Intelligent Signal Processing (CM3065)

Course Notes

Felipe Balbi

July 17, 2021

Contents

Week 1

Key Concepts

- Use an audio editor and write simple programs to work with digital audio signals
- Explain the relevance of bit depth and sampling rates for digital audio signals and select appropriate parameters for different types of signals
- Describe the characteristics of sound waves and how they are perceived by humans

1.004 Technical information and tools used in this module

- Audacity
- P5.js
- Python
- ffmpeg

1.101 What is sound?

From a physics perspective, sound is a form of mechanical energy *produced* by vibrating matter. It needs a medium to propagate. In contrast to electromagnetic energy, sound cannot travel in a vacuum.

From a physiological or psychological perspective, sound is the reception of these waves and its processing by the brain. We capture sound waves with our inner ear and the brain converts it into what we call sound.

1.102 Characteristics of sound waves

In figure 1 below, we see several sine waves. Remember that the sine wave equation is given by $f(x) = A\sin(2\pi ft + \phi)$ where A is the amplitude, f is the frequency, t is the time, and ϕ is the phase.

The speed of sound waves depends on the temperature, elasticity, and density of the medium the sound is travelling through. As an example, at $0\,^{\circ}\mathrm{C}$ the speed of sound travelling through the air is around $331\,\mathrm{m\,s^{-1}}$. When the air is around $20\,^{\circ}\mathrm{C}$ the speed of sound is around $343\,\mathrm{m\,s^{-1}}$.

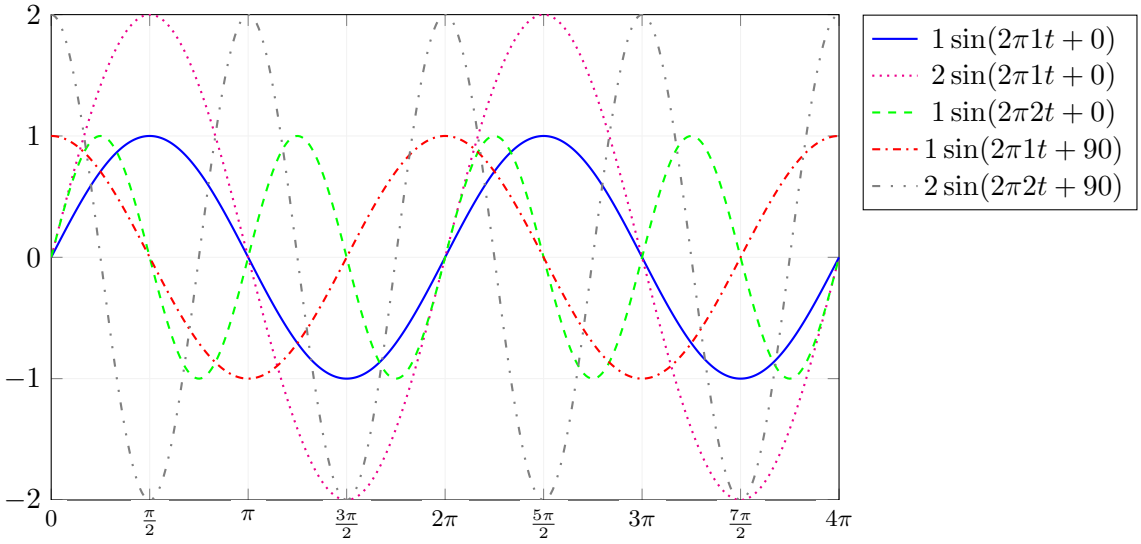

Figure 1: Sine waves

Waves can be represented in the domain of space. In this scenario we can characterize the wave by its wavelength and amplitude. The wavelength (λ) is given by the distance between two subsequent crests or troughs on the wave. The amplitude is given by the distance from the x axis to a peak or trough of the wave. All of these can be seen in figure 2 below. The pressure is measured in pascals (Pa) although we use dB for measuring sound amplitude.

Waves can also be represented in the domain of time. In this scenario we can characterize the wave by its amplitude, frequency, and period as shown in figure 3 below. Note that frequency, measured in hertz (Hz), is given by the inverse of the period, i.e. $f = \frac{1}{T}$ where T is the period in seconds.

There is a direct relation between sound speed, wavelength, and frequency given by:

$$v = f \cdot \lambda$$

For example, if the wavelength is $\lambda = 2\,\mathrm{m}$, $v = 4\,\mathrm{m\,s^{-1}}$, then:

$$\begin{aligned} 4\,\mathrm{m\,s^{-1}} &= f \cdot 2\,\mathrm{m} \\ f &= \frac{4\,\mathrm{m\,s^{-1}}}{2\,\mathrm{m}} \\ f &= 2\,\mathrm{Hz} \end{aligned}$$

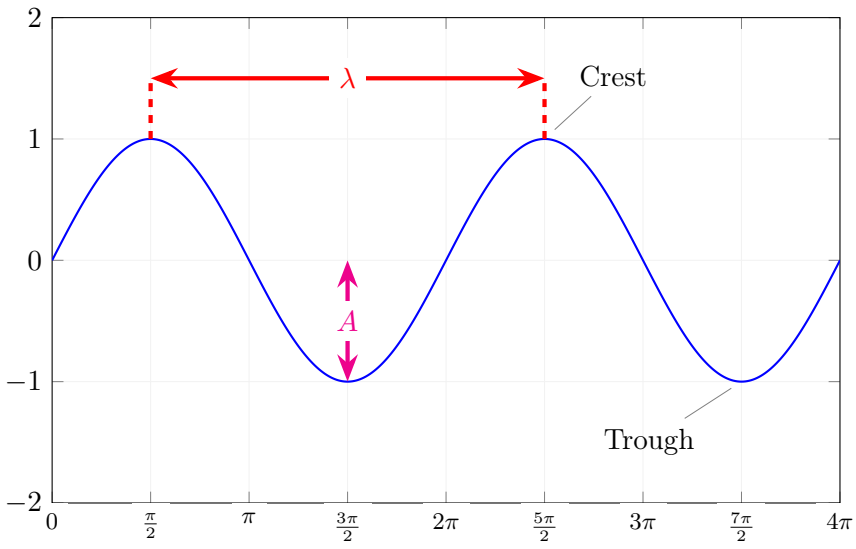

Figure 2: Sine wave in space domain

1.104 Human sound perception

Humans perceive sound through *pitch*, *loudness*, and *timbre*. In table 1 we summarize some of the properties of sound, both physical and psychological.

Table 1: Physical vs Psychological Properties of Sound

Physical	**Psychological**
Frequency	Pitch
Amplitude	Loudness
Waveform	Timbre
Wavelength	
Period	
Duration	

We can say that there is a relation between the matching pairs in table 1. That is, *Frequency* is related to *Pitch*, *Amplitude* is related to *Loudness*, and *Waveform* is related to *Timbre*. The relation, however, is not linear. That is, a certain change in frequency does not correspond to a similar change in pitch.

What is said above is a simplification of reality. A change in frequency also has an impact in loudness and timbre.

Pure Tone is a sound wave composed of a single sine wave. In figure 4 we have a representation of a pure tone while in figure 5 we have a representation of a composition of several sine waves.

The relationship between Pitch and Frequency is non-linear. In practice this means that as the pitch increases, a greater change in frequency is required. Another impor-

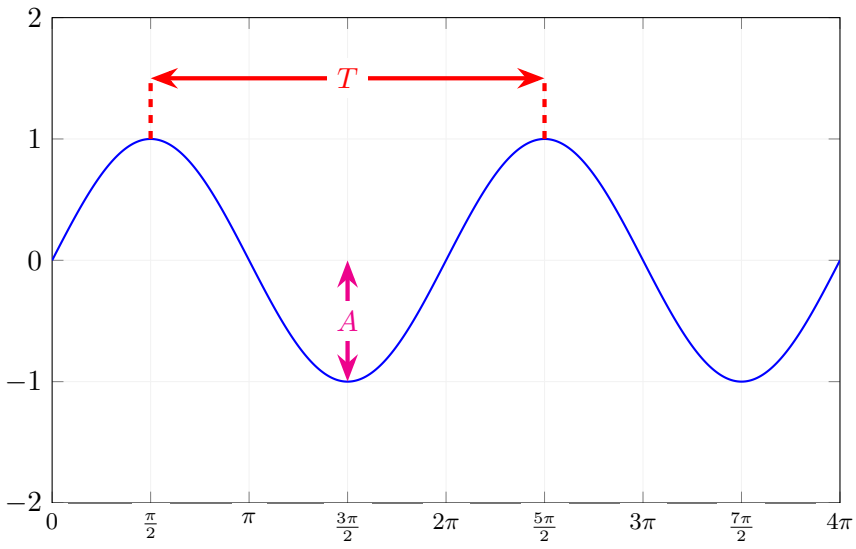

Figure 3: Sine wave in time domain

tant details is that humans can hear sound within the range from 20 Hz to 20 000 Hz. Frequencies below 20 Hz are known as infra-sound and frequencies above 20 000 Hz are known as ultrasound.

Loudness is a sensation to the perception of amplitude in sound waves. The pressure is measured in pascals (Pa). A quiet bedroom as night would measure sound pressure on the order of 630 µPa.

It turns out that measuring sound pressure in pascals is not very convenient, because of that we generally use a logarithmic scale called *dB SPL*. The conversion for pascal to dB is given by:

$$SPL = 20 \log_{10} \left(\frac{p}{p_{ref}} \right) dB$$

Where $p_{ref} = 20\,\mu Pa$, for sound pressure in air.

Timbre, or sound quality, helps us differentiate between two sounds with the same frequency and amplitude. Timbre is related to the waveform and the sound spectrum. In summary, we can hear the different component sound waves of a sound and interpret it as timbre.

1.106 Audio fundamentals

- Hosken, D. W. An introduction to music technology. (New York: Routledge, 2011). – Chapter 1: What is sound (pp.7–9) – Chapter 2: Sound properties and the waveform view (pp.17–26)

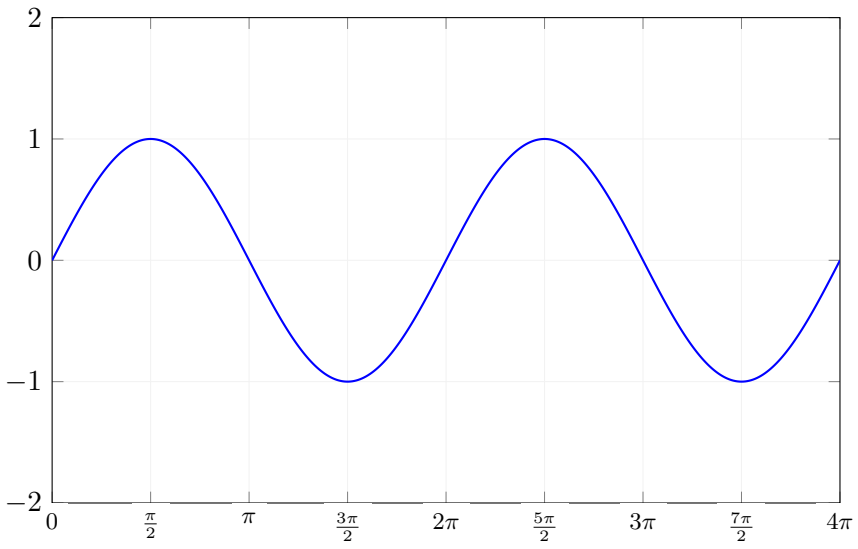

Figure 4: Pure Tone

1.201 The digital world: sample rate and bit depth

A sound level meter uses a microphone to convert air pressure changes into an electrical signal measured in volts. This is the first step to digitize a sound wave.

The signal output of a microphone is very small, therefore we need to amplifier to amplify the signal before passing it along to an Analog to Digital Converter, or ADC. The ADC measures the incoming voltage periodically and converts each sample to a numerical value within its dynamic range.

The sampling rate is the rate at which the ADC is *sampling* the input voltage. The sampling rate is given in Hertz (Hz). Each measurement of the input signal's amplitude is called a *sample*, the faster we sample, the better the quality, but the resulting amount of data per second will be larger. Table 2 below gives a set of common sampling rates and where they're used.

Table 2: Common Sampling Rates

Application	**Sample Rate**
CD	44.1 kHz
DVD	48 kHz
Professional Audio (?)	88.2 kHz
Professional Audio (?)	96 kHz

The Nyquist-Shannon Sampling Theorem states that the sampling rate must be at twice the highest frequency to be sampled. In other words, we can say that the Nyquist Frequency is $\frac{samplerate}{2}$ and any frequency above the Nyquist Frequency will not be recorded properly by the ADC, introducing artificial frequencies through aliasing.

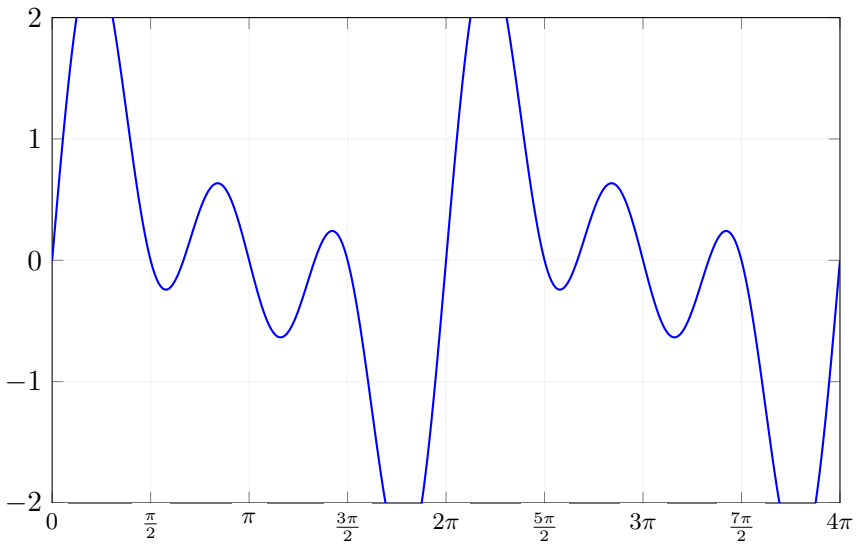

Figure 5: Composite Tone

To counter the problem of aliasing, most audio systems including an anti-aliasing frequencies which are basically a low-pass filter that cuts off frequencies above the Nyquist Frequency of the system.

Bit Depth is the number of bits used to record the samples. It's also referred to as Dynamic Range, sample width, or quantization level. The more bits we use, the more accurately we can measure the analogue waveform, but it results in more memory usage per sample. Common bit widths for digital audio are 8, 16, 24, and 32 bits.

Clipping occurs when the sampled amplitude is outside the ADC's dynamic range, then the value will either be clipped to 0 or maximum allowed value. Clipping generates perceivable distortions in the audio and should be avoided.

1.203 Digital audio representation: the time domain

There are two ways to represent audio:

Time Domain plotted as amplitude vs time

Frequency Domain plotted as amplitude vs frequency

Amplitude is usually represented in *dbFS* or db Full Scale. The relation is given by:

$$dBFS = 20 \log_{10}(|value|)$$

1.205 Digital audio

- Hosken, D. W. An introduction to music technology. (New York: Routledge, 2011). – Chapter 4: Analog and digital (p.51) – Chapter 4: The audio recording path (pp.51–53) – Chapter 5: Digital audio data (pp.72–85) – Chapter 6: Recording (pp.91–94)
- Smith, S. Digital signal processing: a practical guide for engineers and scientists. (Burlington, MA: Elsevier, 2013).

Week 2

Key Concepts

- Use an audio editor and write simple programs to work with digital audio signals
- Explain the relevance of bit depth and sampling rates for digital audio signals and select appropriate paramters for different types of signals
- Describe the characteristics of sound waves and how they are perceived by humans

1.401 Audio programming with p5.js: loading and playing back audio files

A simple example to load an audio file into `p5.js` is shown below:

```
let mySound;
let playStopButton;

function preload() {
  soundFormats('mp3', 'wav')
  mySound = loadSound('sound')
}

function playStopSound() {
  if (mySound.isPlaying()) {
    mySound.stop();
    /* mySound.pause() can be used */
    playStopButton.html('play')
  } else {
    mySound.play()
    playStopButton.html('stop')
  }
}

function setup() {
  createCanvas(400, 200)
  backgroun(180)

```

```
  playStopButton = createButton('play')
  playStopButton.position(20, 20)
  playStopButton.mousePressed(playStopSound)
}

function draw() {
}
```

1.403 Audio programming with p5: playback controls

Augmenting the previous application with some controls, we get:

```
let mySound;
let playStopButton;
let jumpButton;
let sliderVolume;
let sliderRate;
let sliderPan;

function preload() {
  soundFormats('mp3', 'wav')
  mySound = loadSound('sound')
}

function playStopSound() {
  if (mySound.isPlaying()) {
    mySound.stop();
    /* mySound.pause() can be used */
    playStopButton.html('play')
  } else {
    mySound.play()
    playStopButton.html('stop')
  }
}

function jumpSound() {
  let dur = mySound.duration()
  let t = random(dur)
  mySound.jump(t)
}

function setup() {
  createCanvas(400, 200)
  backgroun(180)
```

```
33
34    playStopButton = createButton('play')
35    playStopButton.position(200, 20)
36    playStopButton.mousePressed(playStopSound)
37
38    jumpButton = createButton('jump')
39    jumpButton.position(250, 20)
40    jumpButton.mousePressed(jumpSound)
41
42    sliderVolume = createSlider(0, 2, 1, 0.01)
43    sliderPosition(20, 25)
44    text('volume', 80, 20)
45
46    sliderRate = createSlider(-2, 2, 1, 0.01)
47    sliderRate.position(20, 70)
48    text('rate', 80, 65)
49
50    sliderPan = createSlider(-1, 1, 0, 0.01)
51    sliderPan.position(20, 115)
52    text('pan', 80, 110)
53  }
54
55  function draw() {
56    mySound.setVolume(sliderVolume.value())
57    mySound.rate(sliderRate.value())
58  }
```

1.404 Audio programming with p5: capture and record audio

We can also use `p5.js` to capture audio, an example is shown below.

```
1  let mic
2  let recorder
3  let soundFile
4  let state = 0
5
6  function setup() {
7    createCanvas(400, 400)
8
9    mic = new p5.AudioIn()
10   mic.start()
11
12   recorder = new p5.SoundRecorder()
13   recorder.setInput(mic)
```

```
14
15    soundFile = new p5.SoundFile()
16  }
17
18  function draw() {
19    background(200);
20
21    let vol = mic.getLevel()
22
23  }
24
25  function mouseClicked() {
26    if (getAudioContext().state !== 'running')
27      getAudioContext().resume()
28
29    switch (state) {
30    case 0:
31      background(255, 0, 0)
32      text('Recording', 40, 40)
33
34      recorder.record(soundFile)
35
36      state++
37      break
38    case 1:
39      background(0, 255, 0)
40      text('Click to play and download!', 40, 40)
41
42      recorder.stop()
43
44      state++
45      break
46    case 2:
47      background(200)
48      text('Click to record', 40, 40)
49
50      soundFile.play()
51      save(soundFile, 'output.wav')
52
53      state = 0
54      break
55    }
56  }
```

1.501 Audio programming with Python

Python has useful libraries for sound processing. The Python Audio Wiki has interesting information about these libraries. The Python In Music page also contains important information related to audio processing with Python.

The libraries we will use are:

- NumPy
- SciPy
- matplotlib
- ThinkDSP

1.503 Audio in Python

- Downey, A.B. Think DSP - Digital Signal Processing in Python. (Needham: Green Tea Press, 2014).

Week 3

Key Concepts

- Define homogeneity, additivity and shift invariance as they relate to linear time invariant systems
- Carry out convolution by hand and in code and analyse the process in terms of impulse responses and the properties of linear time invariant systems
- Use and write software to edit and to apply a range of effects processors to digital audio signals

2.104 Digital audio effects

- Audacity 'Index of effects, generators, analyzers and tools' (2021).
- Collins, N. Introduction to computer music. (Chichester: John Wiley & Sons, 2010).
  - Chapter 4: A compendium of marvellous digital audio effects (pp.143–155)

2.107 Processing digital audio in p5.js. The p5.Effect class

The simplest `p5js` to play audio is shown below.

```
let player;

function preload() {
  player = loadSound("sound.wav");
}

function setup() {
  let canvas = createCanvas(200, 200);
  background(0, 255, 0);
  canvas.mousePressed(mousePressed);
}

function mousePressed() {
  player.loop();
}
```

It builds the simple audio pipeline depicted in figure 6.

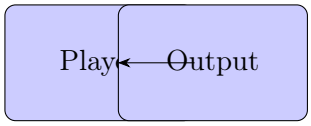

Figure 6: Simple Pipeline

When we want to add effects to the audio, we simply insert boxes between *Player* and *Output*. For example, for a reverb effect, our modified pipeline would look like the one shown in figure 7.

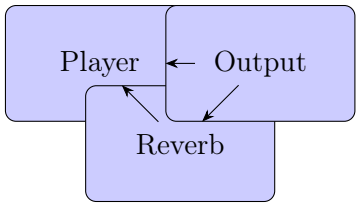

Figure 7: Pipeline with Reverb

Now we just modify our original code to include a reverb object. As shown below.

```
let player;
let reverb;

function preload() {
  player = loadSound("sound.wav");
  reverb = new p5.Reverb();
}

function setup() {
  let canvas = createCanvas(200, 200);
  background(0, 255, 0);
  canvas.mousePressed(mousePressed);
  reverb.process(player, 2, 3);
}

function mousePressed() {
  player.loop();
}
```

Week 4

Key Concepts

- Define homogeneity, additivity and shift invariance as they relate to linear time invariant systems
- Carry out convolution by hand and in code and analyse the process in terms of impulse responses and the properties of linear time invariant systems
- Use and write software to edit and to apply a range of effects processors to digital audio signals

2.201 Filtering. Implementation of an averaging low-pass filter

Signal Averaging is a technique that tries to remove random noise from a signal through the process of averaging. For the averaging, we generally use an N-point moving average, as shown in the equation below:

$$y(n) = \frac{1}{N} \sum_{k=0}^{N-1} x(n-k)$$

A 3-point moving average filter would be implemented in `Python` as shown below:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  values = np.array([3., 9., 3., 4.,
5                     5., 2., 1., 7.,
6                     9., 1., 3., 5.,
7                     4., 9., 0., 4.,
8                     2., 8., 9., 7.])
9
10 N = 3
11
12 averages = np.empty(20)
13
14 for i in range(1, 19):
```

```
    temp = (values[i-1] + values[i] + values[i+1])/N
    averages[i] = temp

# Preserve the edge values
averages[0] = values[0]
averages[19] = values[19]

plt.plot(values, 'b-', label='values')
plt.plot(averages, 'r==', label='averages')
plt.legend(loc="upper left")
```

We can also use `numpy.convolve` or `scipy.ndimage.uniform_filter1d` to implement the moving average:

```
window = np.ones(3)
window /= sum(window)
print(window)
plt.plot(window, 'yo-')
plt.xlabel('Index')

averages = np.convolve(values, window, mode='same')
plt.plot(values, 'b-', label='values')
plt.plot(averages, 'r--', label='averages')
plt.legend(loc="upper left")
```

A moving average filter can be used as a simple low-pass filter as it removes high frequency components from the original signal.

```
import numpy as np
import matplotlib.pyplot as plt
from thinkdsp import SquareSignal, Wave

# Suppress scientific notation for small numbers
np.set_printoptions(precision=3, suppress=True)

signal = SquareSignal(freq=440)
wave = signal.make_wave(duration=1.0, framerate=48000)
wave.make_audio()

window = np.ones(15)
window /= sum(window)
print(window)
plt.plot(window, 'yo-')
plt.xlabel('Index')

```

```
segment = wave.segment(duration=0.01)
plt.plot(segment.ys, 'b-')
plt.xlabel('Samples')

averages = np.convolve(wave.ys, window, mode='same')
smooth = Wave(averages, framerate=wave.framerate)
smooth.make_audio()

plt.plot(wave.ys[:500], 'b-', label='original')
plt.plot(smooth.ys[:500], 'r--', label='filtered')
plt.xlabel('Samples')
plt.legend(loc="upper left")
```

2.203 FIR (Finite Impulse Response) filters

An Averaging Low-Pass Filter is a kind of filter known as Finite Impulse Response Filter. It's *Finite* because its impulse response settles to zero in finite time.

The filter's response is given by the finite series:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

If we let $h(n) = \frac{1}{N}$, then we have our Averaging Low-Pass Filter as before.

To produce different impulse responses, all we have to do is modify the `window` coefficients from the previous code.

```
window = np.array([0.1, 0.2, 0.2, 0.2, 0.1])
print(window)
plt.plot(window, 'yo-', linewidth=2)
decorate(xlable='Index')

filtered = np.convolve(wave.ys, window, mode='same')
filtered_violin = Wave(filtered, framerate=wave.framerate)
filtered_violin.make_audio()
```

2.205 LTI systems

A Linear Time-Invariant System is a system that produces an output signal from any input signal subject to the constraints of linearity and time-invariance[1].

A (digital) signal is just a list of numbers. For audio signal, each number represents the amplitude of the signal sampled over time.

[1] `https://en.wikipedia.org/wiki/Linear_time-invariant_system`

In this context, a System is something we use to process signals, filters for example. If the system happens to be **linear**, **time-invariant** (LTI) system, we can represent its behavior as a list of numbers known as **Impulse Response**. An Impulse Response is the response produced by an LTI system to the impulse signal.

Considering figure 8, what an LTI system would do is move the circles up and down depending on the system's parameters.

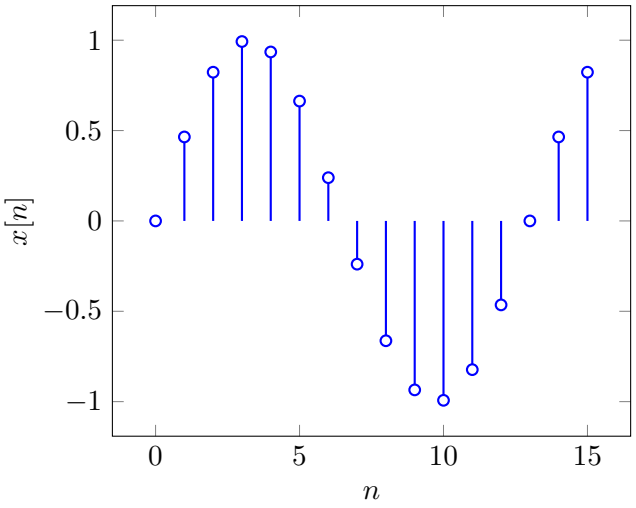

Figure 8: Discrete Sine Wave

LTI Systems use Convolution to process the input signal. Convolution is a mathematical term for a method of combining two functions f and g in a way that g expresses how to modify f. We denote the convolution of f and g and $f * g$.

LTI Systems have three properties:

Homogeneity Linear with respect to scale. Scaling input and applying filter is the same as applying filter and scaling output by the same factor: $0.5f * 0.5g = 0.5(f * g)$

Additivity Can separately process simple signals and add results together. Adding two signals f_1 and f_2 then applying filter is same as applying filter to each signal and adding results: $(f_1 * g) + (f_2 * g) = (f_1 + f_2) * g$

Shift Invariance Later signal results in a later response. Shifting a signal's phase then applying the filter is the same as applying the filter then shifting the output.

Week 5

Key Concepts

- Describe signal analysis with the discrete cosine transform (DCT)
- Review spectral analysis and describe spectral analysis with the discrete Fourier transform (DFT)
- Use the discrete Fourier transform (DFT) to complex analysis and fast convolution

3.001 Review of spectral analysis

When looking at the time domain, we don't get a feeling for how the audio is going to sound because we're oblivious to the component frequencies in the sound.

A Frequency Analyzer will convert the signal to the frequency domain and let us *see* all the component frequencies in the audio sample.

3.003 Spectral analysis of periodic signals

"Fourier Decomposition is a formalisation of the process of figuring out how to make a given signal out of a set of sine waves."

$$S_N(x) = \frac{a_0}{2} + \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi}{P} nx\right) + b_n \sin\left(\frac{2\pi}{P} nx\right) \right)$$

The Discrete Cosine Transform is given by

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\right]$$

where $k = 0, 1, \ldots, N-1$.

Week 6

Key Concepts

- Describe signal analysis with the discrete cosine transform (DCT)
- Review spectral analysis and describe spectral analysis with the discrete Fourier transform (DFT)
- Use the discrete Fourier transform (DFT) to complex analysis and fast convolution

3.101 Complex synthesis

The Discrete Cosine Transform ignores phase. In order to consider phases we will need Complex Numbers. Complex numbers are of the form

$$a + ib$$

where a is a *real* number and b is the *imaginary* part of the number.

In Python the character i is replaced with a j.

```
c = 1j # Creates a new complex number
print(c.real, c.imag)
```

To compute the waveforms from complex numbers we rely in the exponential function e^x (`np.exp(x)` in Python).

Week 7

Key Concepts

- Describe audio signal feature extraction and the characteristics of time domain features and frequency domain features
- Use p5.sound and the JavaScript library Meyda to write software to extract audio signal features
- Use Python and the library Librosa to write software to extract audio signal features

4.001 Introduction to audio feature extraction

An audio feature is a characteristic of an audio audio that gives insight into what the signal contains.

Some examples of audio features are listed below:

- Energy
- Spectrum
- Fundamental Frequency
- Loudness
- Brightness
- Pitch
- Rhythm
- ...

These features can be programmatically extracted from audio signals by using certain algorithms. There are algorithm for real-time and non-real-time feature extraction.

Librosa is an audio and music processing library for Python which implements feature extraction. Meyda is a similar library for JavaScript.

Audio features have several applications, including:

- Real-time audio visualizations
- Visual score generation

- Speech recognition
- Music recommendation
- Music genre classification
- Feature-based synthesis
- Feature extraction linked to audio effects
- ...

From the time-domain representation, we can extract the following features:

- Energy
- RMS energy
- Zero crossing rate
- Amplitude envelope
- ...

From the frequency-domain representation, we can extract the following features:

- Spectral centroid
- Spectral rolloff
- Spectral flatness
- Spectral flux
- Spectral slope
- Spectral spread
- Mel-frequency cepstral coefficients
- Chroma
- ...

The Root-mean-square energy (RMSE) is formalised as shown below

$$RMSE = \sqrt{\frac{1}{N}\sum_{n} \left|x(n)\right|^2}$$

4.003 Audio feature extraction

- Collins, N. Introduction to computer music. (Chichester: John Wiley & Sons, 2010). Chapter 3: Feature extraction (pp.99–111)

4.101 Real-time audio visualisations with p5.js. – time domain features

To extract the amplitude from audio file using p5.js, we can use the Amplitude object.

4.102 Real-time audio visualisations with p5.js – frequency domain features (1)

To extract features from the frequency domain, we need an FFT object.

4.105 Real-time audio visualisations with Meyda

Meyda is an audio feature extraction library for JavaScript. It's written to use the Web Audio API. It supports both real-time and offline feature extraction.

4.107 Meyda: an audio feature extraction library for the Web Audio API

- Fiala, J., N. Segal and H.A. Rawlinson 'Meyda: an audio feature extraction library for the Web Audio API' in Web Audio Conference (WAC 2015), 2015, pp.1-6.

Week 9

Key Concepts

- Describe Automatic Speech Recognition (ASR) and Hidden Markov Models (HMM)
- Use the p5.js library p5.Speech to write speech recognition software
- Use the Python libraries pocketsphinx, vosk and Mozilla DeepSpeech to write Offline ASR software

5.004 Hidden Markov models

Given a set of observations `a a b a c c b b`, to convert it into a Hidden Markov Model the first step is create a state transition diagram from our observations, as shown in figure 9 below.

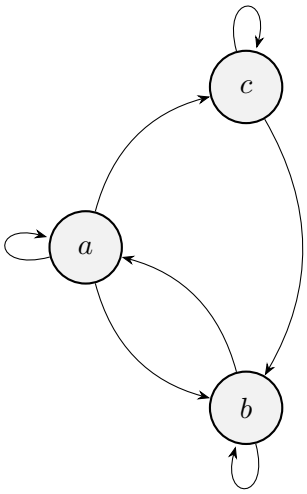

Figure 9: State Diagram From Observations

Afterwards, we annotate the edges with their weights. For example, each of the transitions from a appears $\frac{1}{3}$ of the time. The resulting state transition diagram with weights is shown in figure 10 below.

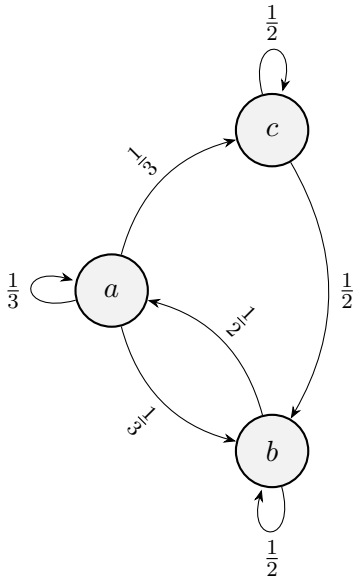

Figure 10: State Diagram With Weights

Given a Markov Model like this, we can randomize its execution. We choose an initial state and roll a die to choose the next state. In the model above, every transition from any state has the same probability, but that need not be the case.

Note that *States* are not the same *Observations*. *States* encode the probability distribution of *Observations*. The goal is to figure out the most likely sequence of states given the observations.

Week 11

Key Concepts

- Capturing, representing and processing camera input, part 1

6.001 2D image representation: recap

2D images are arrays of pixels with a width and a height. They can be black and white or in color.

Images also have a resolution, i.e. the number of pixels in the image. It's not always desirable to have the highest amount of pixels possible, sometimes less pixels can be advantageous.

We tend of think of them as being represented in a cartesian coordinate system but, in reality, they are just a list. Figure 11 below shows an example of a cartesian coordinate system.

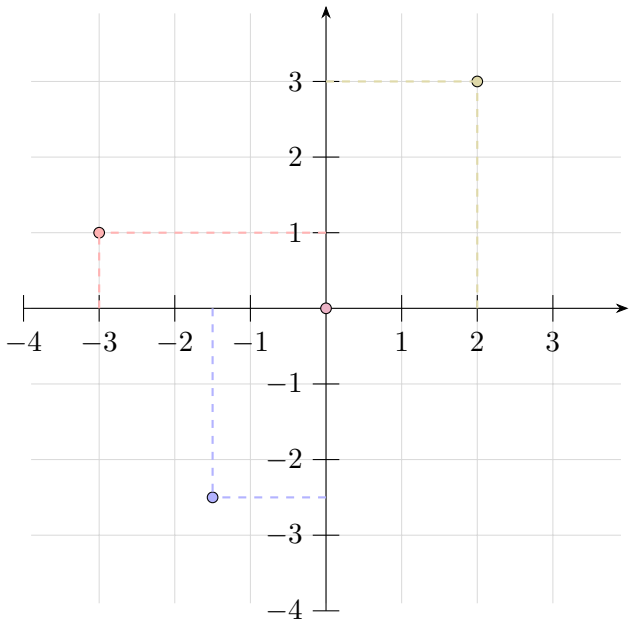

Figure 11: Cartesian Coordinates

The origin is at $(0, 0)$, usually depicted right in the middle, but it need not be. Because we're so used with the cartesian system, we may expect computer images to be

represented the same way, however that's not the case.

The origin on computer image files, is usually at the top left corner. Moreover, the data is usually represented as a single vector as we can easily transform a 1D coordinate to 2D coordinate given height and width.

When it comes to JavaScript, if we want to do any sort of image manipulation, there are three steps to follow:

1. Create a pixel array (or grab it from the canvas)
2. Iterate through the pixel array to apply our manipulations
3. Write resulting pixels back to the canvas

One caveat is that JavaScript is always expecting pixels in RGBA format even when they're grayscale. The canvas in JavaScript can be treated as an RGBA pixel array where every 4 values represent each of our four channels. The resolution of each channel is 8 bits.

Usually we process raw, uncompressed pixel data. The image data is the same resolution as that of the canvas. This means that if we're processing a 1080p video at 25 frames per second, we're dealing with

$$1920 \cdot 1080 \cdot 25 \cdot 32 \approx 198 MB/s$$

of information.

6.003 Realtime pixel pushing in the browser

A simple example:

```
<!DOCTYPE html>
<html>
  <body>
    <canvas></canvas>
    <script>
     let canvas = document.querySelector("canvas");
     let context = canvas.getContext("2d");

     canvas.width = 640;
     canvas.height = 480;

     let imageData = context.createImageData(canvas.width, canvas.height);

     function draw() {
         for (var y = 0; y < imageData.height; y++) {
             for (var x = 0; x < imageData.width; x++) {
```

```
                let pos = (x + y * imageData.width) * 4;

                imageData.data[pos] = Math.random()*255;
                imageData.data[pos+1] = Math.random()*255;
                imageData.data[pos+2] = Math.random()*255;
                imageData.data[pos+3] = 255;
            }
        }

        context.putImageData(imageData, 0, 0);
        requestAnimationFrame(draw);
    }

    requestAnimationFrame(draw);
   </script>
  </body>
</html>
```

6.006 Processing webcam data

To process incoming video from a webcam, the process is a little more involved, here are the necessary steps:

1. Open the webcam
2. Copy a frame image into the canvas
3. Grab the canvas pixels
4. Process those pixels
5. Write pixels back to the canvas

Sample code follows

```
<!DOCTYPE html>
<html>
  <body>
    <video id="video" width="0" height="0"></video>
    <canvas id="canvas1"></canvas>
    <canvas id="canvas2"></canvas>
    <script language="javascript" type="text/javascript">
     let width = 400;
     let height = 300;

```

```
11      let canvas = document.getElementById("canvas1");
12      let context = canvas.getContext("2d");
13
14      let canvas2 = document.getElementById("canvas12");
15      let context2 = canvas2.getContext("2d");
16
17      canvas.addEventListener("mousemove", getMouse, false);
18
19      canvas.width = width;
20      canvas.height = height;
21      canvas2.width = width;
22      canvas2.height = height;
23
24      let video = document.getElementById("video");
25
26      navigator.mediaDevices.getUserMedia({video: true}).then((stream) => {
27          video.srcObject = stream;
28          video.play();
29      });
30
31      let imageData = 0;
32      let resultData = 0;
33
34      function draw() {
35          context2.drawImage(video, 0, 0, width, height);
36
37          imageData = context2.getImageData(0, 0, width, height);
38          resultData = context.getImageData(0, 0, width, height);
39
40          for (var i = 0; i < height; i++) {
41              for (var j = 0; j < width; j++) {
42                  let pos = (j + width * i) * 4;
43
44                  let r = imageData.data[pos];
45                  let g = imageData.data[pos+1];
46                  let b = imageData.data[pos+2];
47
48                  /*
49                   * RGB to Luma:
50                   * https://en.wikipedia.org/wiki/Luma_(video)
51                   */
52                  let grayscale = 0.2126*r + 0.7152*g + 0.0722*b;
53
54                  resultData.data[pos] = grayscale;
```

```
55                  resultData.data[pos+1] = grayscale;
56                  resultData.data[pos+2] = grayscale;
57                  resultData.data[pos+3] = 255;
58              }
59          }
60
61          context.putImageData(resultData, 0, 0);
62          requestAnimationFrame(draw)
63      }
64
65      requestAnimationFrame(draw);
66
67      function getMouse(pos) {
68          mouseX = pos.layerX;
69          mouseY = pos.layerY;
70      }
71     </script>
72   </body>
73 </html>
```

Week 13

7.001 Introduction to Webcam Processing

Implementing rotation:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
          margin: 0px;
          padding: 0px;
      }
    </style>
  </head>
  <body>
    <video id="video" width="0" height="0"></video>
    <canvas id="canvas1"></canvas>
    <canvas id="canvas2"></canvas>

    <script language="javascript" type="text/javascript">
      let width = 400;
      let height = 400;
      let mouseX = 1;
      let mouseY = 1;
      let zoomX = 1.0;
      let zoomY = 1.0;
      let anchorX = 200;
      let anchorY = 200;
      let offsetX = 0;
      let offsetY = 0;

      let canvas = document.getElementById("canvas1");
      let context = canvas.getContext("2d");

      let canvas2 = document.getElementById("canvas2");
      let context2 = canvas.getContext("2d");

```

```
34        canvas.addEventListener('mousemove', getMouse, false);
35
36        canvas.width = width;
37        canvas.height = height;
38
39        canvas2.width = width;
40        canvas2.height = height;
41
42        let video = document.getElementById("video");
43
44        navigator.mediaDevices.getUserMedia({video: true}).then((stream) => {
45          video.srcObject = stream;
46          video.play();
47        });
48
49        let imageData2 = context.getImageData(0, 0, width, height);
50
51        function computeX(theta, i, j) {
52          return Math.floor((Math.cos(theta) / zoomX) *
53                            (j - (offsetX + anchorX)) -
54                            (Math.sin(theta) / zoomX) *
55                            (i - (offsetY + anchorY))) +
56                            anchorX;
57        }
58
59        function computeY(theta, i, j) {
60          return Math.floor((Math.sin(theta) / zoomY) *
61                            (j - (offsetX + anchorX)) +
62                            (Math.cos(theta) / zoomY) *
63                            (i - (offsetY+anchorY))) +
64                            anchorY;
65        }
66
67        let draw = function() {
68          let theta = mouseX / width * Math.PI * 2;
69
70          context2.drawImage(video, 0, 0, width, height);
71
72          let imageData = context2.getImageData(0, 0, width, height);
73          let data = imageData.data;
74
75          for (var i = 0; i < height; i++) {
76            for (var j = 0; j < width; j++) {
77              let x = computeX(theta, i, j);
```

```
78              let y = computeY(theta, i, j);
79              let pos = ((i * width) + j) * 4;
80              let posXY = ((y * width) + x) * 4;
81
82              imageData2.data[pos] = data[posXY];
83              imageData2.data[pos+1] = data[posXY+1];
84              imageData2.data[pos+2] = data[posXY+2];
85              imageData2.data[pos+3] = data[posXY+3];
86            }
87          }
88
89          context.putImageData(imageData2, 0, 0);
90          requestAnimationFrame(draw);
91        }
92
93        requestAnimationFrame(draw);
94
95        function getMouse(mousePosition) {
96          mouseX = mousePosition.layerX;
97          mouseY = mousePosition.layerY;
98        }
99      </script>
100   </body>
101 </html>
```

7.004 - Get the brightest pixel

Brightest pixel detection can be used for implementing Autoexposure by computing the different from the brightest pixel to 255 and adding that delta to every pixel.

The new `draw()` function for brightest pixel detection is shown below.

```
1  function draw() {
2    context.drawImage(video, 0, 0 width, height);
3    imageData = context.getImageData(0, 0, width, height);
4
5    let brightest = 0;
6    let x = 0;
7    let y = 0;
8
9    for (var i = 0; i < height; i++) {
10     for (var j = 0; j < width; j++) {
11       let pos = ((width * i) + j) * 4;
12       let r = imageData.data[pos];
13       let g = imageData.data[pos+1];
```

```
14        let b = imageData.data[pos+2];
15
16        if (r + b + g > brightest) {
17          brightest = r + b + g;
18          x = j;
19          y = i;
20        }
21      }
22    }
23
24    context.fillStyle = "red";
25    context.fillRect(x, y, 20, 20);
26  }
```

Week 14

7.101 Reference materials

- Bovik, A.C. The Essential Guide to Video Processing. (Boston: Academic Press, 2009).
  - Chapter 3: Motion Detection and Estimation (pp. 31-67)
  - Chapter 7: Motion Tracking in Video (pp. 175-230)
- Matsukura M. Brockmole J. R. Henderson J. M. (2009). Overt attentional prioritization of new objects and feature changes during real-world scene viewing. Visual Cognition, 17, 835–855. `https://www3.nd.edu/~jbrockm1/matsukuraEtAl_VC2009.pdf`

Week 15

8.002 - Face Detection Basics

A simple approach to face detection would be to first analyse the image data and attempt to understand and extract structural features of faces. Afterwards, we can use the image features as a template of a face to search through image data until we find a match.

Image Features are information derived from an image by a transform. These features try to *say* something about what's in the image.

SIFT Scale Invariant Feature Transform

SURF Speeded Up Robust Feature

HOG Histogram of Oriented Gradients

All of the transforms above can be used to identify objects in an image. Objects can be tracked with image features.

One of the methods for tracking points is called *Optical Flow*. This is not a specific algorithm. Rather, it's a way of modelling how the eyes track motion. The basic idea is to decide on a specific set of pixels from an image and track them, measuring how they have moved. It's common to plot a quiver plot showing the magnitude and direction of motion between points in the frame to show the *Optical Flow*.

The Lucas-Kanade method is often used to track the movement of points.

8.004 - Creating an image gradient feature

Image Gradients are simple edge vectors, they are 1D convolution filters. Running such a filter accross the X dimension of an image substracts the previous pixel from the next and places the result in the current. It's similar to frame differencing. The implemention is shown below

```
1  var mouseX;
2  var mouseY;
3  var imageObj = new Image();
4
5  imageObj.src = "zolner.jpg";
6
7  var canvas = document.getElementById('myCanvas');
8  var canvas2 = document.getElementById('myCanvas2');
```

```
9
10 var context = canvas.getContext('2d');
11 var context2 = canvas2.getContext('2d');
12 var imageWidth = imageObj.width;
13 var imageHeight = imageObj.height;
14
15 context2.drawImage(imageObj, 0, 0);
16
17 var imageData = context2.getImageData(0, 0, imageWidth, imageHeight);
18 var data = imageData.data;
19
20 /* X gradient */
21 var imageData2 = context.getImageData(0, 0, imageWidth, imageHeight);
22
23 /* Y gradient */
24 var imageData3 = context.getImageData(0, 0, imageWidth, imageHeight);
25
26 for (var i = 1; i < imageHeight-1; i++) {
27   for(var j = 1; j < imageWidth-1; j++) {
28     var pos = (imageWidth * i + j) * 4;
29
30     var collm1 = (imageWidth * (i - 1) + j - 1) * 4;
31     var collp1 = (imageWidth * (i + 1) + j + 1) * 4;
32
33     var rowm1 = (imageWidth * i + j - 1) * 4;
34     var rowp1 = (imageWidth * i + j + 1) * 4;
35
36     /* X gradient */
37     imageData2.data[pos+0] = -1 * data[rowm1+0] + data[rowp1];
38     imageData2.data[pos+1] = -1 * data[rowm1+1] + data[rowp1];
39     imageData2.data[pos+2] = -1 * data[rowm1+2] + data[rowp1];
40     imageData2.data[pos+3] = 255;
41
42     /* Y gradient */
43     imageData3.data[pos+0] = -1 * data[collm1+0] + data[collp1];
44     imageData3.data[pos+1] = -1 * data[collm1+1] + data[collp1];
45     imageData3.data[pos+2] = -1 * data[collm1+2] + data[collp1];
46     imageData3.data[pos+3] = 255;
47   }
48 }
49
50 /* view X gradient */
51 context.putImageData(imageData2,0,0);
52
```

```
53 /* view Y gradient */
54 //context.putImageData(imageData3,0,0);
```

8.006 Image gradients: read and run the code!

- https://mimicproject.com/code/e9e74da1-ecd7-719d-9c8c-51d5b52d4cad
- https://mimicproject.com/code/6180b873-3167-8d8f-8ba5-a251605f6423

8.007 - Histogram of Oriented Gradients

Similar types of images might have similar kinds of histograms. It's a simple way of detecting people without ML.

8.009 Histogram of oriented gradients: read and run the code!

- https://mimicproject.com/code/6180b873-3167-8d8f-8ba5-a251605f6423

8.010 Reading materials for classic face detection techniques

- Ming-Hsuan Yang, David Kriegman, and Narendra Ahuja, "Detecting Faces in Images: A Survey", IEEE Transactions on Pattern Analysis and Machine Intelligence https://faculty.ucmerced.edu/mhyang/papers/pami02a.pdf
- Bovik, A.C. The Essential Guide to Video Processing. (Boston: Academic Press, 2009). Chapter 20: Face Recognition from Video (pp. 31-67)
- D. G. Ganakwar and V. K. Kadam, "Comparative Analysis of Various Face Detection Methods," 2019 IEEE Pune Section International Conference (PuneCon), 2019, pp. 1-4, doi: 10.1109/PuneCon46936.2019.9105893

Week 16

8.101 Face detection with machine learning: face API

Face-API is a really way to detect faces using machine learning. It can also detect some simple emotions. It's essentially pre-trained face detection model which we can use from the browser.

8.103 Face-api: read and run the code!

- `https://mimicproject.com/code/d87b4f42-c131-ca7a-127f-c6df8f475329`

Week 17

9.001 Quick review: Sample Rate, Bit Depth and Bit Rate

Sampling rate The frequency at which a signal is sampled, or measured

Bit depth How precise is each sample, or measurement

Nyquist sampling theorem A sampling rate f_{sr} can accurately sample a signal up to $\frac{1}{2}f_{sr}$. In other words, to sample any signal of frequency f, we need a sampling rate of at least $2f$

Bit rate (uncompressed) Bits per sample times sample rate: $bits\_per\_sample \times sample\_rate$

Bit rate (compressed) Samples are not represented the same way. kbit/sec is a common unit.

9.003 Lossless Compression with FLAC

Lossless compression You can accurately restore the original signal

Lossy compression Some data has been lost due to compression which results in parts of the original signal being lost

Codec set of algorithms for processing a digital signal into some specified format

Container structure for the storage of encoded data streams

Zip, *Gzip*, *XZ*, *Bzip2* and similar have not been designed for compressing audio data. They are general algorithms that work reasonably well for most applications. Conversely, *FLAC* has been design specifically for the lossless compression of audio data.

Figure 12 depicts how the FLAC codec processes audio.

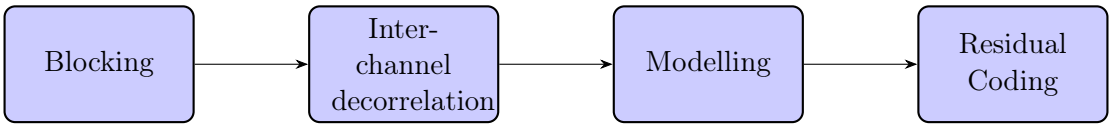

Figure 12: FLAC Overview

Blocking splits input signal into 4096-sample blocks (by default)

Inter-channel decorrelation for each block, select the most compact representation out of 4 possible modes

Modelling try to model the signal using less data than the original (constant, verbatim, fixed linear predictor, FIR linear predictor)

Residual coding residuals are smaller than original sample values so can be encoded with less bits. Rice Coding is used to compress residuals

Rice encoding works like this[1]:

1. Let S be the input sample
2. Let K be the number of bits
3. Let $M = 2^K$
4. Let $R_1 = S \,\&\, (M - 1)$
5. Let $R_2 = S \gg K$ in unary
6. Return the concatenation $R_2 \parallel R_1$

Rice decoding works like this[1]:

1. Let S be the encoded value
2. Let K be the bit depth
3. Let $M = 2^K$
4. Let Q be the number of 1s before the first zero in S in binary
5. Let R be the next K bits of S
6. Return $Q \times M + R$

[1] `http://michael.dipperstein.com/rice/index.html`