3 = generate_sine(3*bf,1/3*ba)
o4 = generate_sine(4*bf,1/4*ba)
o5 = generate_sine(5*bf,1/5*ba)
o6 = generate_sine(6*bf,1/6*ba)
o7 = generate_sine(7*bf,1/7*ba)
o8 = generate_sine(8*bf,1/8*ba)
o9 = generate_sine(9*bf,1/9*ba)
o10 = generate_sine(10*bf,1/10*ba)
o11 = generate_sine(11*bf,1/11*ba)
o12 = generate_sine(12*bf,1/12*ba)
o13 = generate_sine(13*bf,1/13*ba)
o15 = generate_sine(15*bf,1/15*ba)

signal = f1+o2+o3+o4+o5+o6+o7+o8+o9+o10+o11+o12+o13 # Saw
play_signal(signal,forever=True)
```

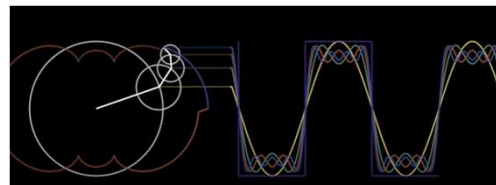


```
from Lklib import *
init_mixer()
bf = 440
ba = 8000

f1 = generate_sine(bf,ba) # fundamental frekvens

# n'th overtone med n*(fundamental frekvens) og 1/n * (fundamental amplitude)
o3 = generate_sine(3*bf,1/3*ba)
o5 = generate_sine(5*bf,1/5*ba)
o7 = generate_sine(7*bf,1/7*ba)
o9 = generate_sine(9*bf,1/9*ba)
o11 = generate_sine(11*bf,1/11*ba)
o13 = generate_sine(13*bf,1/13*ba)
o15 = generate_sine(15*bf,1/15*ba)

signal = f1+o3+o5+o7+o9+o11+o13+o15 # Square
play_signal(signal,forever=True)
```



Med Lklib.py kan man lave flere signaler og addere dem. Så simpel addition med overtoner kan eksempelvis bruges til at generere sav og firkant funktioner. Det illustrerer helt grundlæggende princippet i additiv syntese.

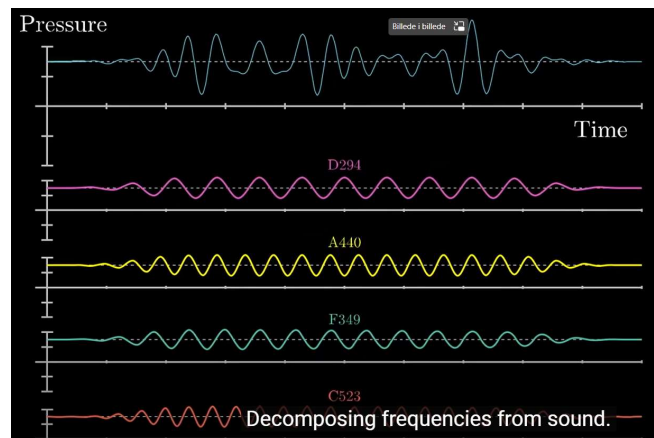
Samtidigt knytter det an til teori i fysik omkring partialtoner og resonans i den fysiske/mekaniske verden.

Live animationer på klikbare links på billeder for hhv. saw og square wave. God visuel illustration af hvad der sker når man adderer sinus signaler (med en vis amplitude og frekvens). Hastigheden i omdrejningen er frekvensen og radius i de enkelte er forholdet i amplitude. Der er en kobling til den rene model omkring partialtoner i åbne hhv. halvåbne rør.

Samtidigt kan det give en intuitiv forståelse for Fourier serier som kan bruges til meget andet end lyd.

Brug "LKS3-Ex3-Overtone-ulige.py" samt "LKS3-Ex4-Overtone-alle.py" til at demonstrere.

Spor 3: Fourier analyse.



En 20 minutter video (via klikbar link på billedet) der visualiserer Fourier transformationen.

Kan gives som hjemmeopgave. Ikke for at elever skal forstå den, men simpelt hen fordi den kan give en intuition om hvad det er der sker ved en Fourier Analyse, som de skal kunne lave med nogle af de værktøjer der præsenteres for på skolen.

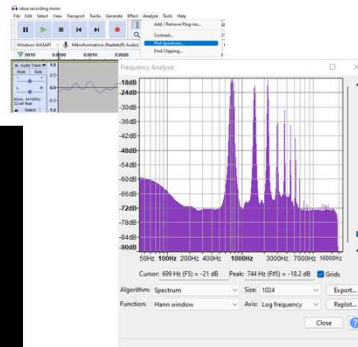
Det 'modsatte' af Fourier serierne.

At kunne dekomponere et sammensat signal og finde dets grundbestanddele. De frekvenser som signalet består af.

En mulig opgave vil efterfølgende være at få eleverne til at lave en Fourier analyse (en måling) på et blæseinstrument. Finde de hoved frekvenser, som signalet er bygget op af. Argumenter for om det er et åbent eller halvåbent rør, samt at få dem til at forsøge at lave en emulering af lyden ved at kombinere sinus signaler med indbyrdes relativ intensitet som i målingen.

Måske skal man stoppe 12 minutter inde, hvor der kommer komplekse tal på bordet.

Spor 3: Fourier analyse. Mulige opgaver.



Ser man kun på peaks fra diagram:

Hz	dB	Amplitude = 10 ^(dB/20)
744	-18.2	0.128027
1480	-20.9	0.090157
2218	-18.4	0.120226
2957	-27.4	0.042658
3705	-28.5	0.037584
4444	-35.3	0.017179

```
from LKlib import *  
init_mixer()  
ba = 10000  
f1 = generate_sine(744, 0.1230*ba)  
o2 = generate_sine(1480, 0.0901*ba)  
o3 = generate_sine(2218, 0.1202*ba)  
o4 = generate_sine(2957, 0.0426*ba)  
o5 = generate_sine(3705, 0.0375*ba)  
o6 = generate_sine(4444, 0.0171*ba)  
s1 = f1+o2+o3+o4+o5+o6  
play_signal(s1, forever=True)
```

Lav måling (Fourier analyse) på et instrument der spiller en ren tone – ex Obo i link i billedet (tidskode 4:56)

Man kan bruge de måleinstrumenter man har i fysik – logger pro eller andet. Det giver noget at det foregår med en mikrofon!

I slide er Audacity brugt på optaget lydfil (ligger sammen med præsentation). "oboe recording mono.mp3"

Find peaks i måling.

Konverter dB til intensitet (de enkeltes bidrag)

Lav en emulering ved at generere signaler og addere dem.

Giver anledning til en snak om hvorfor det IKKE lyder som det rigtige instrument.

Der er flere faktorer:

- Der er flere toner i spil – Man kan se på frekvens analysen at der rundt om peak er flere som bidrager.
- Selvom man måler over et stykke tid er den tone man laver kun statisk – (ens hele

vejen mens den afspilles) – En tone man trækker (i langdrag) som kun svarer til et øjebliksbillede.

- Man kan sige at tonen er for ren (og kedelig) i forhold til virkeligheden. I virkeligheden varierer tonen hele tiden – også når den der spiller prøver at holde tonen ren. Det vil svare til at man i Fourier serien ser radier og hastigheder variere en smule løbende.
- Desuden vil den der spiller på instrumentet typisk addere noget liv – som ex. i klippet hvor der efter den rene tone tilføjes vibrato.

Ser man på moderne frekvens og spektrum analysatorer så kan man få dem til at lave målinger i realtid, og så kan man se at spektret står og ændrer sig hele tiden når man måler.

Der er nogle eksempler i supplerende materiale man kan bruge til at illustrere (slide 26)

Igen er virkeligheden mere kompliceret, men den grundlæggende teori er også rigtig og med til at skabe forståelse. Der er bare rigtig mange bidragsydere til den oplevede lyd, så den er ikke bare sådan ligetil at efterligne. Og sådan er det generelt med modeller – også i den digitale verden. Det er kun en model!

Man vil også kunne lave andre typer af opgaver ved ex. at lade en gruppe elever generere en lyd der modsvarer et halvåbent rør, og så lade en anden gruppe finde ud af hvad det er for et – ved at måle.

Man kan måske lave hemmelige beskeder?

Man kan lave flere ting ved signalet som gør at man får en mere interessant lyd. Det er ting som modulering, tilføjelse af envelopes og filtre etc.

Ting man støder på ifm synthesizers (elektronisk musik), men som altså i den grad også er teknologi/fysik og matematik. I option delen kradser vi ganske lidt i overfladen for at vise at man kan lave andre lyde rent syntetisk.

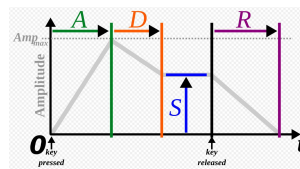
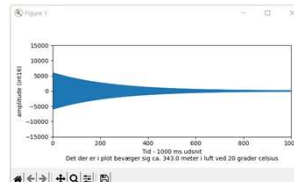
Spor 3: Option – Dæmpning (decay) og envelope

```
from LKlib import *
from LKSEELib import *

init_mixer()

signal = generate_sine(440)

sig_decay = exp_decay(signal)
plot_signal(sig_decay, time_to_plot=1000)
play_signal(sig_decay)
```



Attack, Decay, Sustain, Release

I virkelighedens verden vil et pendul eller en anslået streng over tid stoppe med at svinge hvis ikke der tilføres energi.

Det kan også emuleres ved at et rent signal ganges med en eksponentielt faldende funktion. (værdier mellem 1 og nul)

Man kan sige at det oprindelige signal pakkes ind (envelope) i den eksponentielt faldende funktion.

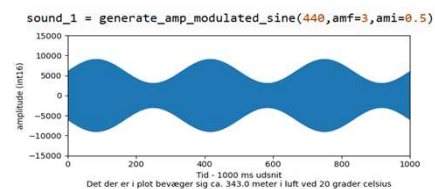
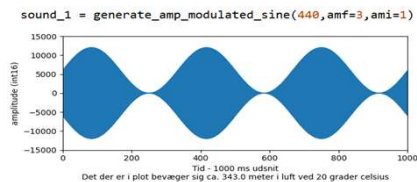
Ovenfor er der et basalt eksempel på det med funktion `exp_decay(signal)` fra `LKlib.py`.

Det ændrer naturligvis radikalt på oplevelsen af det man hører/oplever.
I eksemplet bliver den rene tone til noget mere klokke agtigt.

I forbindelse med elektronisk musik (synthesizers) taler man om ADSR envelopes til at emulere anslag og dæmpning på lydene.

Spor 3: Option – Amplitude modulation

Lukket bog:



Åben bog: "Lklib.py"

```
# -----  
# Funktion der genererer og returnerer et sinus signal som er amplitude modelleret med den frekvens der angives som argument.  
#  
# Parametre:  
# - freq : Den frekvens man ønsker signal lavet i, altså hovedsignalet som lydens pitch er bestemt af  
# - amp_max : Amplituden. Dvs den største værdi signalet skal have.  
#           Kan udelades og så er default 6000  
# - amf : frekvens på amplitude modulations signalet  
#           Kan udelades. Så er default 60, som tydeligt kan ses på plot (og høres). Bør også prøves med andre værdier.  
# - ami : Index som ganges på modulations signalet. Giver et forhold mellem amplituderne på de 2 signaler (carrier og modulator)  
#           Kan udelades. Så er default 1, som betyder at amplituden i samlet signal går til 0.  
#           Bør prøves med andre værdier (mindre end 1 - ex 0.5 som betyder samlet signal går til 0.5 istedet for 0)  
#  
def generate_amp_modulated_sine(freq, amp_max=6000, amf=60, ami=1):  
    t = np.arange(0, DURATION, 1/SAMPLERATE)  
    am_signal = (amp_max*(1+ami*np.sin(amf*2*np.pi*t))*np.sin(freq*2*np.pi*t)).astype(np.int16)  
    return am_signal
```

$m(t) = A_m \cos \omega_m t$
Modulator Signal

$c(t) = A_c \cos \omega_c t$
Carrier Signal

$AM(t) = [A_c + m(t)] \cos \omega_c t$
Amplitude Modulated Signal

Tag udgangspunkt i "LKS3-Ex10-Amplitude-Modulation.py"

Amplitude modulation er at pakke et signal ind i en 'envelope' som varierer amplituden efter en sinus kurve. Med et modulations index der bestemmer om den går helt til nul eller mindre.

Argumenterne "amf = 3" og "ami = 1" giver et signal, der ligner stødtone med 440 og 443 lagt sammen – men ikke helt. Lyder næsten ens.

ami = 0 svarer til ren sinus

Brug værdier mellem 0 og 1 som betyder at signal ikke går i nul – men OK at prøve værdier der er større end 1. Det giver 'sjove' resultater.

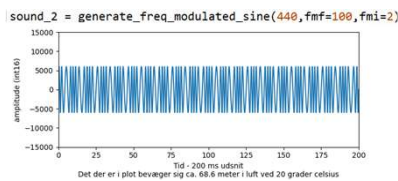
Kun en simpel implementering af amplitude modulation. Ikke tænkt som tilbundsående, da det kræver mere.

Kun for at indikere overfor eleverne at der ligger noget her. At man kan manipulere signalerne digitalt og få 'sjove' effekter ud af det.

I formlen til venstre bruges cos() men vi bruger sinus i stedet for at få et signal der starter i nul. Husk det er en simplificering.

Spor 3: Option – frekvens modulation

Lukket bog:



Åben bog: "Lklib.py"

$$FM(t) \approx A \sin(\omega_c t + \beta \sin(\omega_m t))$$

```
# -----
# Funktion der genererer og returnerer et sinus signal som er frekvens modelleret med den frekvens der angives som argument.
#
# Parametre:
# - freq : Den frekvens man ønsker signal lavet i, altså hovedsignalet som lydens pitch er bestemt af
# - amp_max : Amplituden. Dvs den største værdi signalet skal have.
#           Kan udelades og så er default 6000
# - fmf : frekvens på modulations signalet
#           Kan udelades. Så er default 100, som tydeligt kan ses på plot (og høres). Bør også prøves med andre værdier.
#           En frekvens der er et rent forhold til frekvens på carrier giver noget der lyder harmonisk.
# - fmi : index som ganges på modulations signalet. Giver et forhold mellem amplituderne på de 2 signaler (carrier og modulator)
#           Kan udelades. Så er default 1. Bør prøves med andre værdier (større end 1)
#
def generate_freq_modulated_sine(freq, amp_max=6000, fmf=100, fmi=1):
    t = np.arange(0, DURATION, 1/SAMPLERATE)
    fm_signal = (amp_max*(np.sin(freq*2*np.pi*t+fmf*np.sin(fmf*2*np.pi*t))))).astype(np.int16)
    return fm_signal
```

Tag udgangspunkt i "LKS3-Ex9-freq-Modulation.py"

Frekvens modulation kan siges at være en måde at 'trække i' det oprindelige signal – så frekvensen varierer lidt (med en anden frekvens)

Eksperimenter med argumenterne og skab 'sjove' lyde.
Eksempelvis:

- fmf = 1 fmi = 1 - Det samme som ren sinus
- fmf = 1 fmi = 200 - står og svinger
- fmf = 590 fmi = 1
- fmf = 8 fmi = 200

Der er mange kombinationer som giver ændring i lyd, men som er svære at se på plot – så man kan også bare fjerne plot og 'stole' på ørene når man eksperimenterer.

Kun en simpel implementering af frekvens modulation. Ikke tænkt som tilbundsgående, da det kræver mere. Kun for at indikere overfor eleverne at der ligger noget her. At man kan manipulere signalerne digitalt og få 'sjove' effekter ud af det.

Der er også en funktion, hvor man både har frekvens og amplitude modulation

kombineret.

Kør og hør eksemplet i "LKS3-Ex11-Freq-og-Amp-Modulation.py", hvor også er en udgave med `exp_decay()` - så har man en lyd der er lidt mere 'interessant' og sin egen – genereret rent digitalt ved noget digital matematik på en række værdier i en tabel.

Spor 3: Option – Simpelt klaver med digitalt genererede lyde

```
# Funktion der laver tonerne.  
# Modificer funktionen for at lave andre toner ved at bruge og kombinere signaler lavet med funktionerne i Lklib.  
# Ex ved at addere overtoner. Prøv dem som er udkommenteret.  
def make_tones(freq):  
#   x = generate_sine(freq)  
#   x = env(generate_sine(freq))  
#   x = exp_decay(generate_sine(freq))  
#   x = generate_triangle(freq)  
#   x = env(generate_triangle(freq))  
#   x = exp_decay(generate_freq_modulated_sine(freq,fmf=int(freq/2),fmi=5),-3)  
#   x = exp_decay(generate_amp_modulated_fmsine(freq,fmf=int(freq/2),fmi=5,amf=8,ami=0.4),-3)  
  x = env(generate_amp_modulated_fmsine(freq,fmf=int(freq/2),fmi=10,amf=8,ami=0.2))  
  return x
```

Ved at lade eleverne tage udgangspunkt i "LKS3-simple-7key-piano.py" kan man give dem en mulighed for at lave 'klaver' med egne lyde.

Programmet virker med at generer lyde med funktionerne fra Lklib.py, og mapper dem til tasterne "a,s,d,f,g,h,j" på deres laptop's

De kan/skal som det eneste ændre i koden i funktionen "make_tones()" for at få en ny lyd.

Der er allerede en række forskellige bud som bare er kommenteret ud, og som man kan prøve at skifte imellem (én udgave pr. linie)

En lidt mere avanceret opgave (programteknisk) kunne gå ud på at bruge en sammensat lyd – svarende til den man har lavet ud fra en måling – eller bare en man selv sammensætter.

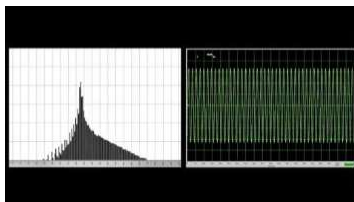
Man kan også lave en med overtoner og forhold som man ex. har fundet i en opgave med Fourier analyse.

Og den avancerede vil nok hurtigt finde ud af at kunne udvide til/med flere taster/frekvenser ved at ændre i tabellerne i programmet – men så er man i gang med det der ligger i spor 4.

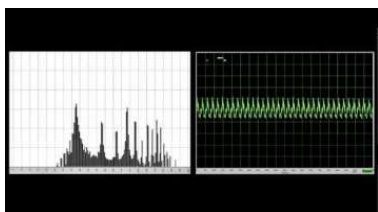
Spor 3: Supplerende materiale. Diverse – realtidsmålinger.

Projekt støttet af:
midt
regionmidtjylland

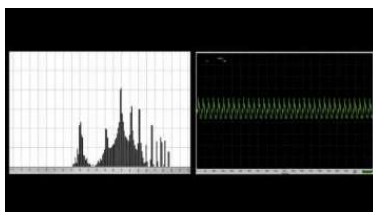
Ren sinus



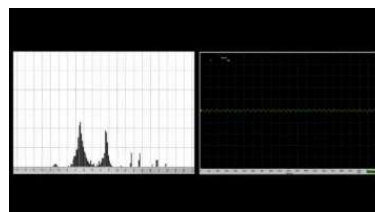
Sax



Obo



Klaver



4 videoer på klikbare links – som kan bruges til at vise at spektret varierer i tid på live signal fra instrumenter.

(I de viste eksempler er der fortsat lidt forsinkelse)

Selv sinus er ikke bare en enkelt søjle men der er en række komponenter der kommer fra system mellem signal og mikrofon.

Prøv den selv med en ren sinus og mål med eget måleudstyr – også selvom det kun viser et øjebliksbillede.

Spor 3: Supplerende materiale. 3D lyd i headphones

Projekt støttet af:
midt
regionmidtjylland

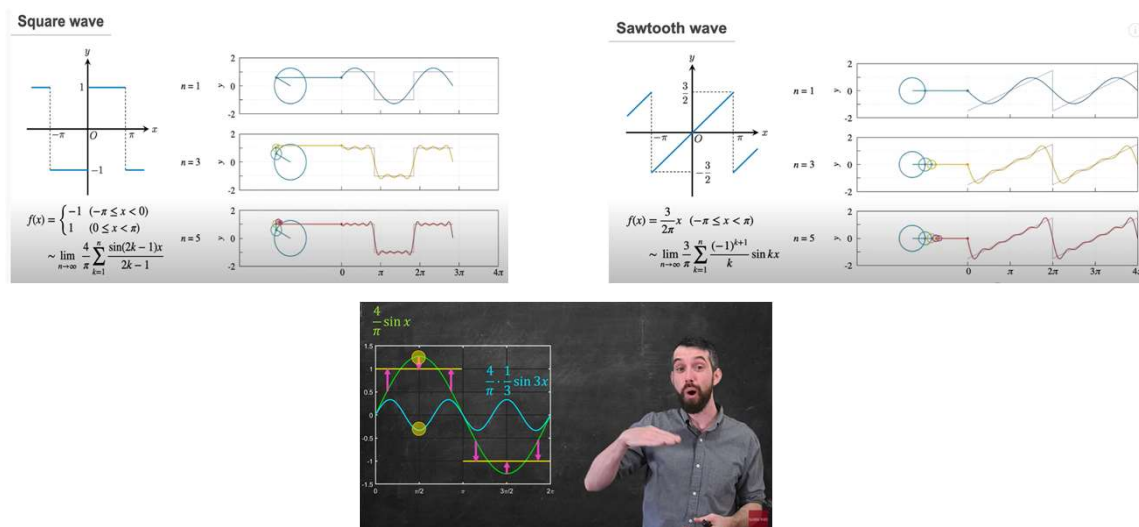


Klikbare links på billederne

Bruges med hovedtelefoner. Bemærk at den ene er en video af en app. Så der er det endnu bedre at downloade og bruge app'en interaktivt.

For at give et billede på hvad man kan opnå med signal behandling for at skabe rum fornemmelse.

Spør 3: Supplerende materiale. Alternative Fourier serie animation



Her med de rigtige formler og en der forklarer det endnu mere.
Klikbare links på billederne (De 2 øverste er i samme video)

Spor 3: Supplerende materiale. Diverse - Kan lyd smadre et glas?

Projekt støttet af:
midt
regionmidtjylland



Klikbar link på billedet

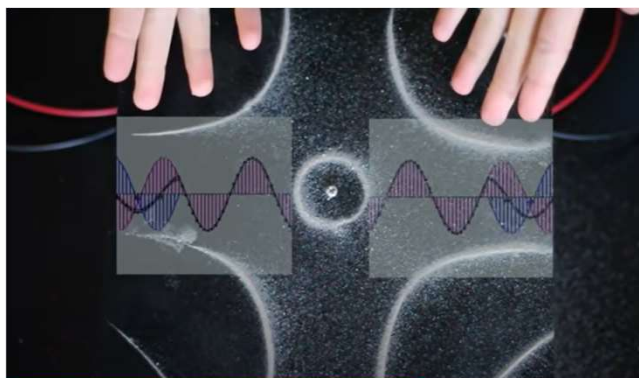
Prof Walter Lewin med forelæsning for ikke studerende fra i 1996 om musik og bølger.

Bl.a. med én demonstration af at lyd kan smadre et glas.

1:36 inde – Link fører til tidskode lidt inden.

Spor 3: Supplerende materiale. Diverse – Visualisering af lyd.

Projekt støttet af:
midt
regionmidtjylland



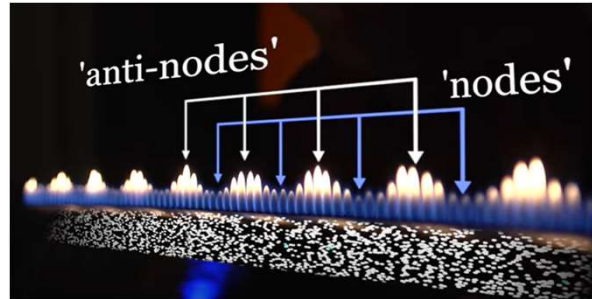
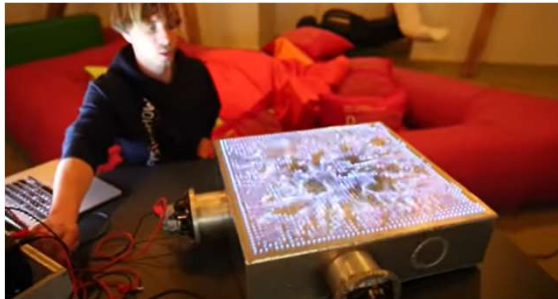
Klikbar link på billedet

Kort video på 4:19 omkring bølger og visualisering på Chladni plader.

BEMÆRK at der 3:10 inde er ting som holdes flyvende i luft med lyd bølger!

Spor 3: Supplerende materiale. Diverse – Visualisering af lyd.

Projekt støttet af:
midt
regionmidtjylland

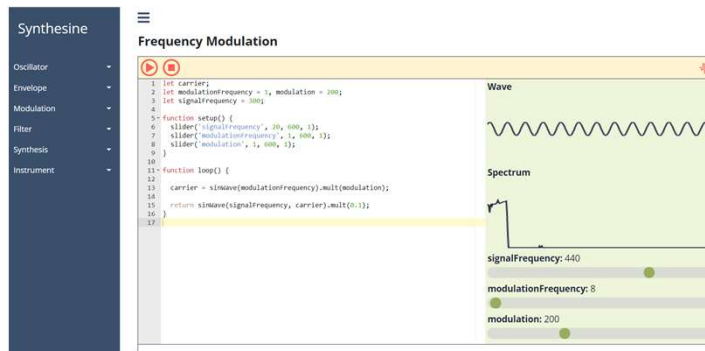


Klikbar link på billeder (samme på begge)

Kort video på 6:33 med udgangspunkt i Rubens rør og resonans, men her taget et trin videre i 2 dimensioner.

Fra DK – fysikshow!

Spor 3: Supplerende materiale.



[Frequency Modulation \(aatishb.com\)](http://aatishb.com)

[Sound Design Basics: FM Synthesis – Cymatics.fm](http://Cymatics.fm)

[Sine, Square, Triangle, Saw : Synth Waveforms - Perfect Circuit](#)

Et alternativ til at lege med Frekvens modulation (og andre effekter), hvor man fortsat kan se at det er kode som generere signalet.

Og så kan man dynamisk ændre i værdierne og høre resultatet med det samme.

Koblingen til matematikken er ikke helt så tydelig. Og man har ikke muligheden for at bygge med den (koble det til sensor input)

Klikbar link på slide fører til Frekvens Modulation. Tryk på den røde play knap og brug slider til at ændre i værdierne.

De 3 sliders svarer til:

freq

fmf

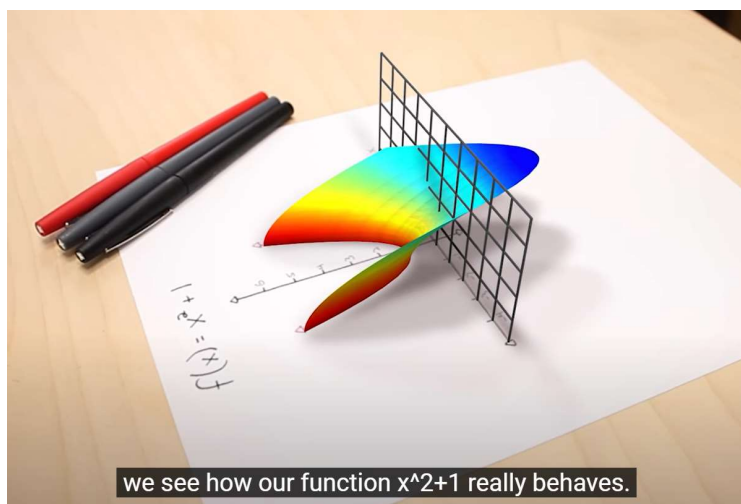
Fmi

argumenterne i "generate_freq_modulated_sine()" funktionen i Lklib.py

Med nogle værdier lyder det anderledes – Det skyldes forskellig håndtering af algoritme og digitale artefakter.... Der er så også en del af den digitale virkelighed.

Spor 3: Supplerende materiale. Om komplekse tal

Projekt støttet af:
midt
regionmidtjylland

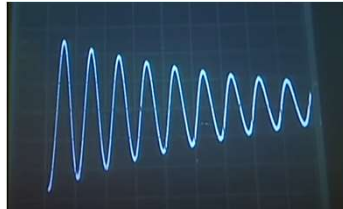
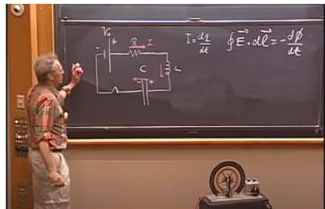
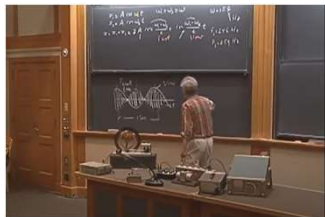


Hvis man vil koble til komplekse tal (Da man støder på det i video omkring Fourier transformationen) kan der henvises til en video serie der forklarer historien om de komplekse tal

Klikbar link på billedet.

Spor 3: En forelæsning om svingninger og dæmpning

Projekt støttet af:
midt
regionmidtjylland



Til perspektivering.

Det er en lang video, så nok ikke brugbar overfor eleverne (bortset fra i små bidder).

Klikbar link på billederne til det sted der henvises til fra en længere forelæsning af Prof Walter Lewin optaget i 2004 forklarer matematikken bag stødtoner (beats), samt oscillators og dæmpning. Det fine er at han både bruger eksempler fra den fysiske verden (mekanisk) såvel som til elektronik, og kobler det til teorien/matematikken bag.

Fra start til ca. 10 min inde drejer det sig om stødtoner (beats) og han bruger stemmegaflerne til at demonstrere – men har så mikrofon og oscilloskop til at måle -> eftervise teorien.

Senere i video viser han dæmpning først med pendul og så i et elektrisk kredsløb – og igen senere en elektromekanisk dæmpning.

Så nu skal vi så bare også have det digitale med 😊 Både mekanik og analog elektronik er der jo fortsat – så det er ikke enten eller – det er både og!

Spor 3: Links

Video om lyd – KhanKademy (slide 6)

[Waves and sound | AP®/College Physics 1 | Science | Khan Academy](#)

<https://youtu.be/nGKffdal4Pg> - Production of sound

<https://youtu.be/-xZZt99MzY> - Sound Properties

<https://youtu.be/UgE2GIQwUCw> - Speed of sound

Mekanisk genereret lyd – demo 1969 (slide 9)

[Lesson 32 - Sound Waves - Sources of Sound - Demonstrations in Physics – YouTube](#)

Rum (slide 11)

[How Sound Works \(In Rooms\) - YouTube](#)

Interferens – Teori (slide 17)

[Physics - Mechanics: Sound and Sound Waves \(11 of 47\) Doubling Sound Intensity - YouTube](#)

[Physics - Mechanics: Sound and Sound Waves \(15 of 47\) Sound Interference – YouTube](#)

Stødtoner som demonstreret i 1969 (slide 18)

[Lesson 35 - Sounding Pipes - Demonstrations in Physics – YouTube](#)

Fourier serier (slide 19)

[Fourier Series Animation \(Square Wave\) – YouTube](#)

[Fourier Series Animation \(Saw Wave\) – YouTube](#)

Spor 3: Links

Fourier transformation (slide 20)

[But what is the Fourier Transform? A visual introduction. – YouTube](#)

Obo lyd og Audacity (slide 21)

<https://youtu.be/QNBsgfh4UMY?t=293>

[Audacity® | Free, open source, cross-platform audio software for multi-track recording and editing. \(audacityteam.org\)](#)

Div målinger (slide 26)

[8 Equ Scope Klaver – YouTube](#)

[8 Equ Scope Obo – YouTube](#)

[8 Equ Scope alt sax – YouTube](#)

[8 Equ Scope Sinus - YouTube](#)

3D lyd i hovedtelefoner – demo (slide 27)

[dearVR API | real-time 3D audio app – YouTube](#)

[dearVR PRO | Immersive 3D audio VST, AAX, AU plugin – YouTube](#)

Alternative forklaringer fourier serier (slide 28)

[Visualizing the Fourier Series of Square / Triangle / Sawtooth Wave Using Circles \[gnuplot\] – YouTube](#)

[Intro to FOURIER SERIES: The Big Idea - YouTube](#)

Spor 3: Links

Div – Smadre glas og Visualiseringer (slide 29,30,31)

<https://youtu.be/GFR8UJK3Mzc?t=5703>

[Singing plates - Standing Waves on Chladni plates - YouTube](#)

[Musical Fire Table! - YouTube](#)

Side på net med syntetisk generation af lyde (siden med Frekvens modulation – slide 32)

[Frequency Modulation \(aatishb.com\)](#)

Imaginary number/komplekse tal (slide 33)

[Imaginary Numbers Are Real \[Part 1: Introduction\] – YouTube](#)

Walter Lewin forelæsning (slide 34)

[8.03 - Lect 2 - Beats, Damped Free Oscillations, Quality Q – YouTube](#)