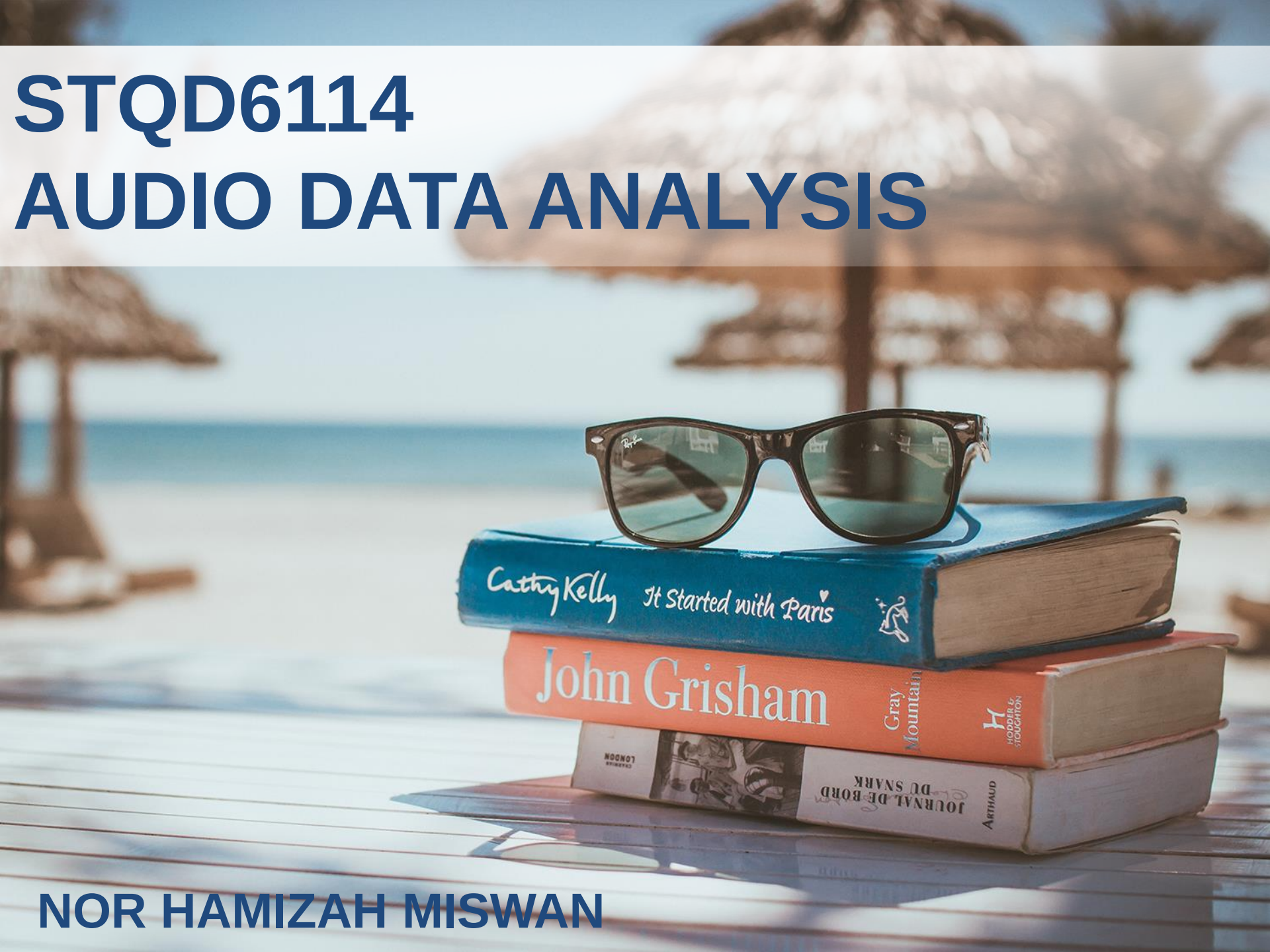
STQD6114
AUDIO DATA ANALYSIS



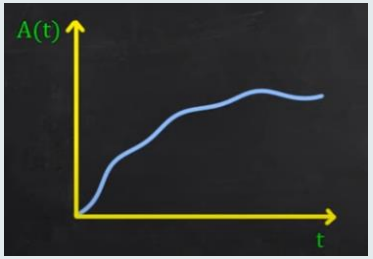
NOR HAMIZAH MISWAN

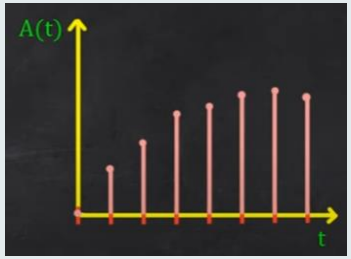
Audio Signal

- Audio signals are the representation of sound, which is in the form of digital and analog signals.
- Sound frequencies range between 20 to 20,000 Hz, and this is the lower and upper limit of our ears.
- The human auditory system is most sensitive between 500Hz and 5000 Hz.
- However, sound under the frequency of 20Hz can also affect the ear even though we are unable to hear them.
- Especially dangerous is infrasound at the frequency of 7Hz, since this frequency is close to characteristics frequencies of the organs of our body, may disturb the heart or brain activity.
- Audio signal processing is a method where intensive algorithms, techniques are applied to audio signals.

Digital Signal Processing

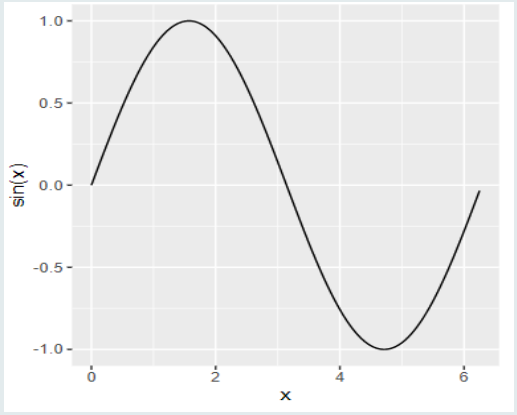
- Digital signal processing → concerned with the representation of signals by sequence of numbers or symb ols and the processing of these sequences.
- Signal → a function that conveys information, generally about state or behavior of a physical system.
- Mathematically, signals are represented as functions of one or more independent variables.

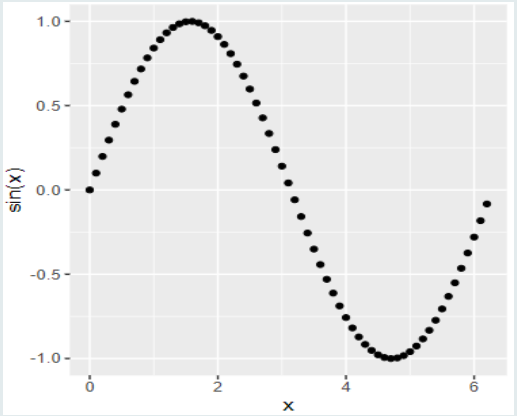


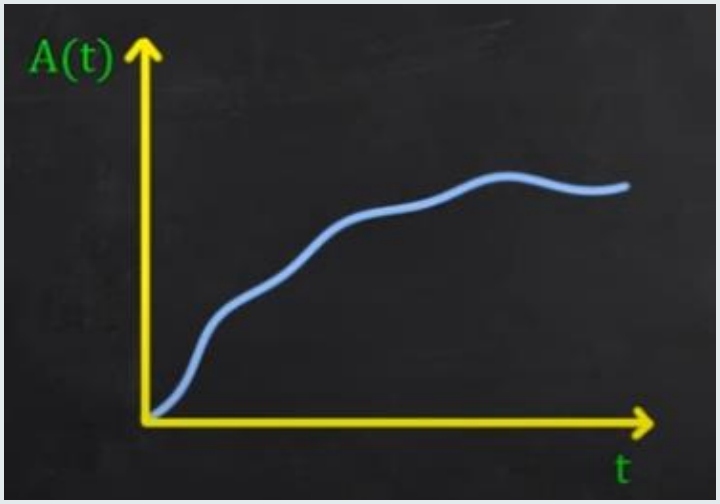


Digital Signal Processing

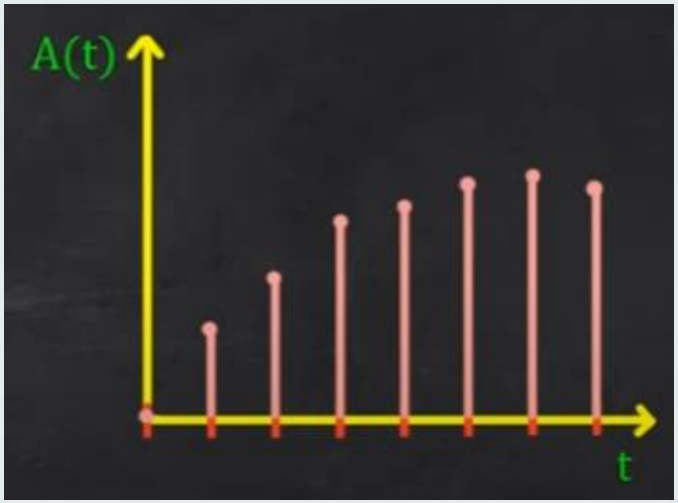
- Two main types of signals; continuous time signals and discrete time signals







Analog signals are those signals for which both time and amplitude are continuous



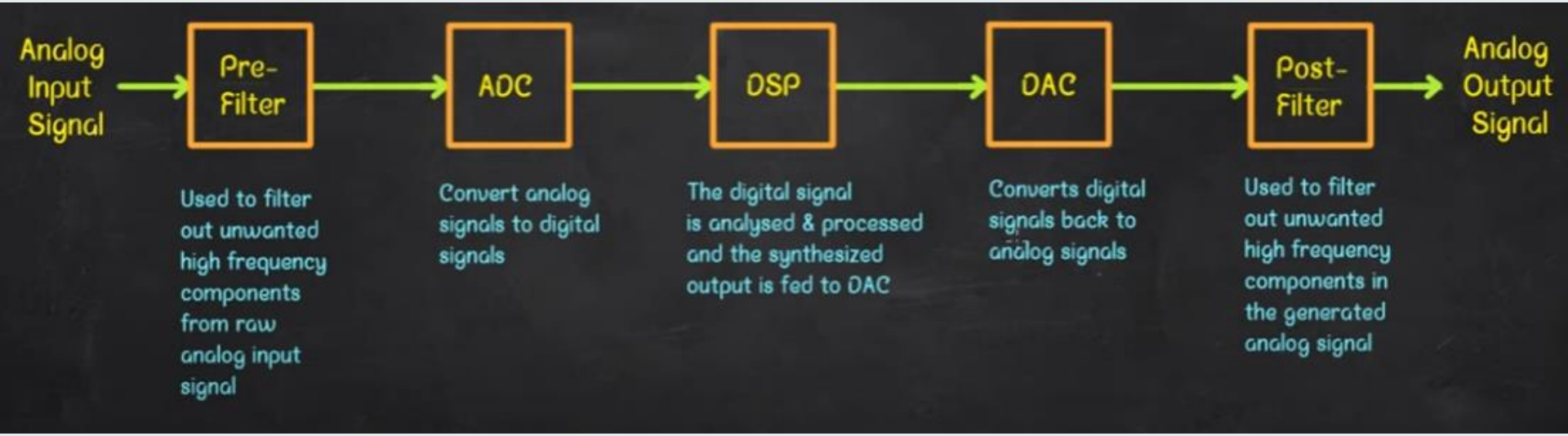
Discrete signals are those signals that are defined only at discrete units of time

Digital Signal Processing

- Signal processing involves analyzing, modifying & synthesizing signals to pull meaning out of it.
- Analog signal processing
  - Analog signal processing deals with transformation of analog signals.
  - Processing is done using electrical networks consisting of active and passive elements.
- Digital signal processing
  - Digital signal processing deals with the processing of discrete signals
  - Processing is done using general purpose computers, ASICs, FPGAs, DSP chips, etc.

Digital Signal Processing

- Block diagram of a digital signal processing system



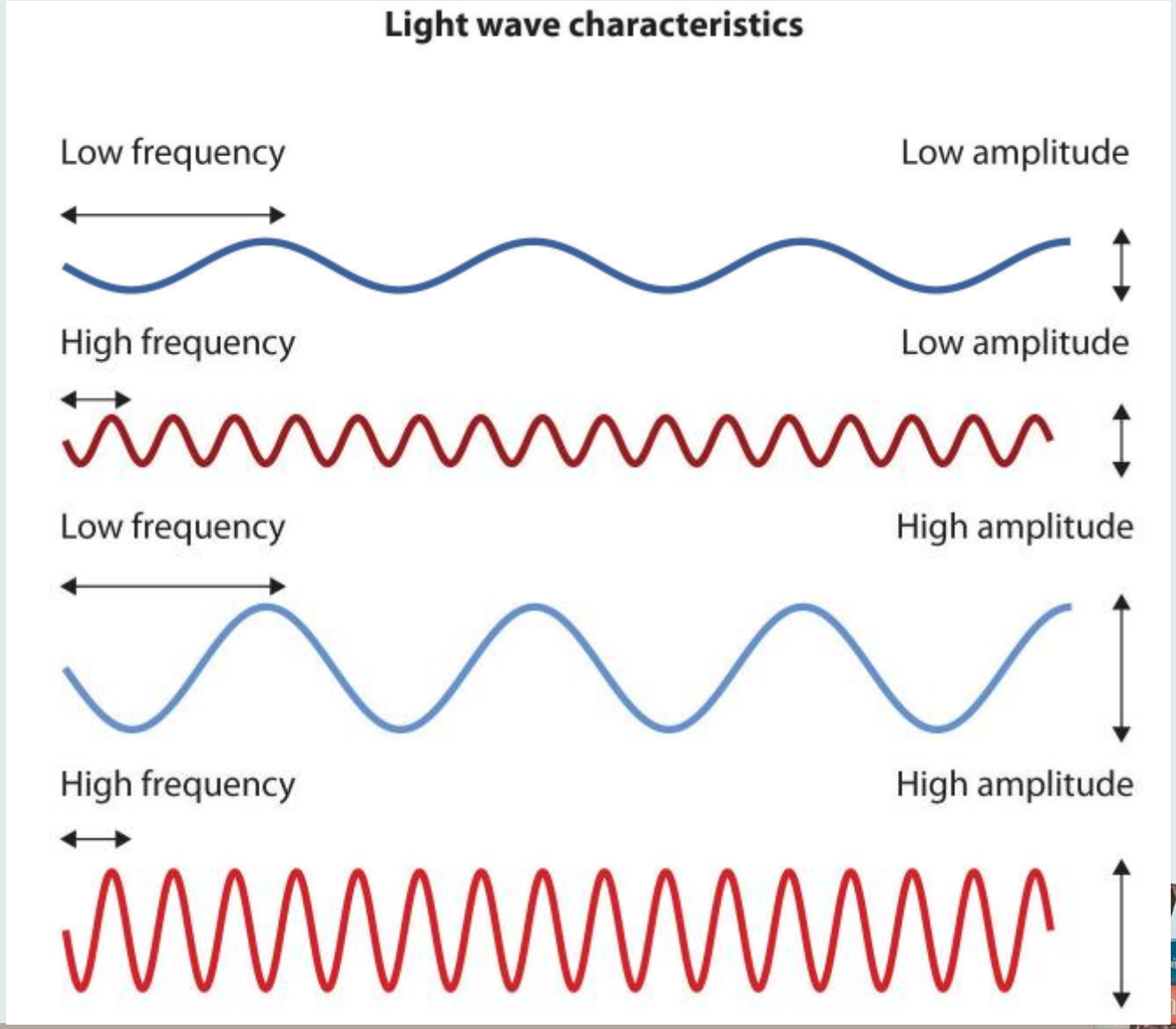
Digital Signal Processing

- Applications of digital signal processing systems
  - Speech and audio processing.
  - Image and video processing.
  - Military and telecommunication.
  - Healthcare and biomedical sector
  - Consumer electronics.
- Advantages of digital signal processing systems
  - Flexibility.
  - Error detection and correction features.
  - Data storage is easier.
  - Low cost.
- Disadvantages of digital signal processing systems
  - Higher power consumption.
  - Higher learning curve is required for operation.

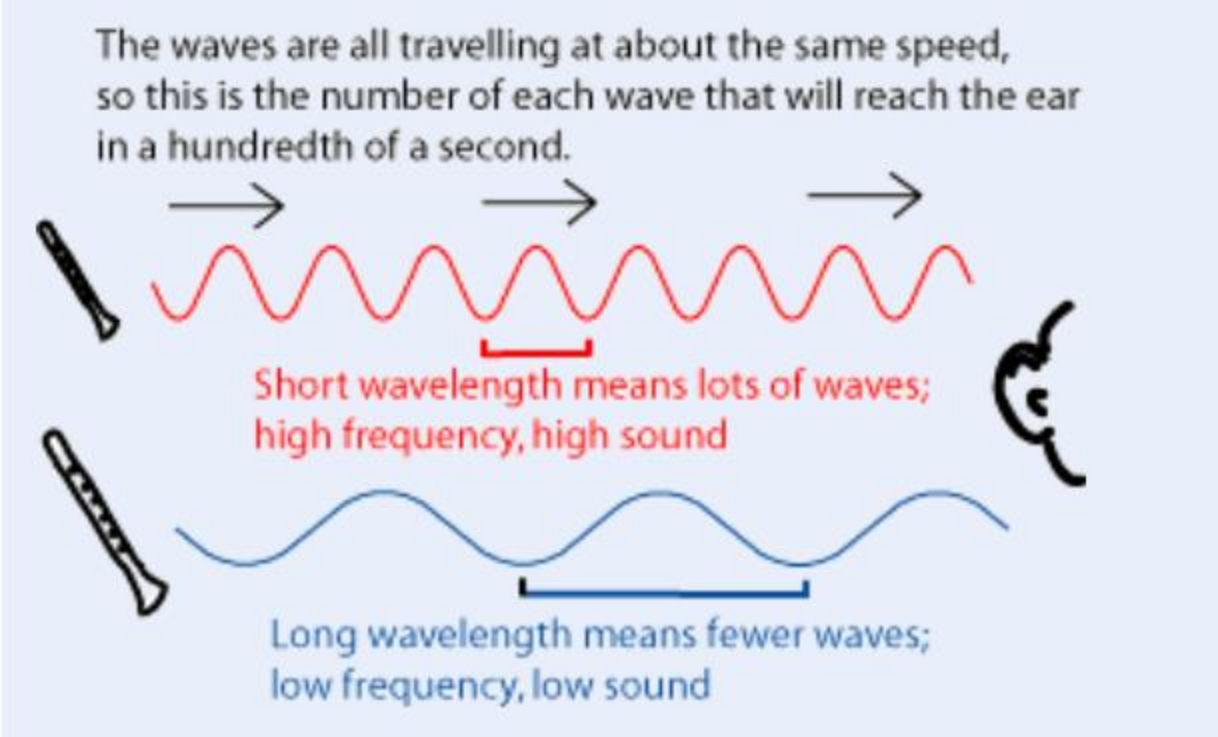
Digital Signal Processing

- Important concept in digital signal processing
  - **Fourier Transforms**: signal processing involves looking at the frequency representation of signals for both insight and computations.
  - **Noise**: any signal that is not desired. A huge part of signal processing is understanding how noise is affecting your data and how you can efficiently remove or manage it.
  - **Filters**: allow you to remove out specific portions of a signal at once, or allow you to remove noise from certain signals.

Signal wave

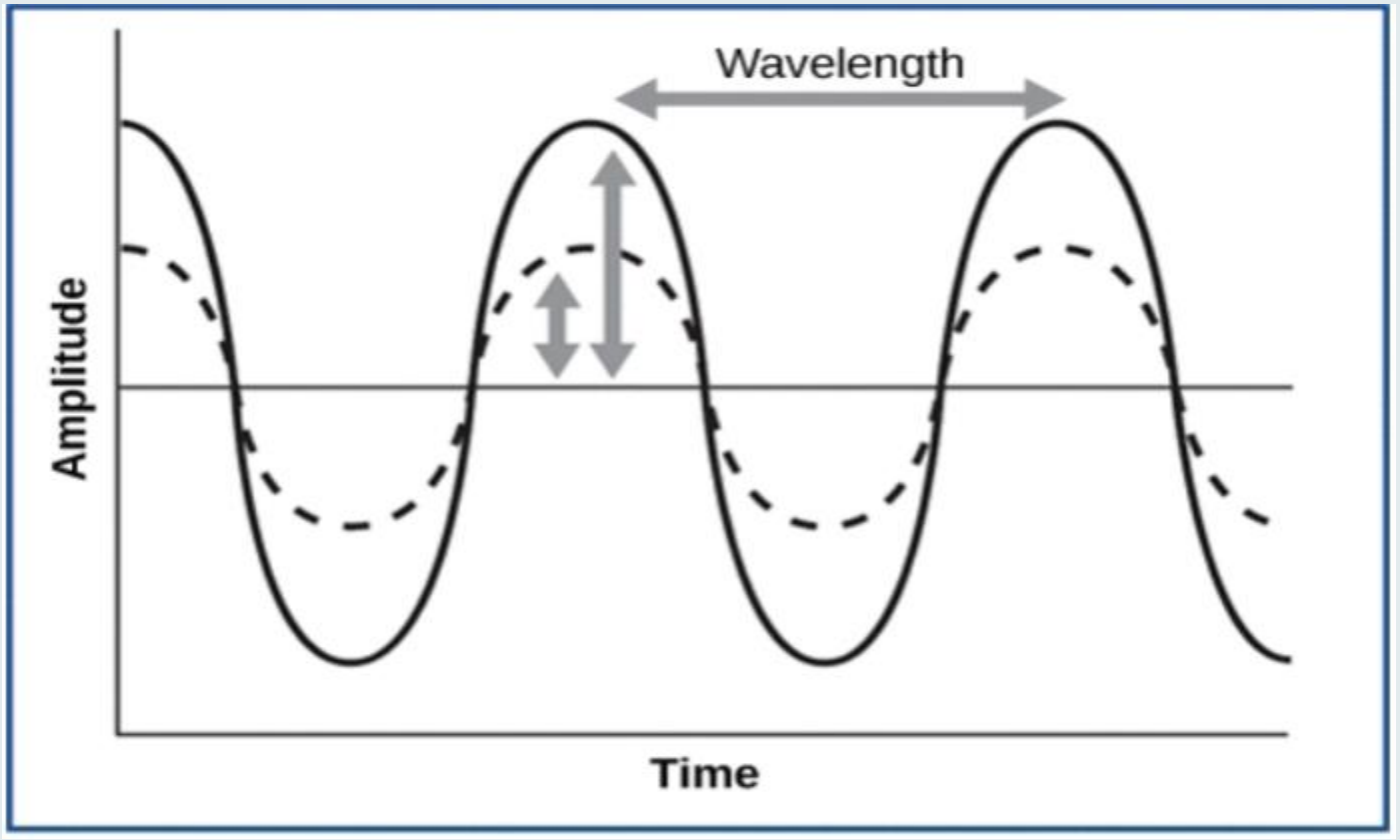

Light wave characteristics
Low frequency
Low amplitude
High frequency
Low amplitude
Low frequency
High amplitude
High frequency
High amplitude

Signal wave



Since the sounds are travelling at about the same speed, the one with shorter wavelength will go by more frequently, it has a higher frequency, or pitch.

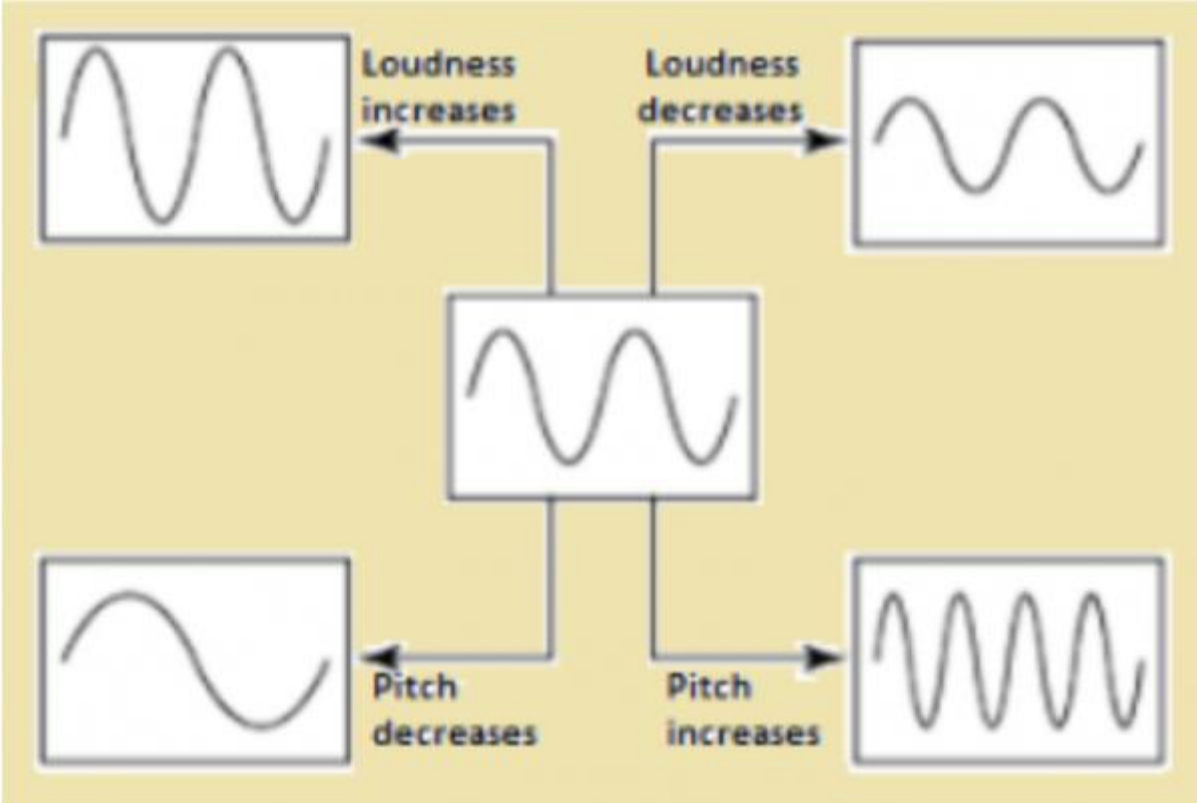
Signal wave



For sound waves, wavelength corresponds to pitch. Amplitude of the wave corresponds to volume. The sound wave shown with a dashed line is softer in volume than the sound wave shown with a solid line

Signal wave

Difference between pitch and loudness



The pitch of a sound depends on the frequency while loudness of a sound depends on the amplitude of sound waves.

Fourier Transforms and Spectrograms

- The most popular way to explore signals in the frequency domain is to use Fourier Transforms.
- A **Fourier Transform** is a reversible mathematical transform developed by Joseph Fourier in the early 1800s.
- The transform breaks apart a time series into a sum of finite series of sine and cosine functions (any signal can be broken down into a series of sine and cosine curves).
- The **Discrete Fourier Transform** (DFT) is a specific form of Fourier Transform applied to time wave.
- The DFT allows you to switch from working in the time domain to the frequency domain by using the amplitude and the frequency of the signal to build the frequency spectrum of the original time wave.

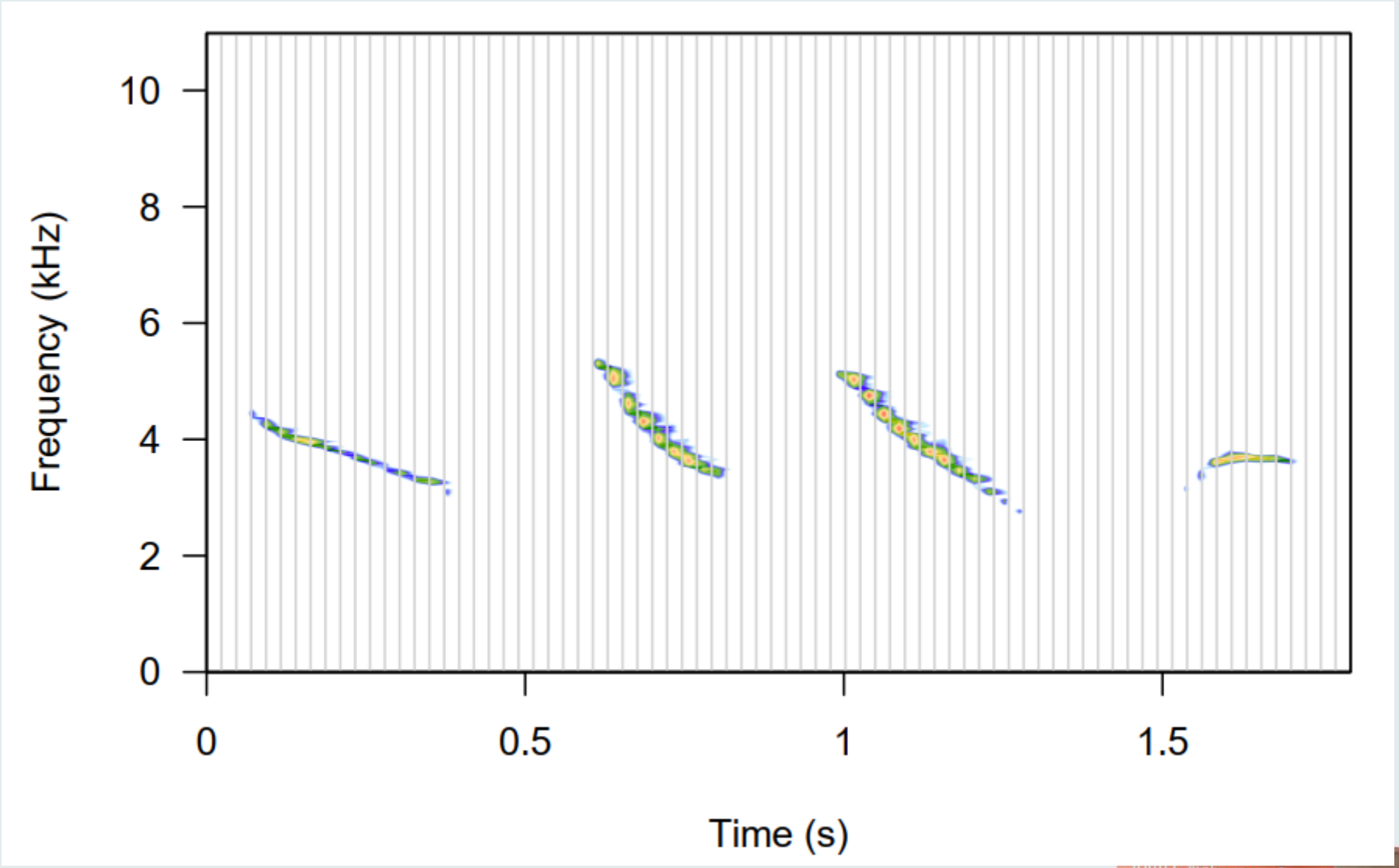
Fourier Transforms and Spectrograms

- To apply Fourier Transforms we use a window size. The length of the FT is controlled by the *window length*.
- Increasing the window size will increase frequency resolution, but this will also make the transform less accurate in terms of time as more signal will be selected for the transform.
- Computing a Fourier Transform on a whole sound or a single segment of the sound may not be enough information for what you desire.
- One great idea is to compute the DFT on successive sections along the entire sound signal.

Fourier Transforms and Spectrograms

- A window is “slided” along the signal and a DFT is computed at each of these slides.
- This process is called a **Short-Time Fourier Transform** (STFT). This method allows you to represent the signals in the frequency spectrum, while also maintaining most of the information from the time dimension.
- We can plot the results of an STFT in what is called a ***spectrogram*.**
- The spectogram is useful in bioacoustics, acoustics, and ecoacoustics (among others) because it allows you to represent three variables in a 2D plot, which avoids using a 3D representation that is genereally not appealing.

Fourier Transforms and Spectrograms



Analyzing Audio Data in R

- Within R, digitized sound can be stored in three categories of files:
  - uncompressed format (`.wav`): the full information is stored in a heavy file.
  - lossy compressed format (`.mp3`): the information is reduce d. Time, amplitude and frequency parameters can be impaired.
  - losslessly compressed format (`.flac`): the full information is stored in a reduced size file.
- All these formats generate binary files, sound being encoded into a succession of 0 and 1.

Other Resources

- Package library tuneR → https://cran.r-project.org/web/packages/tuneR/tuneR.pdf
- Package library seewave → https://cran.r-project.org/web/packages/seewave/seewave.pdf
- Notes on seewave (part 1) → https://cran.r-project.org/web/packages/seewave/vignettes/seewave_IO.pdf
- Notes on seewave (part 2) → https://cran.r-project.org/web/packages/seewave/vignettes/seewave_analysis.pdf
- A gentle introduction: R in digital signal processing → https://rpubs.com/eR_ic/dspr
- R for sound analysis tutorial → https://www.denaclink.com/post/20220317b-r-tutorial/
- Basics of Audio File Processing in R → https://medium.com/@taposhdr/basics-of-audio-file-processing-in-r-81c31a387e8e

Appendix

```
library(tuneR)
library(seewave)

# let's make a sine wave and play it
help(Wave)
sr = 8000       # the sampling rate
t = seq(0, 2, 1/sr)              # times in secs if sample for 2 seconds at 8KHz
y = (2^15-1)*sin(2*pi*440*t)                    # sine wave a 440 Hz scaled to fill out 16 bit range
#plot(y, type="l")
plot(y[1:500], type="l")
w = Wave(y, samp.rate = sr, bit = 16)      # make the Wave representation
play(w)

plot(y[1:500], type="l")
y1 = (2^15-1)*sin(2*pi*220*t)
w1 = Wave(y1, samp.rate = sr, bit = 16)
play(w1)

#Compare w and w1
par(mfrow=c(2,1)) #layout parallel
plot(y[1:500], type="l")
plot(y1[1:500], type="l")
#Compare in 1 layout (make redundant graph for easier comparison)
par(mfrow=c(1,1))
plot(y[1:500], type="l") #plot y above
lines(y1[1:500], lty=3, col="red") #plot y1 in plot y
```

Appendix

```
#Try create another
wsum = normalize(w+w1, unit='16') #must normalize to make it also 16hz
play(wsum)
plot((y+y1)[1:500], type='l') #plot from 1 to 500
w3 <- normalize(bind(w,w1,wsum), unit='16')
play(w3)
y3 <- c(y,y1, y+y1)
plot(y3, type='l')

#The above to create secara manual. Next we will use the function in tuneR
# Other types of waveforms,
?sine
Au1 <- sine(500, duration=100000) #100=freq, duration if 100000, means if divide 44100, its in 2+ second only
play(Au1)
writeWave(Au1, 'Audio1.wav') #save data
Au11 <- readWave('Audio1.wav') #Read the audio

Au2 <- noise(duration = 100000) #look noise function from ?sine
play(Au2) #seems sound broken tv/radio
Au2 <- noise(kind='pink',duration = 100000) #Change to pink noise (correlated noise).
play(Au2)

Au3 <- pulse(220, duration = 100000)
play(Au3)
```

Appendix

```
#plot 3 signal above
par(mfrow=c(3,1))
plot(Au1[1:1000]) #The first 1000 signal for Au1
plot(Au2[1:1000])
plot(Au3[1:1000])

#other signal in ?sine
Au4 <- sawtooth(100, duration = 100000)
play(Au4)
plot(Au4[1:1000])
plot(Au4[1:2000])


Au5 <- square(200, duration = 100000)
play(Au5)
plot(Au5[1:2000])

#Combine all Au1 sampai Au5
Asum = normalize(bind(Au1, Au2, Au3, Au4, Au5), unit='32')
play(Asum)

Au6 = normalize(bind(Au1+Au2), unit='32')
play(Au6)
```

Appendix

```
#data tico from notes seewave
data(tico)
par(mfrow=c(1,1))
timer(tico, f=22050, threshold=5, msmooth=c(50,0))#f is freq, msmooth to smooth the line
?timer
timer(tico, f=22050, threshold=5, msmooth=c(100,0))



sr = 8000
t = seq(0, 2, 1/sr)
y = (2^15-1)*sin(2*pi*440*t)
plot(y[1:500], type="l")
spectrum(y, span=20, log=c("no"))
plot.frequency.spectrum <- function(X.k, xlimits=c(0,length(X.k))) {
  plot.data  <- cbind(0:(length(X.k)-1), Mod(X.k))

  # TODO: why this scaling is necessary?
  plot.data[2:length(X.k),2] <- 2*plot.data[2:length(X.k),2]

  plot(plot.data, t="h", lwd=2, main="",
       xlab="Frequency (Hz)", ylab="Strength",
       xlim=xlimits, ylim=c(0,max(Mod(plot.data[,2]))))
}
```

Appendix

```
Yk <- fft(y)
plot.frequency.spectrum(Yk)
plot.frequency.spectrum(Yk[1:5000])#look roughly where the signal is
plot.frequency.spectrum(Yk[1:1000])#around 800+

data(tico)
Ticok <- fft(tico@left) #audio file have left channel and right channel, so we want only left channel
plot.frequency.spectrum(Ticok[1:(length(Ticok)/2)])#mirror, focus only one side

library(rpanel)
dynspec(tico, wl=1024, osc=T) #function seewave
spectro(tico) #give spectrogram
meanspec(tico) #obtain average for the whole time length

z = readWave("babycry.wav")
play(z)
timer(z, f=22050, threshold=5, msmooth=c(100,0))
Zk <- fft(z@left)
plot.frequency.spectrum(Zk)
plot.frequency.spectrum(Zk[1:20000])
dynspec(z, wl=1024, osc=T) #function seewave
spectro(z) #give spectrogram
meanspec(z) #obtain average for the whole time length
```