Machine Learning for Music

Introduction

History of “algorithmic” music generation

(mostly from “a brief history of algorithmic composition”:https://ccrma.stanford.edu/~blackrse/algorithm.html, which in turn is based on

“techniques for algorithmic composition of music”: http://alum.hampshire.edu/~adaF92/algocomp/algocomp95.html, among others)

History of “algorithmic” music generation

“Canonic” composition (~15th century)

“The prevailing method was to write out a single voice part and to give instructions to the singers to derive the additional voices from it. The instruction or rule by which these further parts were derived was called a canon, which means 'rule' or 'law.' For example, the second voice might be instructed to sing the same melody starting a certain number of beats or measures after the original; the second voice might be an inversion of the first or it might be a retrograde [etc.]" (Grout, 1996)

History of “algorithmic” music generation

Mozart’s *Musikalisches Wurfelspiel* (“dice music”, 1792)

A musical game which *"involved assembling a number of small musical fragments, and combining them by chance, piecing together a new piece from randomly chosen parts"* (Alpern, 1995)

(Mozart wasn’t the first to do this; just the best-known instance)

History of “algorithmic” music generation

Mozart’s *Musikalisches Wurfelspiel* (“dice music”, 1792)

- Roll two dice and add them (to get a number from 2-12)
- For each of 16 bars, select the corresponding entry

https://libraries.mit.edu/news/mozart-rolls/22909/

History of “algorithmic” music generation

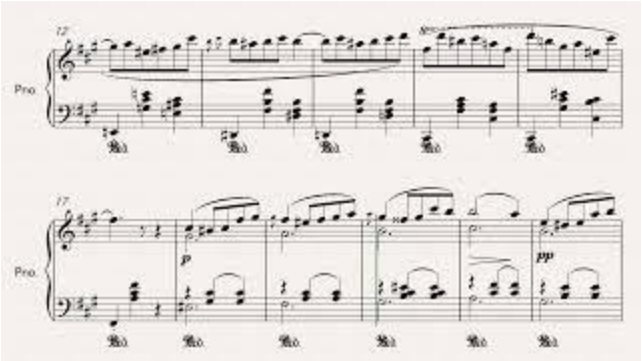
Mozart’s *Musikalisches Wurfelspiel* (“dice music”, 1792)

Try it out!

- Generate sheet music: https://dice.humdrum.org/
- Rendered example: https://onlinekyne.substack.com/p/how-mozart-wrote-759-trillion-songs

History of “algorithmic” music generation

(a more recent AI generated waltz...)



“NotaGen: Advancing Musicality in Symbolic Music Generation with Large Language Model Training Paradigms”
https://electricalexis.github.io/notagen-demo/
https://www.youtube.com/watch?v=mNpegaVDloU

History of “algorithmic” music generation

A more modern example: **Karlheinz Stockhausen's *Klaveirstucke XI* (~1957):**

“Klavierstück XI consists of 19 fragments spread over a single, large page. The performer may begin with any fragment, and continue to any other, proceeding through the labyrinth until a fragment has been reached for the third time, when the performance ends.” (wikipedia)

History of “algorithmic” music generation

John Cage (20th century)

Reunion: performed by playing chess on a photo-receptor equipped chessboard: *"The players' moves trigger sounds, and thus the piece is different each time it is performed"* (Alpern, 1995)

Atlas Eclipticalis: composed by laying score paper on top of astronomical charts and placing notes simply where the stars occurred, delegating the compositional process to indeterminacy (Schwartz, 1993)

(lots of other examples from other composers, these are just the most prominent)

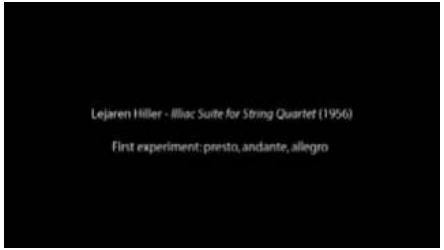
History of “algorithmic” music generation

With computers (though still not with “ML”):

Lejaren Hiller and Leonard Isaacson, *Illiac Suite* (1957):

“The score of the piece was composed by the computer and then transposed into traditional musical notation for performance by a string quartet. What Hiller and Isaacson had done in the Illiac Suite was to

(a) generate certain "raw materials" with the computer
(b) modify these musical materials according to various functions, and then
(c) select the best results from these modifications according to various rules” (Alpern, 1995)



(https://www.youtube.com/watch?v=n0njBFLQSk8, performed by humans)

History of “algorithmic” music generation

Some more...

- Lannis Xenakis – Atrées (1962) and Morsima-Amorsima (1962): *“The program would deduce a score from a list of note densities and probabilistic weights supplied by the programmer, leaving specific decisions to a random number generator"*
- William Shottstaedt – automatic species counterpoint: *“Writes music based on rules from Johann Joseph Fux' [...] instruction book from the early 18th-century [...] The program is built around almost 75 rules, such as 'Parallel fifths are not allowed' and 'Avoid tritones near the cadence in lydian mode.' Schottstaedt assigned a series of 'penalties' for breaking the rules [...] based on the fact that Fux indicated that there were some rules that could never be broken, but others did not have to be adhered to as vehemently. As penalties accumulate, the program abandons its current branch of rules and backtracks to find a new solution"* (Grout, 1996; Burns, 1997)

History of “algorithmic” music generation

We can already characterize the methods to some extent:

Stochastic methods rely on randomness in the composition process, e.g. by learning distributions over notes or randomly combining musical elements

Rule-based methods attempt to discover an underlying “grammar” in music that can be followed to generate music systematically

(of course, methods can be in between these two, e.g. language models combine stochastic processes with “learned” grammars)

History of “algorithmic” music generation

Stochastic: e.g. Markov Chains (e.g. *Illiac Suite*, 1950s):

- **How it works**
  - Analyze existing music to calculate transition probabilities between notes
  - Generate new sequences based on these probabilities
  - Use nth-order chains to consider longer musical contexts
- **Example: First-Order Markov Chain**
  - Note C → D (0.3) → E (0.5) → G (0.2)
  - Note E → C (0.4) → G (0.6)
  - Sample sequence generation:
    C → E → G → E → C
  - **(Module 3!)**

Numbers in parentheses represent transition probabilities based on analysis of existing musical pieces

History of “algorithmic” music generation

Rule-Based Systems (e.g. automatic species counterpoint):

- Core Concepts:
  - Encodes musical theory rules into algorithmic form
  - Uses hierarchical rule sets for different musical aspects:
    - Harmony rules (no parallel 5ths, etc.)
    - Melody construction
    - Rhythm patterns
  - Implements constraints from music theory

History of “algorithmic” music generation

Rule-Based Systems

- Examples
  - Rule 1: Chord Progression
    - IF current_chord = C major,
      - THEN next_chord = [F major, G major, A minor]
  - Rule 2: Voice Leading
    - IF bass_note > middle_C + octave,
      - THEN move_bass_down(octave)
  - Rule 3: Cadence
    - IF phrase_ending = true
      - THEN use_progression([V, I])

(we mostly won’t discuss these in this class though…)

History of “algorithmic” music generation

Rule-Based Systems

- EMI (Experiments in Musical Intelligence) by David Cope (1983)

History of “algorithmic” music generation

Genetic Algorithms in Early AI Music Generation

- Inspired by the process of natural selection (e.g., mutation)
- Represents musical elements as genes
- Genetic operations:
  - Crossover: Combining parts of two melodies
  - Mutation: Random changes in notes/rhythm
  - Selection: Choosing best-fitting melodies
- Example operations
  - Initial Population: Melody1: [C4,E4,G4,C5], Melody2: [G4,F4,E4,D4]
  - Child After Crossover: [C4,E4,E4,D4]
  - Mutant After Mutation: [C4,E4,F4,D4]
- NEUROGEN (Gibson & Byrne, 1991) was one of the earliest approaches (but I couldn’t find a video?)

So what?

Mostly just want to give you a sense that this is not a new topic! (though arguably it's only just beginning to transition from "not working" to "working")

Most of the above methods aren't particularly relevant, except perhaps Markov Chains (Module 3), which begin to suggest approaches based on token prediction and language modeling

So what?

Worth thinking about the relationship to language!

“Early” work mostly consistent of trying to codify rules or underlying “grammar” of language/music; modern techniques rely on large datasets and generally eschew hard-coded rules

Data structures for music and data ingestion

Module 1.1: ML terminology

Introduction to Machine Learning

What is Machine Learning?

Machine Learning is teaching computers to learn from data and make predictions or decisions, without being explicitly programmed for each task

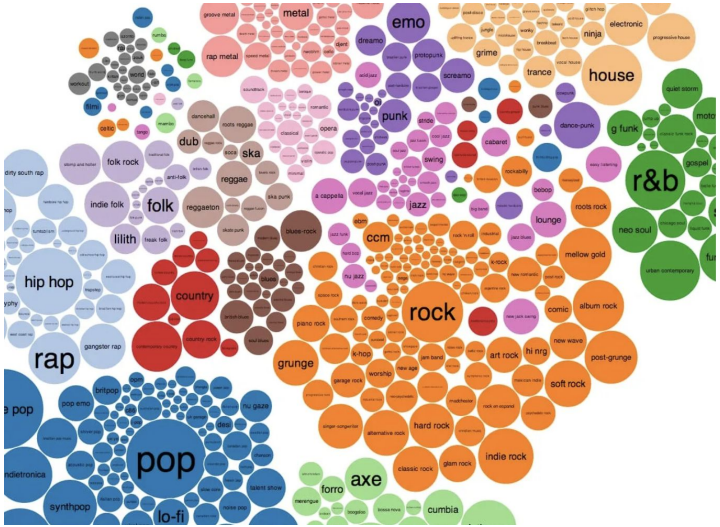
Real-World Musical Applications:

- Music Recommendation (Spotify, Apple Music): Create personalized playlists by learning from your listening history
- Shazam: Identifies songs by learning patterns in audio signals
- Suno/Udio: Automatically generates music audio by learning from existing music

Supervised Learning

Like a music teacher providing correct answers

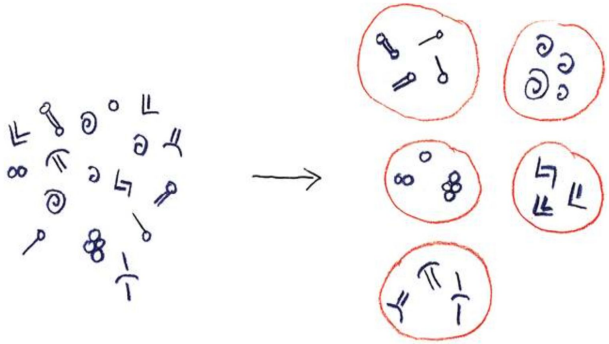
- System learns from **labeled data** (input → correct output pairs)
- Musical Examples:
  - **Genre Classification:** Given a song, predict if it's "Jazz" or "Classical"
  - **Instrument Recognition:** Identify which instrument is playing
  - **Chord Prediction:** Predict the next chord in a progression



Unsupervised Learning

Like discovering patterns in music without a teacher

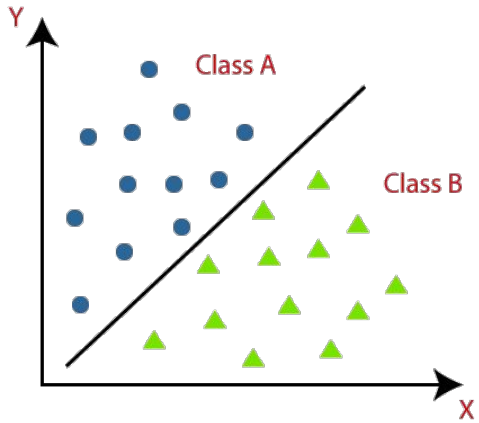
- System finds patterns in **unlabeled data** on its own
- Musical Examples:
  - **Music Clustering:** Group similar songs without predefined genres
  - **Pattern Discovery:** Find recurring melodic patterns in Bach chorales
  - **Listener Segmentation:** Group users with similar music taste



Andrew Glassner, “Deep Learning A Visual Approach”

Classification

- **What is Classification?**: Teaching a computer to sort things into *categories*
- **Types of Classification:**
  - Binary Classification
    - Two possible categories
    - Example: Is this a vocal or instrumental track?
  - Multi-class Classification
    - Multiple possible categories
    - Example: Is this Jazz/Rock/Classical/Hip-Hop?
- **Applications**:
  - Mood Classification for Playlists (Happy? Sad?)
  - Instrument Recognition in Mixed Audio (Guitar? Piano?)
  - Music Recommendation Systems



https://www.javatpoint.com/classification-algorithm-in-machine-learning

Regression

- **What is Regression?** Teaching a computer to predict *numerical values*
- **Real-World Applications:**
  - Audio Effects Parameter Estimation
  - Predicting song popularity (e.g. recommender systems)
  - ?

https://datahacker.rs/002-machine-learning-linear-regression-model/

Regression and classification in a couple of slides…

We’ll discuss ML approaches (and MIR more broadly) in Module 2; but for the first HW we’ll still use these methods as “black boxes;” so to give you a sense…

Regression and classification in a couple of slides…

We’ll discuss ML approaches (and MIR more broadly) in Module 2; but for the first HW we’ll still use these methods as “black boxes;” so to give you a sense…

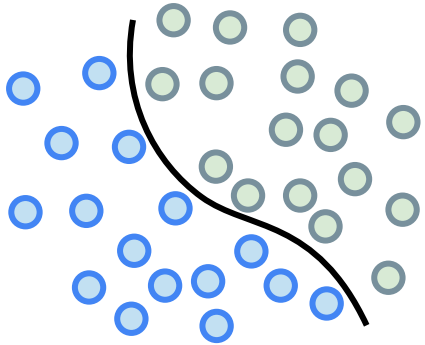
Discriminative versus *Generative* AI

Discriminative AI is concerned with estimating the probability of a *label* given some observed data, i.e., *p(y|x)*

Generative AI is concerned with understanding the distribution of the data itself, i.e., *p(x)*, such that *new samples* can be drawn from that distribution

Discriminative versus *Generative* AI

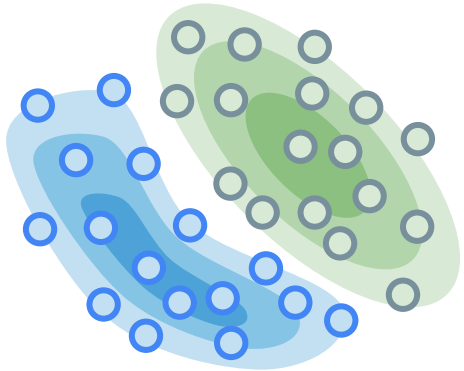
Discriminative



Discriminative models learn the decision boundary

$P(y|x)$

Generative

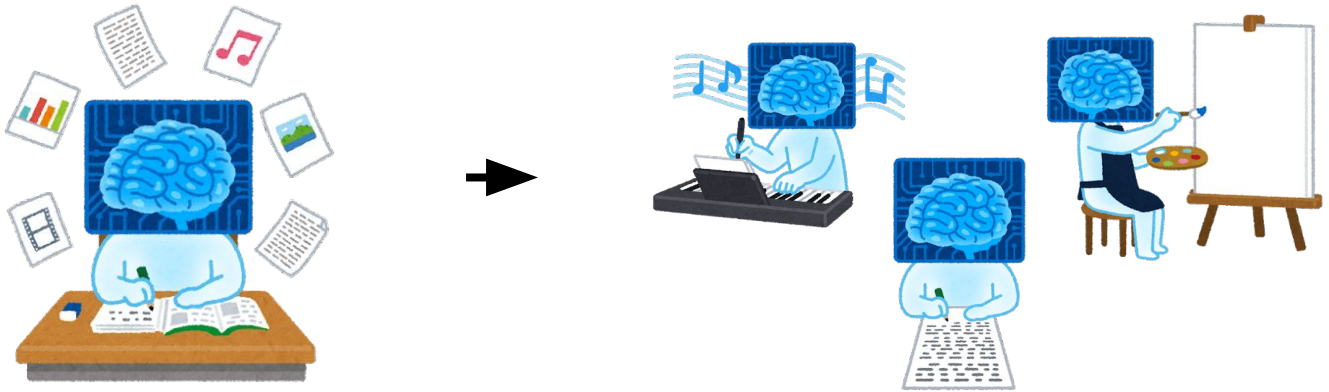


Generative models learn the underlying distribution

$P(x)$ or $P(x|y)$

Generative AI

Generative AI is AI capable of generating text, images, music or other media



Music and AI: https://hermandong.com/teaching/pat498_598_winter2025/

Discriminative versus *Generative* AI

In the context of Music for AI, *discriminative tasks* might include:

- What is the genre of a piece of music?
- What song is most relevant to a user's interests (recommender systems)?
- (everything we saw when discussing e.g. classification above)

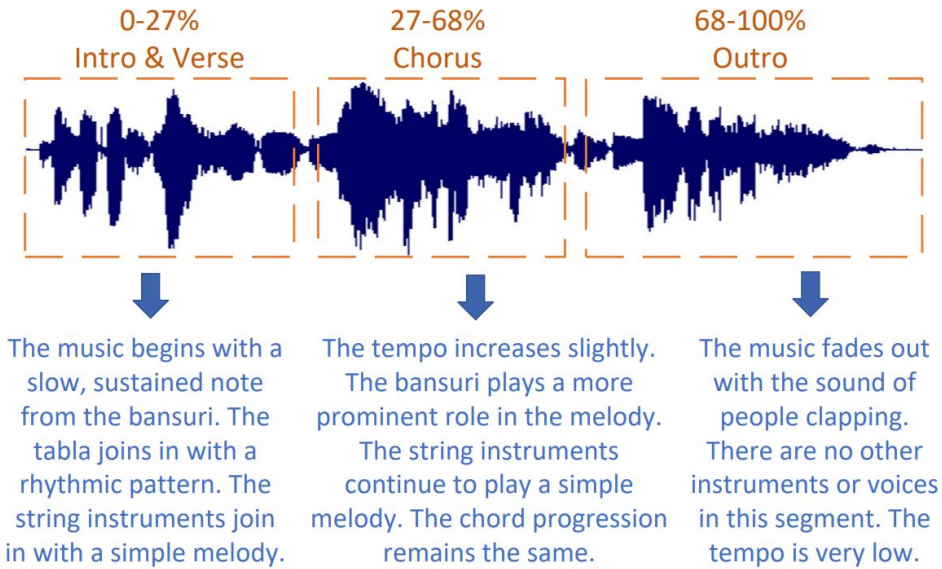
Generative AI for music

Generative AI samples from the underlying distribution (x), so can “generate” new samples

NVIDIA Fugatto (2024)

Generative AI for music

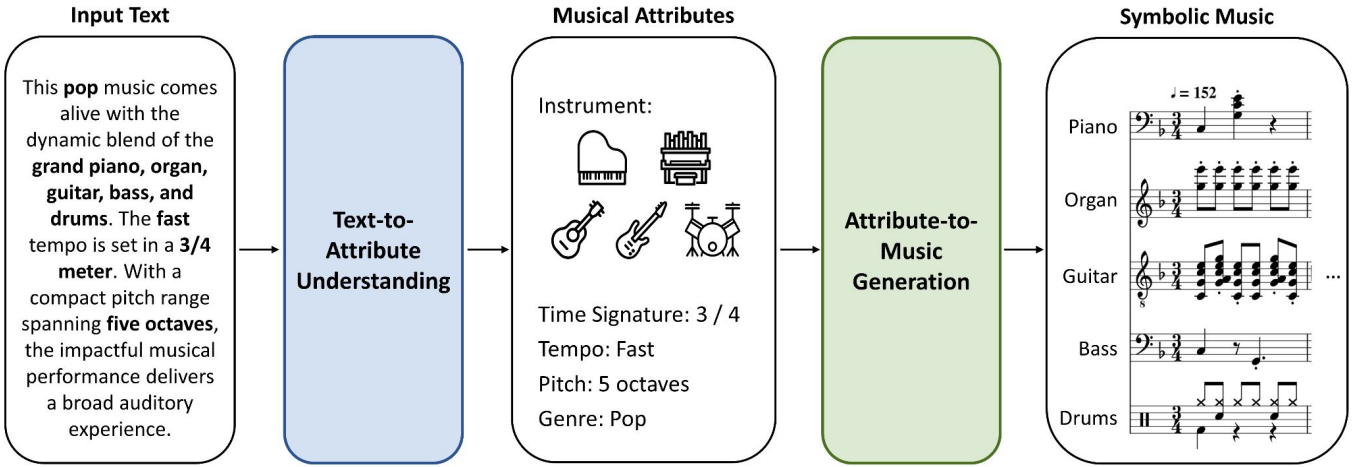
Generating textual description from music (Captioning)



From: “Towards Fine-grained Music Understanding through Temporally-enhanced Generative Augmentation”

Generative AI for music

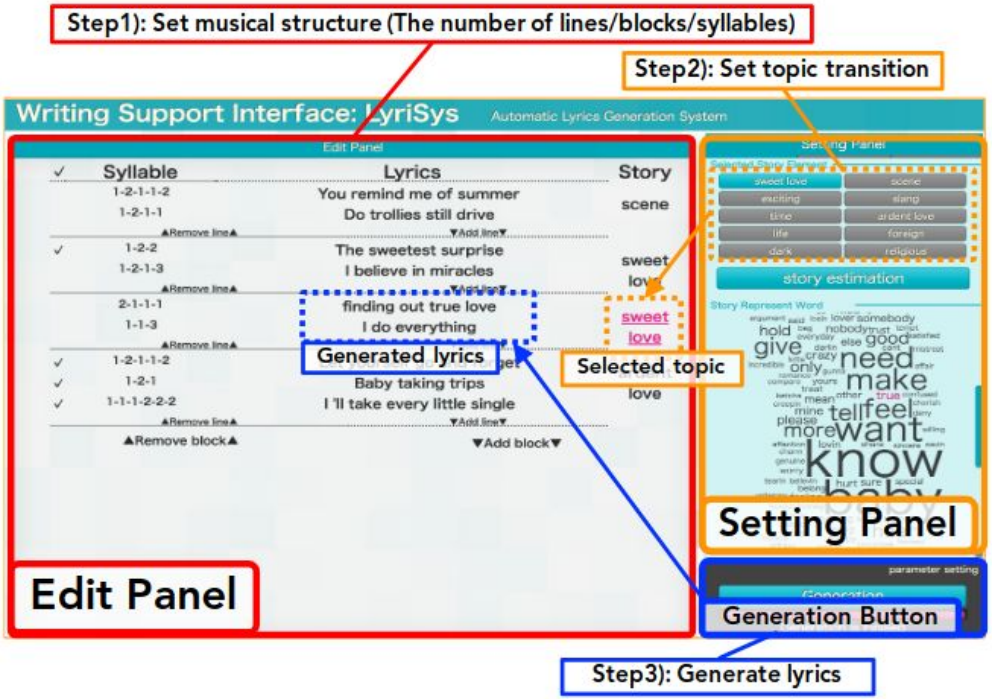
Generating symbolic music



"MuseCoco: Generating Symbolic Music from Text"(2023)

Generative AI for music

Generating lyrics



"Lyrisys: An interactive support system for writing lyrics based on topic transition" (2017)

Take-homes

I (mostly) won't teach introductory ML in this course:

- I am sick of teaching it
- It takes too much time given its lower relevance to this course

We'll mostly treat ML "methods" as black boxes for various tasks; you should try to understand the above to the extent that you can competently invoke library functions, and so that you know what type of method to use when

The provided notebooks will help you! (There'll also be some more in **Module 2**)

Data structures for music and data ingestion

1.2 Primer on audio and audio terminology

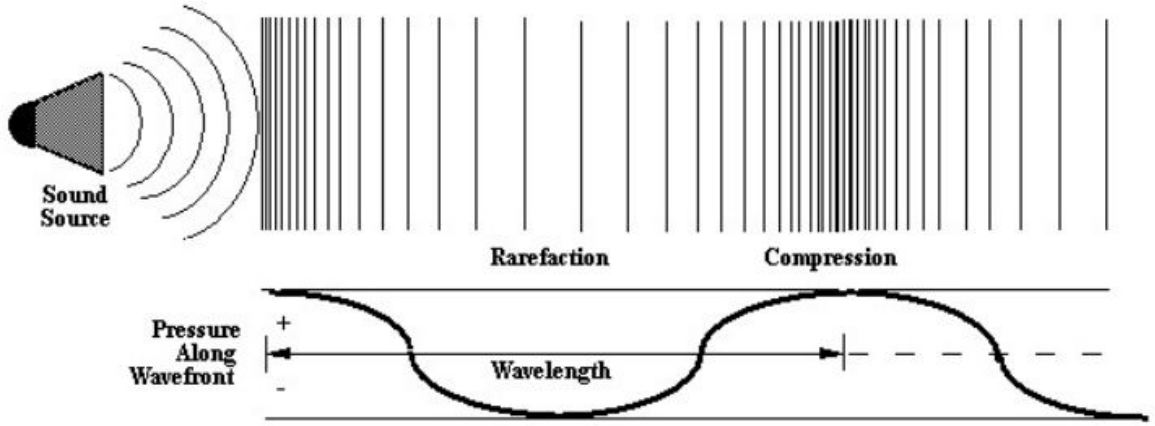
Fundamentals of sound and digital audio

(based on “Intro to Computer Music”: https://www.cs.cmu.edu/~15322/)

- What is sound?
- How do we *represent* sound on a computer as digital audio?
- How do we *perceive* sound and music?

What is sound?

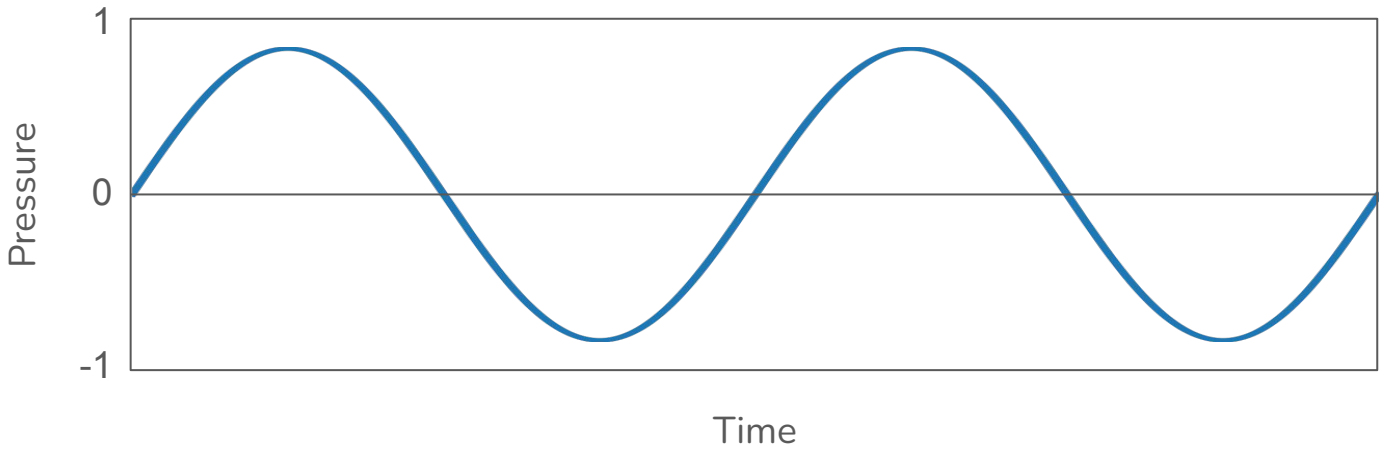
- Sound is a variation in pressure
- Pressure variations travel through air as *waves*
- In natural sounds, these variations tend to oscillate in cyclical patterns



Rarefaction and Compression of a Sound Wave

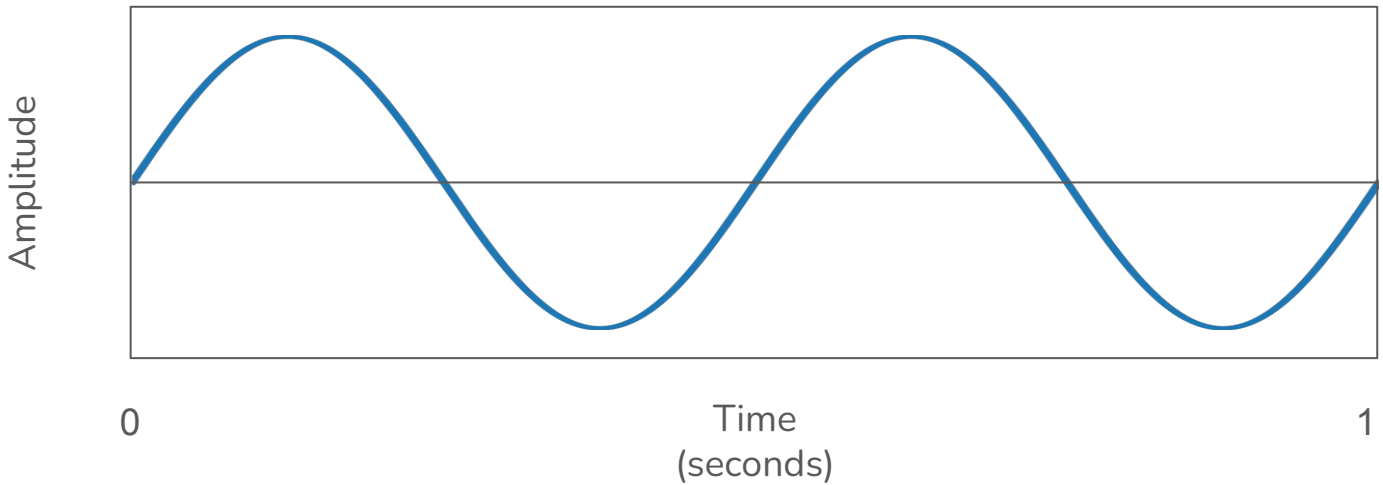
What is sound?

A waveform is a measurement of pressure over time:



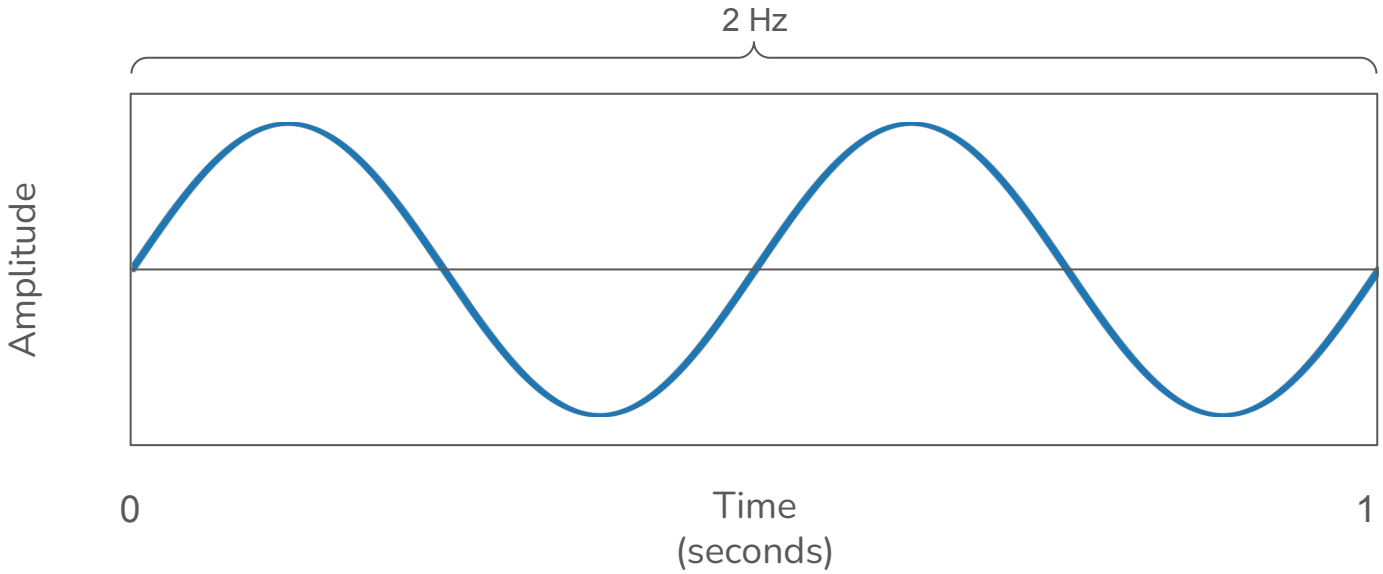
What is sound?

Once recorded with a microphone, we often refer to the measured sound pressure as the *amplitude*



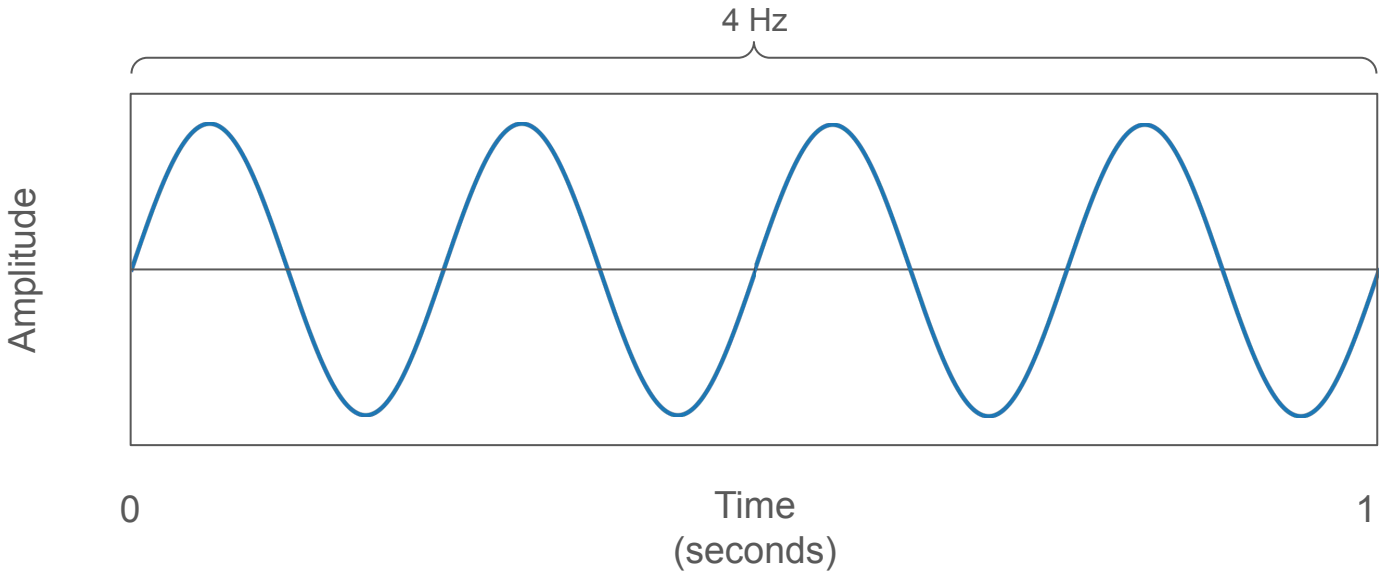
What is sound?

Frequency is the number of cycles per second (measured in Hertz, Hz)



What is sound?

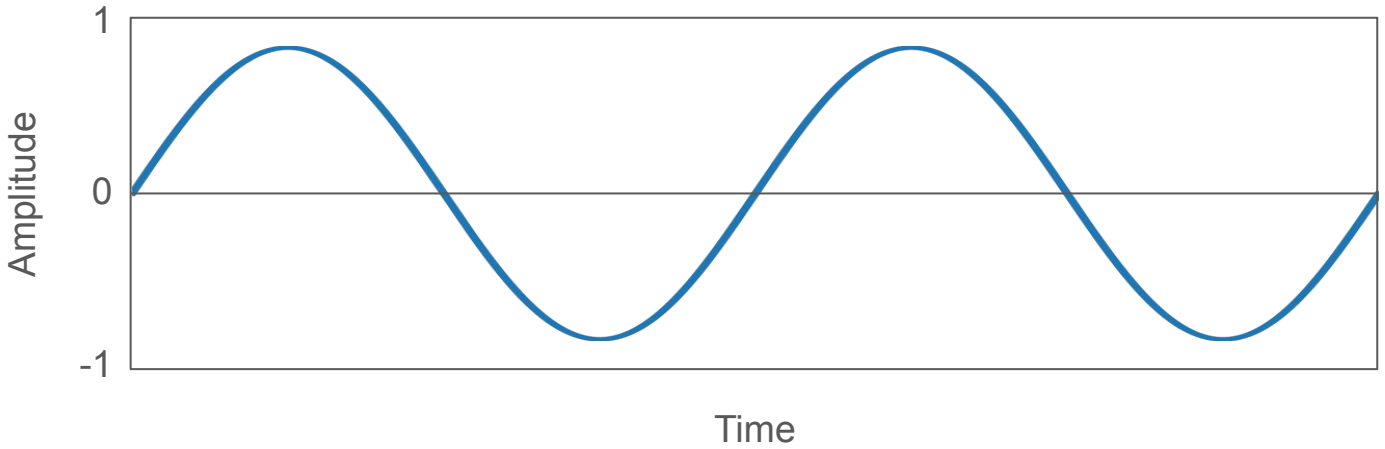
Frequency is the number of cycles per second (measured in Hertz, Hz)



Introduction to *digital audio*

Audio is a continuous or *analogue* measurement of fluctuating air pressure

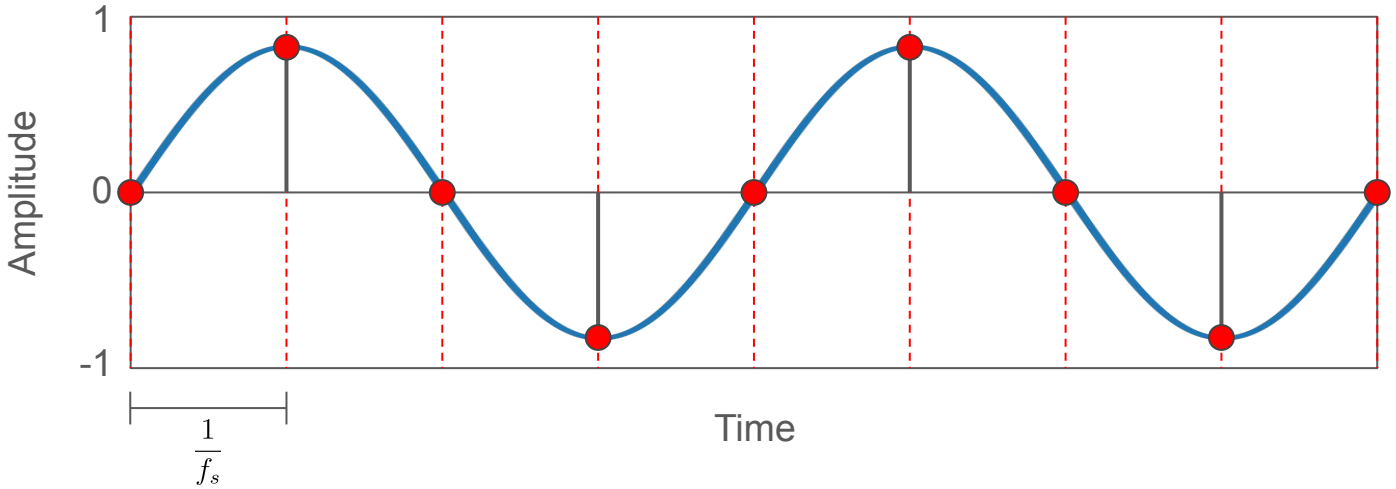
But analogue signals cannot be natively stored on digital media



Analogue to digital conversion: *sampling*

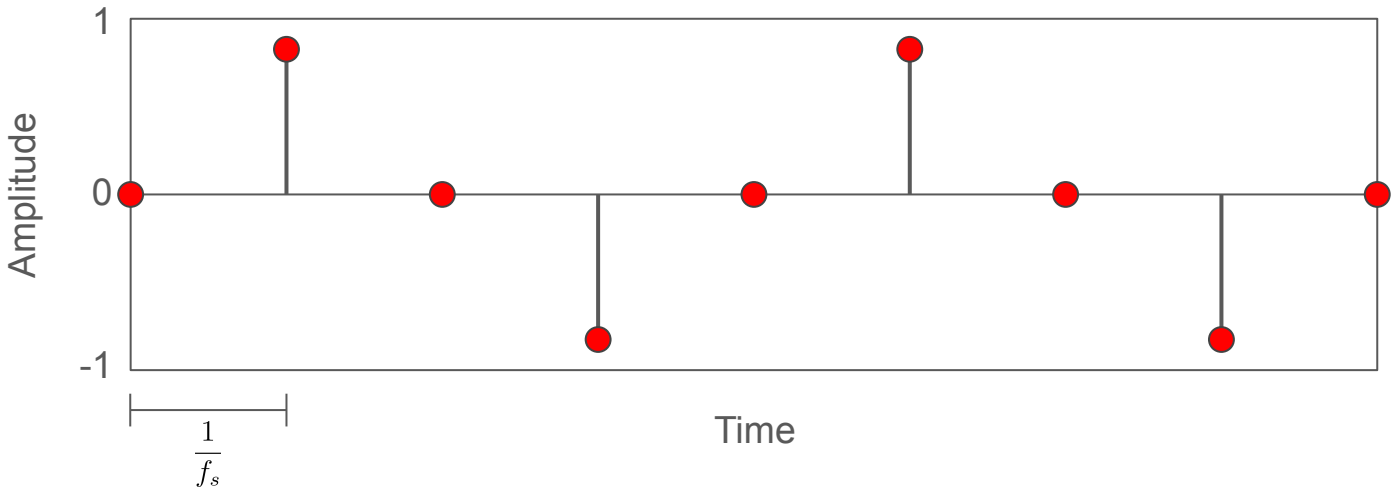
Digital audio involves *sampling* the analogue signal at uniform intervals

The number of *samples* per second is the *sample rate* (f_s)



Analogue to digital conversion: *sampling*

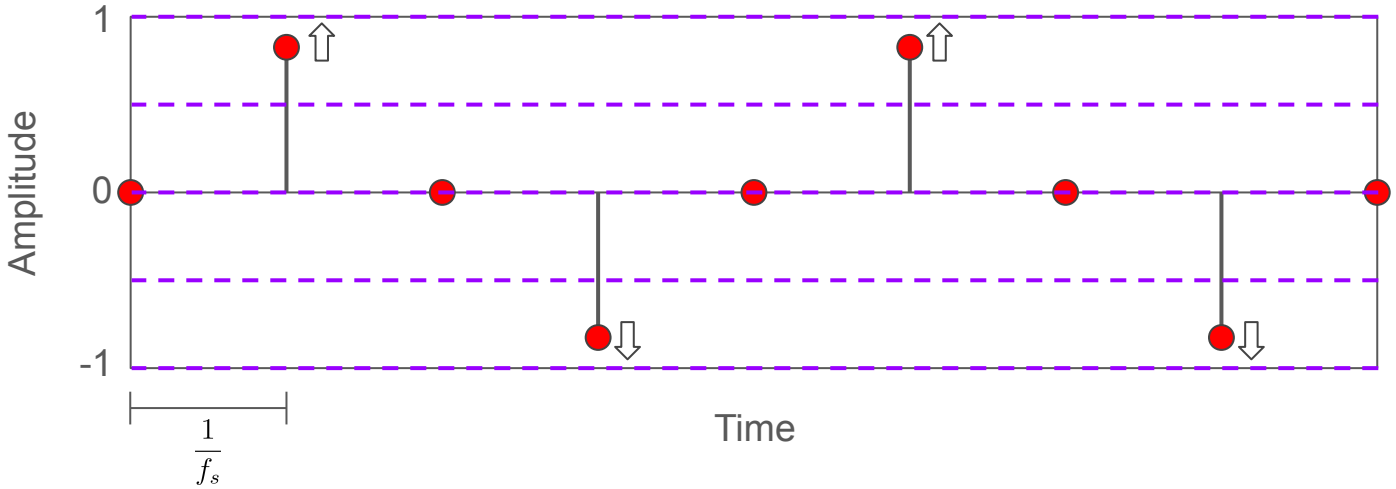
But amplitude values are *also* continuous? How do we store them?



Analogue to digital conversion: *quantizing*

Pick some set of amplitude values, usually linearly spaced in amplitude

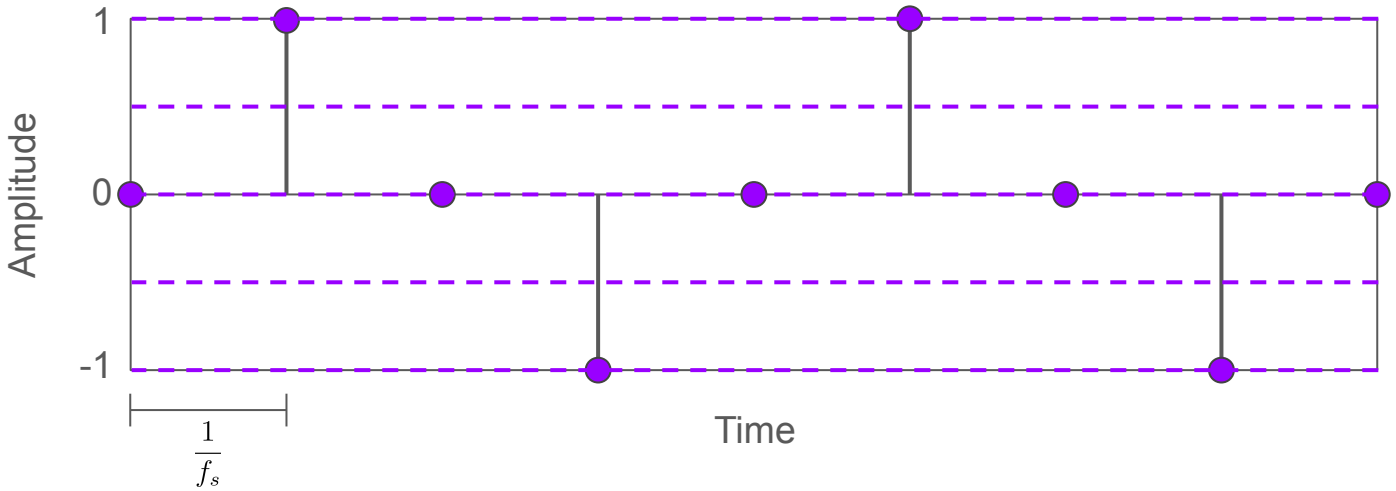
Round each sample to the nearest amplitude in the set



Analogue to digital conversion: *quantizing*

Pick some set of amplitude values, usually linearly spaced in amplitude

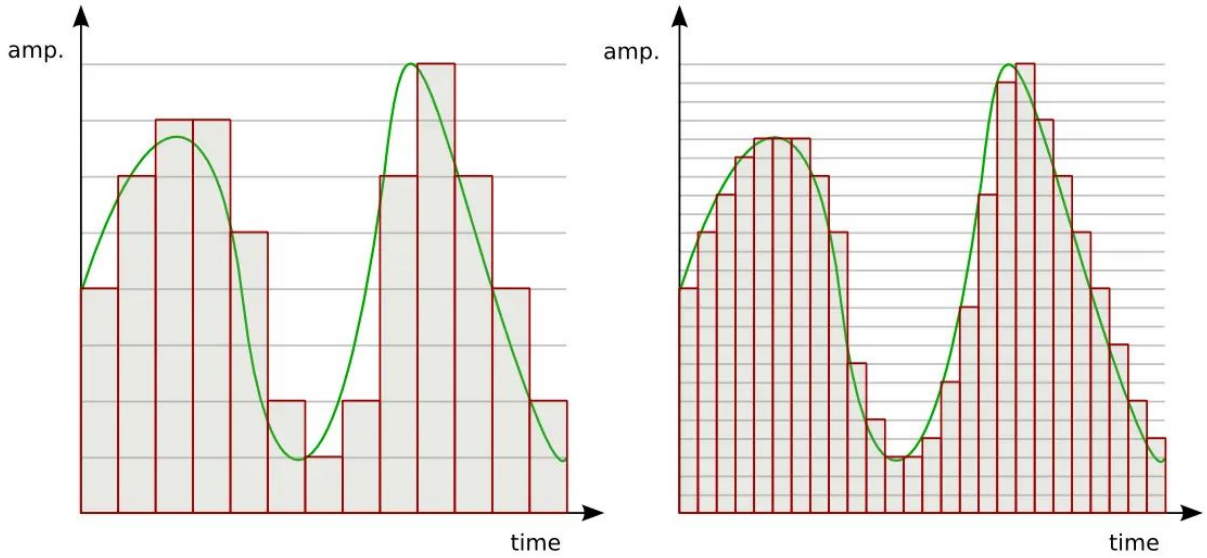
Round each sample to the nearest amplitude in the set



Bit Depth

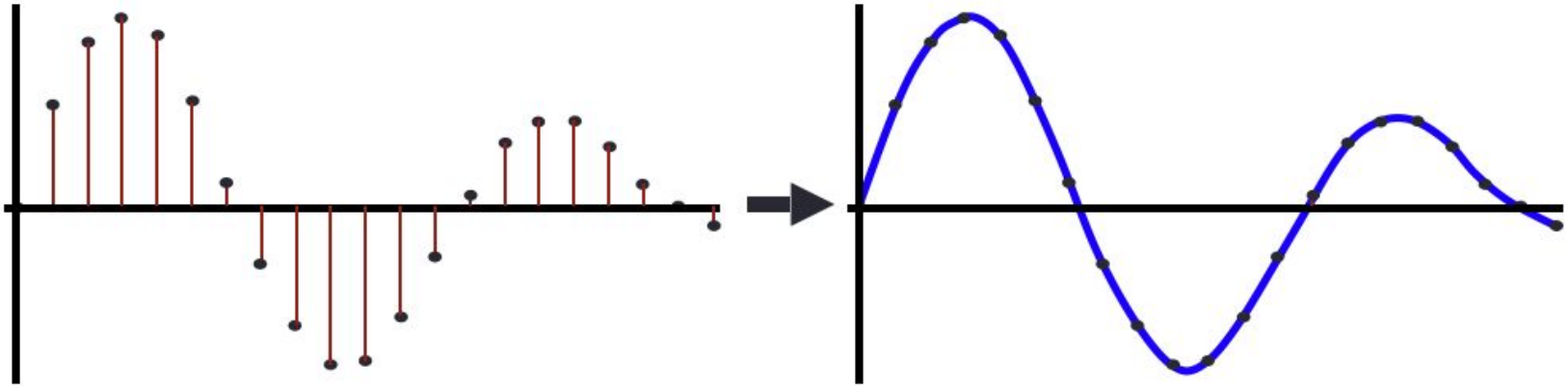
- The number of possible values to store the amplitude (the greater the number of possible values is, the more precise the sampled amplitude will be
- With n bits, 2^n possible values that can be represented
  - For CD quality audio, n = 16
  - Each sample can have 2^{16} = 65,536 possible values.
  - The highest possible amplitude is 2^{15}-1 = 32,767 (since audio signals are either positive, zero, or negative.)

Bit depth and frequency



Digital to analogue conversion

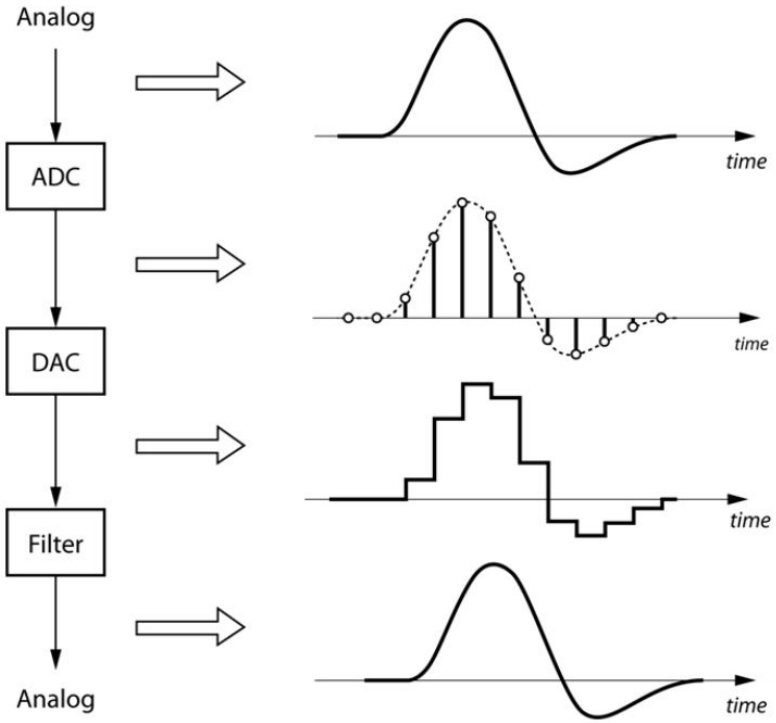
- When we need to reproduce the digital audio, we *filter* the samples to produce a smooth signal
- Filtering is almost always done for you by the “DAC” (digital analogue converter) on your hardware
- (not particularly relevant to this class!)



How do we actually store digital audio as a file?

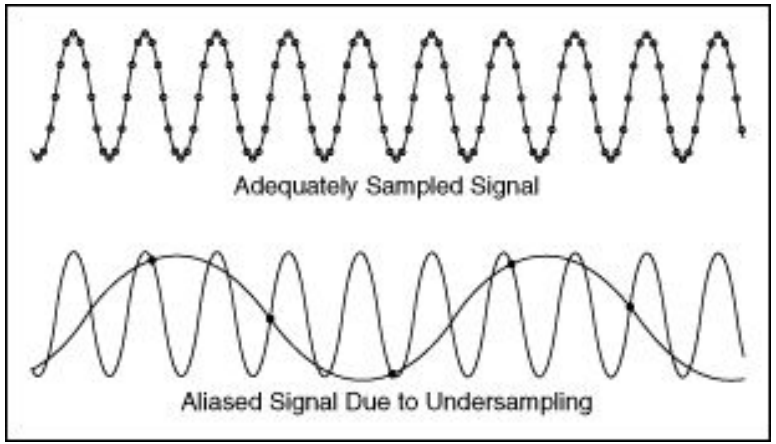
- Audio formats like WAV contain three high-level pieces of information:
  - *Header*: some metadata about the audio and file format
  - *Samples*: a list of integer amplitude values [0, 2000, -4000, ...]
  - *Sample rate*: at what rate should these samples be *played back*
- File size in bits = sample rate * *channels* * *seconds* * *bits* per sample
  - E.g., 10s of 44.1kHz mono audio at 16 bits: 44100 * 1 * 10 * 16
- Other file formats like MP3 / OGG / FLAC / etc are just fancy ways of *compressing* the samples for easier storage and transmission

Summary of digital-to-analogue conversion



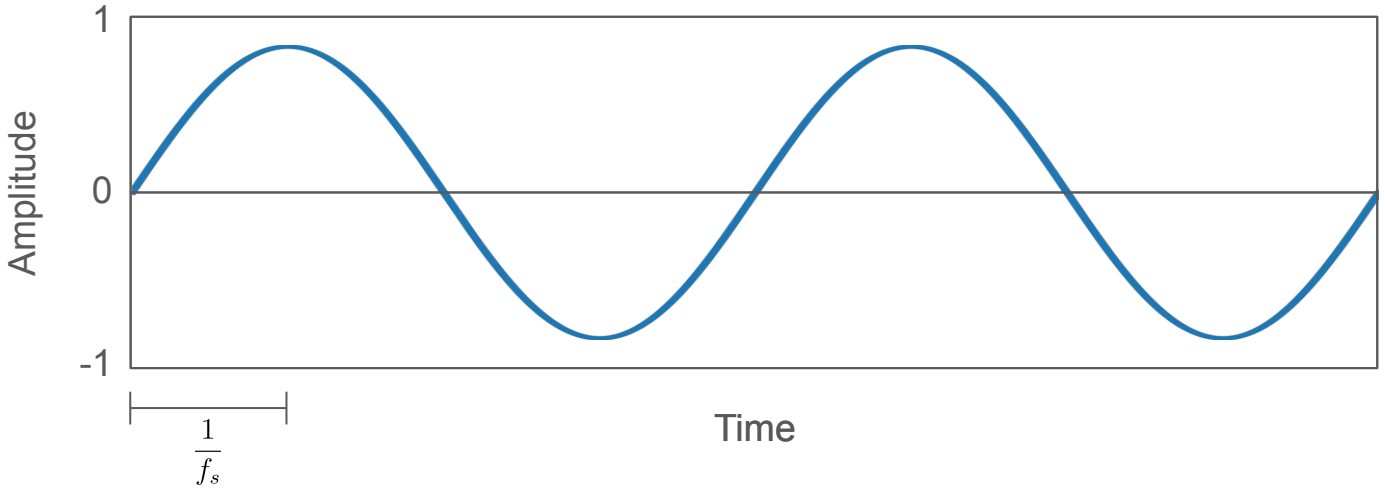
Aliasing: a side effect of sampling

- *Aliasing*: infinitely many continuous waveforms that map to the same samples
- Nyquist-Shannon sampling theorem (1915): aliasing can be prevented if our sampling rate is at least double the highest frequency in the waveform
- (mostly we'll avoid going too deep into "audio stuff" though...)



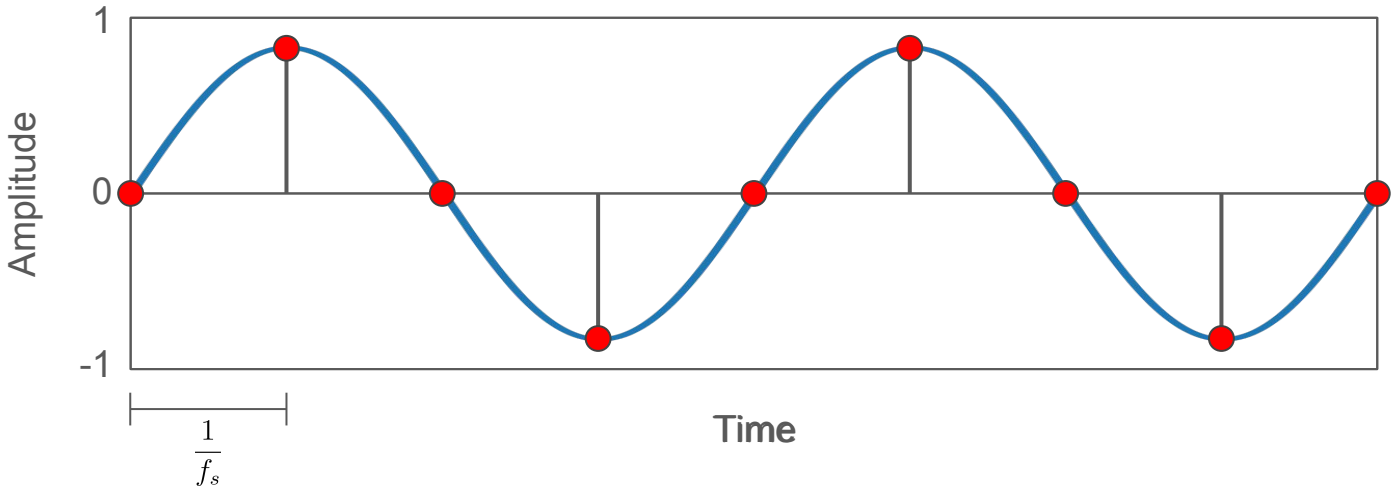
How do we *synthesize* this wave on a computer?

- Now that we know how to sample natural sound as digital audio, how might we synthesize new sounds in code?
- Building block: all sounds can be represented as a sum of sine waves
- $a(t) = \sin(2\pi ft + \varphi)$



How do we *synthesize* this wave on a computer?

- $a(t) = \sin(2\pi f t + \varphi)$
- To synthesize this at sampling rate f_s, we only need to calculate the value of this function at points $t = i / f_s$, where $i \in Z^+$



How do we *synthesize* this wave on a computer?

(see code later in the module; let's study musical notes and pitches first)

Take-homes

Just wanted to cover the very very basics of audio

Compared to the course on which the above slides were based (https://www.cs.cmu.edu/~15322/) this course has very little focus on *audio* processing as such, and much more of a focus on *music*

So lots I don't cover, e.g. spectral processing, vocal synthesis, physical models, compression, etc.

Happy to take feedback (at the end of the quarter) if there are any topics you'd like to see more of

Data structures for music and data ingestion

1.3: Primer on music and musical terminology

Music theory

If you truly don’t care, feel free to let your eyes glaze over for this part, and I may not bother to go through it all anyway – just treat it as something that’s here for a reference

These concepts will certainly come up in various papers and case studies, and while they’re not necessary to pass the homeworks, if you want to thoroughly understand the topic you should try to learn these basics

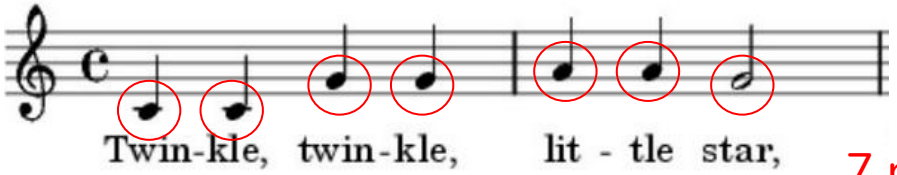
Pitch and Notes

Pitch

- A musical quality that makes a sound "high" or "low"
- Determined by the frequency of sound waves (faster vibration = higher pitch)
- Each distinct pitch in Western music has a specific letter name (A through G)

Musical Notes

- Written symbols that represent specific **pitches and durations** in music



7 musical notes

Scale

A **scale** is a series of notes, ordered by pitch (low to high) within an octave

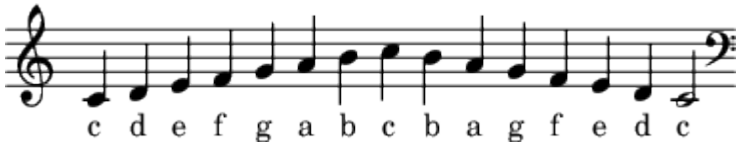
An **octave** is a pair of notes with frequencies *(f, 2f)*

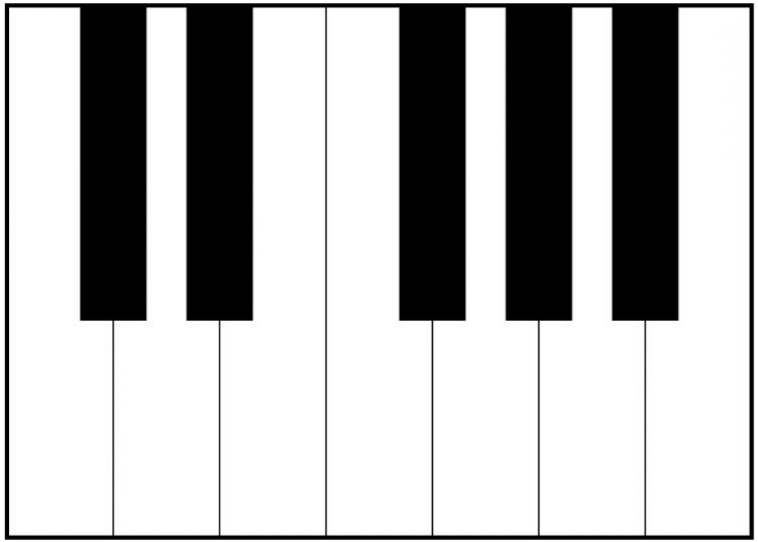
But really, a scale can just be thought of as a set of notes that “go together”

Scale

E.g. C major:

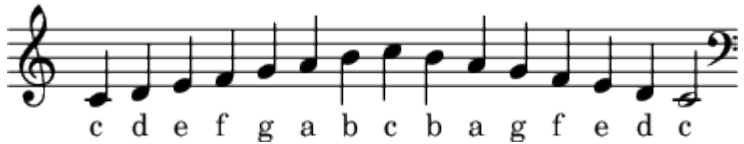
(https://upload.wikimedia.org/score/1/n/1nd2lidxg20ikt80gxhzffyc8o259nz/1nd2lidx.mp3)

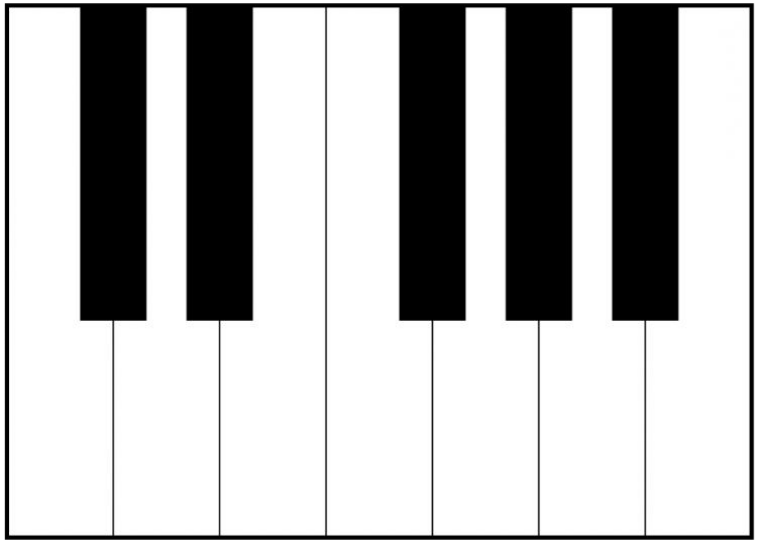




Scale

Why does it sound good?

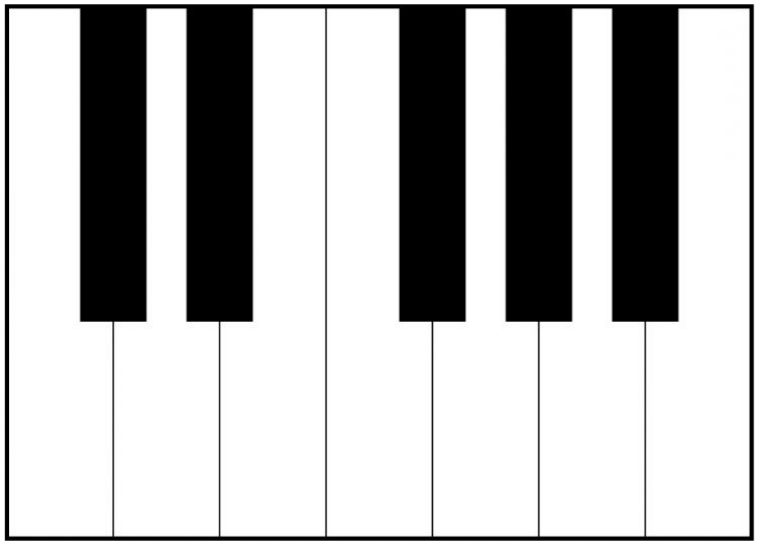




Scale

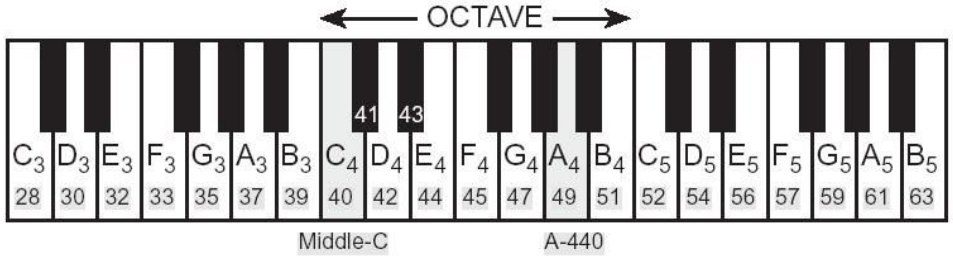
E.g. pentatonic

A “pentatonic” scale is any scale with five notes; e.g. C major pentatonic:



Musical “alphabet”

- Western music uses 12 distinct semitones in each octave:
  - Natural notes: A, B, C, D, E, F, G
  - Notes with accidentals (next slide): A#/Bb, C#/Db, D#/Eb, F#/Gb, G#/Ab
- Octaves are named with numbers (C4 is middle C)
  - Lower numbers indicate lower octaves (C3, C2, C1)
  - Higher numbers indicate higher octaves (C5, C6, C7)
- Each octave represents a doubling of frequency
  - Example: A4 = 440 Hz, A5 = 880 Hz



https://musicnotes101.wordpress.com/2010/04/20/the-musical-alphabet-clefs-the-musical-staff-and-the-keyboard/

Musical “alphabet”

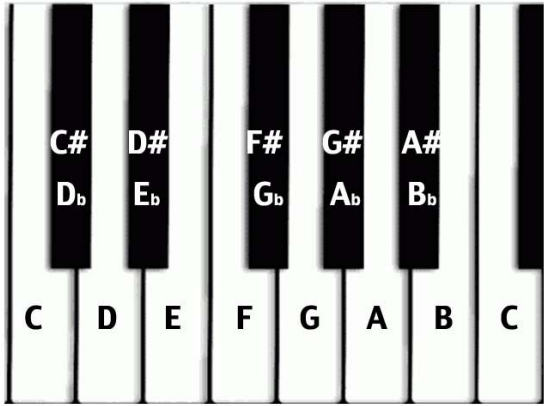
Why does (most Western) music use 12 tones?

Musical “alphabet”

Plenty of other tonal systems (and musical interfaces other than the piano!)

Accidentals

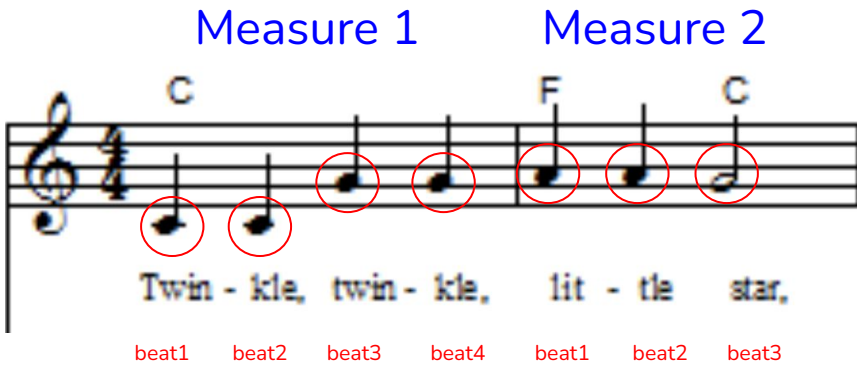
- Symbols that modify the pitch of a note
- Three main types:
  - Sharp (♯): Raises the pitch by one half step
  - Flat (♭): Lowers the pitch by one half step
  - Natural (♮): Cancels previous sharps or flats
- Each pair of natural notes has a black key between them on the piano, EXCEPT:
  - Between B and C
  - Between E and F
- * Examples:
  - C♯ is the black key immediately to the right of C
  - D ♭ is the black key immediately to the left of D
  - C♯ and D ♭ represent the same pitch



Note Values in Simple Time Signature

Simple Time Signatures

- Most common: 4/4 time
  - 4 beats per measure
  - Quarter note gets one beat
- *Other common signatures: 2/4, 3/4
  - First number: beats per measure
  - Second number: note value that gets one beat



Note Values in Simple Time Signature

WHOLE NOTE

- 4 beats in 4/4 time
- Holds for entire measure in 4/4

HALF NOTE

- 2 beats in 4/4 time
- Half the duration of a whole note

QUARTER NOTE

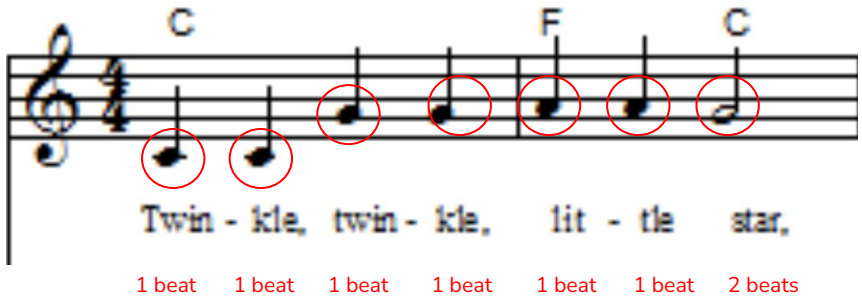
- 1 beat in 4/4 time

EIGHTH NOTE

- 1/2 beat in 4/4 time

https://molliegoddard.com/2021/05/08/note-lengths-whole-half-quarter-eighth-and-their-rests/?v=92a31fc033f7

Note Values in Simple Time Signature



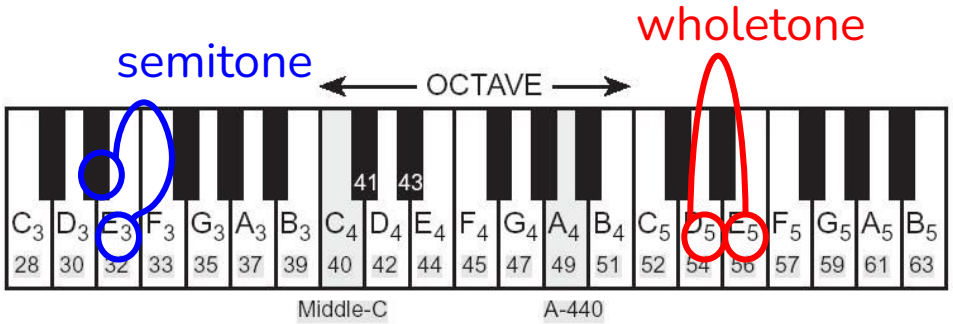
Semitone

Semitone?

- The smallest distance between two notes in Western music
- Examples on a piano:
  - From any key (white or black) to the very next key
  - From B to C (white to white)
  - From E to F (white to white)
  - From F to F# (white to black)
  - From B ♭ to B (black to white)

Whole Tone

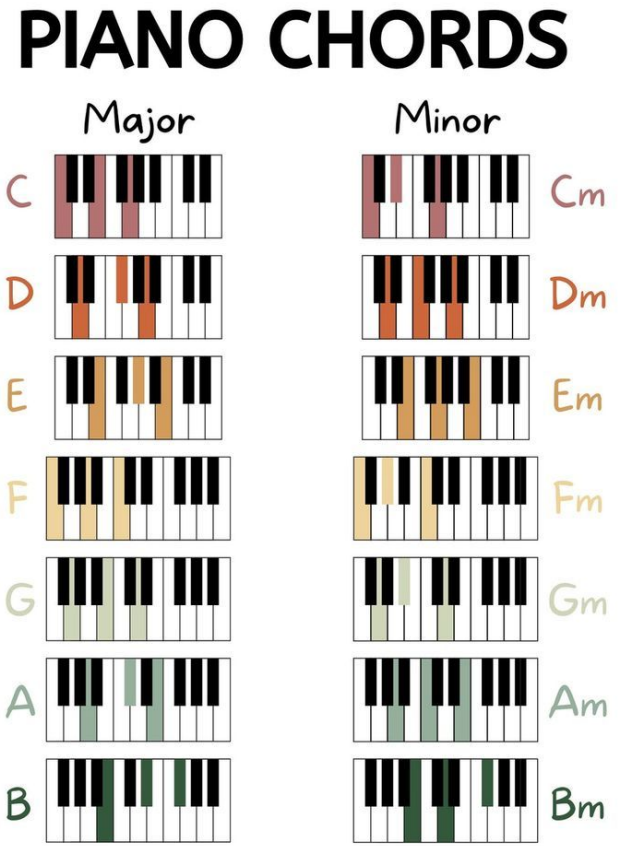
- Equal to two semitones
- Examples:
  - * C to D (skipping C#/D ♭)
  - * F to G (skipping F#/G ♭)



Understanding Chords

- What is a Chord?
  - Three or more notes played simultaneously
  - Built by stacking thirds
  - Most basic form: triad (three notes)
- Triad Structure
  - Root: The base note that gives the chord its name
  - Third: 3rd note up from the root
  - Fifth: 5th note up from the root

https://pin.it/3BpoAvnLV

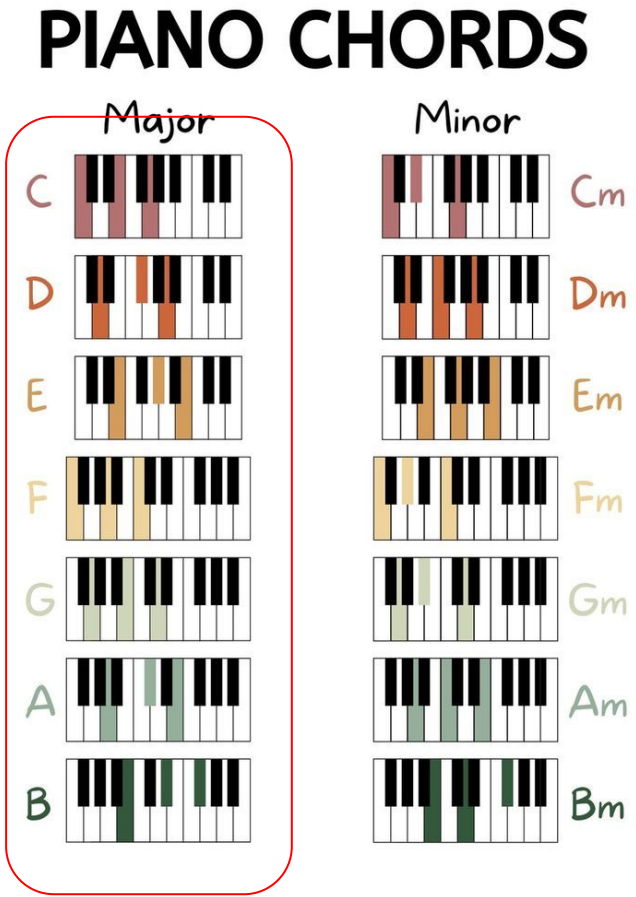


Understanding Chords

Major Chords

- Structure
  - From root to third: 4 semitones (major third)
  - From third to fifth: 3 semitones (minor third)
  - Total from root to fifth: 7 semitones (perfect fifth)
- Examples:
  - C major (C-E-G):
    - C to E: 4 semitones
    - E to G: 3 semitones
  - F major (F-A-C)
  - G major (G-B-D)

https://pin.it/3BpoAvnLV

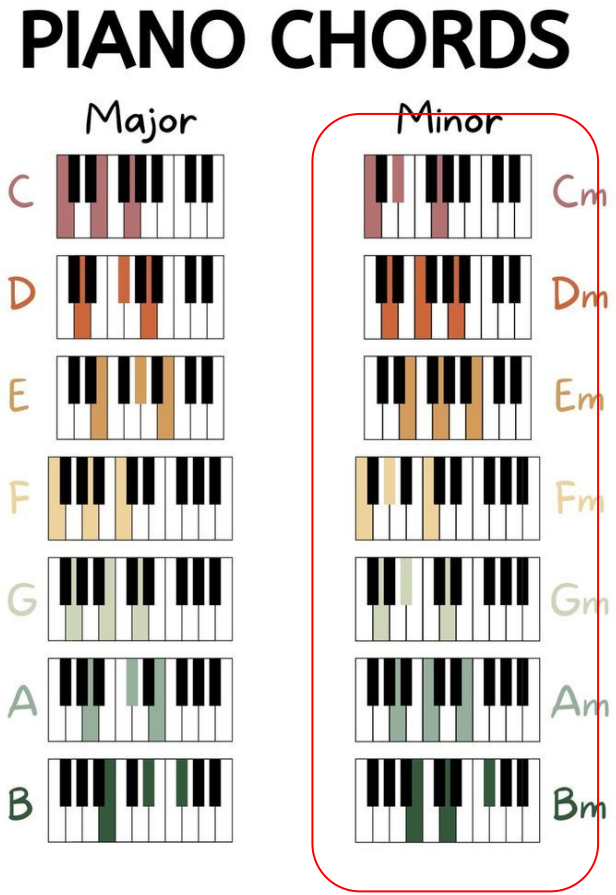


Understanding Chords

Minor Chords

- Structure
  - From root to third: 3 semitones
  - From third to fifth: 4 semitones
  - Total from root to fifth: 7 semitones
- Examples:
  - A minor (A-C-E):
    - A to C: 3 semitones
    - C to E: 4 semitones
  - D minor (D-F-A)
  - E minor (E-G-B)

https://pin.it/3BpoAvnLV



Chord Progression

What is a Chord Progression?

A chord progression is a sequence of musical chords played one after another, forming the harmonic foundation of a piece of music

Common Example: I-V-vi-IV

Key	Chord Progression			
	I	V	vi	IV
C	C	G	Am	F
Db	Db	Ab	Bbm	Gb
D	D	A	Bm	G
Eb	Eb	Bb	Cm	Ab
E	E	B	C#m	A
F	F	C	Dm	Bb
Gb	Gb	Db	Ebm	Cb
G	G	D	Em	C
Ab	Ab	Eb	Fm	Db
A	A	E	F#m	D
Bb	Bb	F	Gm	Eb
B	B	F#	G#m	E

Song structure

Song Structure

Song structure is the overall arrangement of musical sections that form a complete composition

Musical Sections

Musical sections are distinct parts within a song, each serving a specific purpose in the overall composition. Common sections include:

- Core Sections:
  - Verse (tells the story)
  - Chorus (main message)
  - Bridge (contrast/variety)
  - Pre-chorus (builds tension)
- Support Sections:
  - Intro (opening)
  - Outro (ending)
  - Instrumental break
  - Interlude

Song structure

https://blacktidemusic.com/blog/song-structure-template/

Take-homes

Again: you don't *have* to follow the above to pass the course

Treat it as a reference in case some slides mention these concepts and you need to refresh what's being talked about

Always be willing to ask basic ("dumb") questions about music throughout the quarter! E.g. if I offhandedly mention some musical concept you don't know about, lots of people will share your confusion

That being said, if the only thing you learn in this course is some introductory music theory, it will have been worth it!

Data structures for music and data ingestion

1.4: Generating sound and music (in Python)

Generating sound and music

Mostly based on:
https://www.audiolabs-erlangen.de/resources/MIR/FMP/B/B.html

Which is in general an excellent resource, but also includes Python examples, with audio processing specifically in mind

Code example

(**workbook1.ipynb**, from course webpage)

How to calculate note frequencies

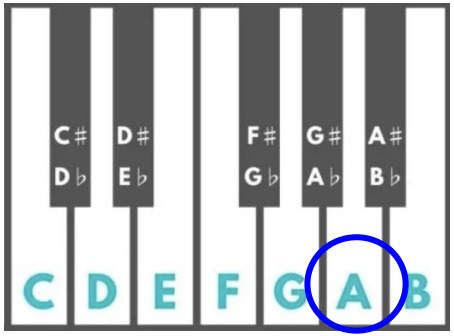
Starting Point: A4 = 440Hz

Basic Formula for any note:

```
frequency = 440 × (2^(1/12))^n
```

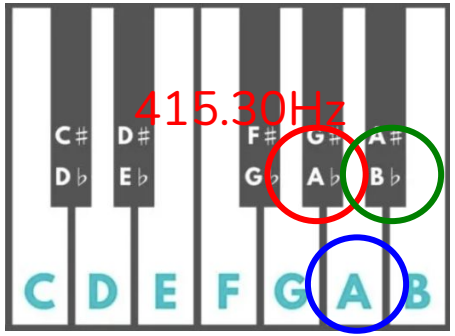
where n = number of semitones from A4

- Negative n: going down from A4
- Positive n: going up from A4



440Hz

How to calculate note frequencies



G#4 (One Semitone Below A4)

```
n = -1
frequency = 440 × (2^(1/12))^(-1)
= 440 × 0.9439
= 415.30 Hz
```

A#4 (One Semitone Above A4)

```
n = 1
frequency = 440 × (2^(1/12))^1
= 440 × 1.0595
= 466.16 Hz
```

Note	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	16.35 Hz	32.70 Hz	65.41 Hz	130.81 Hz	261.63 Hz	523.25 Hz	1046.50 Hz	2093.00 Hz	4186.01 Hz
C#/Db	17.32 Hz	34.65 Hz	69.30 Hz	138.59 Hz	277.18 Hz	554.37 Hz	1108.73 Hz	2217.46 Hz	4434.92 Hz
D	18.35 Hz	36.71 Hz	73.42 Hz	146.83 Hz	293.66 Hz	587.33 Hz	1174.66 Hz	2349.32 Hz	4698.63 Hz
D#/Eb	19.45 Hz	38.89 Hz	77.78 Hz	155.56 Hz	311.13 Hz	622.25 Hz	1244.51 Hz	2489.02 Hz	4978.03 Hz
E	20.60 Hz	41.20 Hz	82.41 Hz	164.81 Hz	329.63 Hz	659.25 Hz	1318.51 Hz	2637.02 Hz	5274.04 Hz
F	21.83 Hz	43.65 Hz	87.31 Hz	174.61 Hz	349.23 Hz	698.46 Hz	1396.91 Hz	2793.83 Hz	5587.65 Hz
F#/Gb	23.12 Hz	46.25 Hz	92.50 Hz	185 Hz	369.99 Hz	739.99 Hz	1479.98 Hz	2959.96 Hz	5919.91 Hz
G	24.50 Hz	49 Hz	98 Hz	196 Hz	392 Hz	783.99 Hz	1567.98 Hz	3135.96 Hz	6271.93 Hz
G#/Ab	25.96 Hz	51.91 Hz	103.83 Hz	207.65 Hz	415.30 Hz	830.61 Hz	1661.22 Hz	3322.44 Hz	6644.88 Hz
A	27.50 Hz	55 Hz	110 Hz	220 Hz	440 Hz	880 Hz	1760 Hz	3520 Hz	7040 Hz
A#/Bb	29.14 Hz	58.27 Hz	116.54 Hz	233.08 Hz	466.16 Hz	932.33 Hz	1864.66 Hz	3729.31 Hz	7458.62 Hz
B	30.87 Hz	61.74 Hz	123.47 Hz	246.94 Hz	493.88 Hz	987.77 Hz	1975.53 Hz	3951.07 Hz	7902.13 Hz

Generating sound and music

First, we'll define the frequencies for each musical note

```
NOTE_FREQUENCIES = {
    'C4': 261.63, 'D4': 293.66, 'E4': 329.63, 'F4': 349.23,
    'G4': 392.00, 'A4': 440.00, 'B4': 493.88, 'C5': 523.25,
    'F3': 174.61, 'G3': 196.00, 'A3': 220.00, 'B3': 246.94
}
```

440Hz defined as A4

The frequency difference between semitones is

a ratio of $2^{1/12}$ (higher notes, higher frequencies).

Note	Octave 4
C	261.63 Hz
C#/Db	277.18 Hz
D	293.66 Hz
D#/Eb	311.13 Hz
E	329.63 Hz
F	349.23 Hz
F#/Gb	369.99 Hz
G	392 Hz
G#/Ab	415.30 Hz
A	440 Hz
A#/Bb	466.16 Hz
B	493.88 Hz

Generating sound and music

Let's create a simple sine wave

```
def create_sine_wave(frequency, duration, sample_rate=44100):
    t = np.linspace(0, duration, int(sample_rate * duration))
        #When duration is 1, t = [0, 1/44100, 2/44100, ..., 44099/44100, 1]
    wave = np.sin(2 * np.pi * frequency * t)
        #The equation of a sine wave
    return wave
```

$$x(t) = A\sin(\omega t + \phi)$$

$$\omega = 2\pi f$$

http://musicweb.ucsd.edu/~trsmyth/sinusoids171/

Generating sound and music

Let’s hear the sound we’ve just created

```
single_note = create_sine_wave(NOTE_FREQUENCIES['A4'], 1)
```

#440Hz, 1 second

```
wavfile.write('single_note.wav', sample_rate, (single_note *
32767).astype(np.int16))
```

#$2^{16}/2-1 = 32767$ (bit depth is 16)

Generating sound and music

Let's build a chord with `create_sine_wave` function.

```
def create_chord(frequencies, duration, sample_rate=44100):
    waves = [create_sine_wave(f, duration, sample_rate) for f in frequencies]
        #Get the list of sine waves for each frequencies
    return np.mean(waves, axis=0)
        #Get the average values


chord_frequencies = [NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'],
NOTE_FREQUENCIES['G4']] #C major chord
chord = create_chord(chord_frequencies, 2)#C major chord (C,E,G) and 2 seconds
wavfile.write('chord.wav', sample_rate, (chord * 32767).astype(np.int16)
#Generated sample:
```

Generating sound and music

Let's build a chord progression with `create_chord` function.

```
    chord_progression = [
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'], NOTE_FREQUENCIES['G4']], 2.0),#C Major chord for 2 secs
        ([NOTE_FREQUENCIES['F3'], NOTE_FREQUENCIES['A3'], NOTE_FREQUENCIES['C4']], 1.0),#F Major chord for 1 sec
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'], NOTE_FREQUENCIES['G4']], 1.0),#C Major Chord for 1 sec
        ([NOTE_FREQUENCIES['F3'], NOTE_FREQUENCIES['A3'], NOTE_FREQUENCIES['C4']], 1.0),#F Major Chord for 1 sec
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'], NOTE_FREQUENCIES['G4']], 1.0),#C Major Chord for 1 sec
        ([NOTE_FREQUENCIES['G3'], NOTE_FREQUENCIES['B3'], NOTE_FREQUENCIES['D4']], 1.0),#G Major Chord for 1 sec
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'], NOTE_FREQUENCIES['G4']], 1.0)]#C Major Chord for 1 sec
    chord_waves = []
    for frequencies, duration in chord_progression:
        chord_wave = create_chord(frequencies, duration)
        chord_waves.append(chord_wave)
    backing_track = np.concatenate(chord_waves)
wavfile.write('backing_track.wav', sample_rate, (backing_track * 32767).astype(np.int16))
```

#Generated sample:

Generating sound and music

Let's build a melody with `create_sine_wave` function.

Let's define a melody from Twinkle Twinkle little star.

```
melody_notes = [
        # twinkle twinkle little star
        NOTE_FREQUENCIES[ 'C4'], NOTE_FREQUENCIES[ 'C4'], NOTE_FREQUENCIES[ 'G4'], NOTE_FREQUENCIES[ 'G4'],
        NOTE_FREQUENCIES[ 'A4'], NOTE_FREQUENCIES[ 'A4'], NOTE_FREQUENCIES[ 'G4'],
        # how I wonder what you are
        NOTE_FREQUENCIES[ 'F4'], NOTE_FREQUENCIES[ 'F4'], NOTE_FREQUENCIES[ 'E4'], NOTE_FREQUENCIES[ 'E4'],
        NOTE_FREQUENCIES[ 'D4'], NOTE_FREQUENCIES[ 'D4'], NOTE_FREQUENCIES[ 'C4']
    ]
melody_durations  = [0.5] * 14 #14 notes (0.5 sec for one beat)
melody_durations[ 6] = 1.0#The 7th note lasts for two beats
melody_durations[ -1] = 1.0#The last note lasts for two beats
```

Generating sound and music

```
def create_melody(notes, durations, sample_rate=44100):
    waves = []
    for note, duration in zip(notes, durations):
        wave = create_sine_wave(note, duration, sample_rate) #Generate a sine wave for each note
        waves.append(wave) #Append to the list
    return np.concatenate(waves) #Concatenate

melody = create_melody(melody_notes, melody_durations)
wavfile.write('melody.wav', sample_rate, (melody * 32767).astype(np.int16))
```

#Generated sample:

Generating sound and music

Finally, let’s combine the melody and the backing chord

```
melody = melody * 0.7 #adjust the amplitude
backing_track = backing_track * 0.3#adjust the amplitude
combined = melody + backing_track#combine
combined = combined / np.max(np.abs(combined))# normalize (to prevent clipping)
wavfile.write('combined.wav', sample_rate, (combined * 32767).astype(np.int16))
```

#Generated sample:

Take-homes

Work through the code (fairly similar to HW exercise!)

Worth understanding how sound is created (algorithmically) and stored, even if you’re less interested in (e.g.) the musical concepts

Data structures for music and data ingestion

1.5: Symbolic representations of music

Symbolic representations of music

Several slides based on

“Fundamentals of Music Processing (Meinard Müller, International Audio Laboratories Erlangen)

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C0/C0.html

and some from **“Generative AI for Music and Audio Creation” (Michigan):**

https://hermandong.com/teaching/pat498_598_fall2024/

What is the goal of a music representation?

Some possible (incompatible!) goals:

- Capable of describing existing (written) music (and therefore highly **flexible**)
- **Readable** by humans or machines
- Contain guidance about how the music should be **performed**
- **Playable** (or “readable”) by a device, such as an audio synthesis chip or a player piano
- Understandable by a language model (so that we can train **predictive** algorithms)

Assembly code

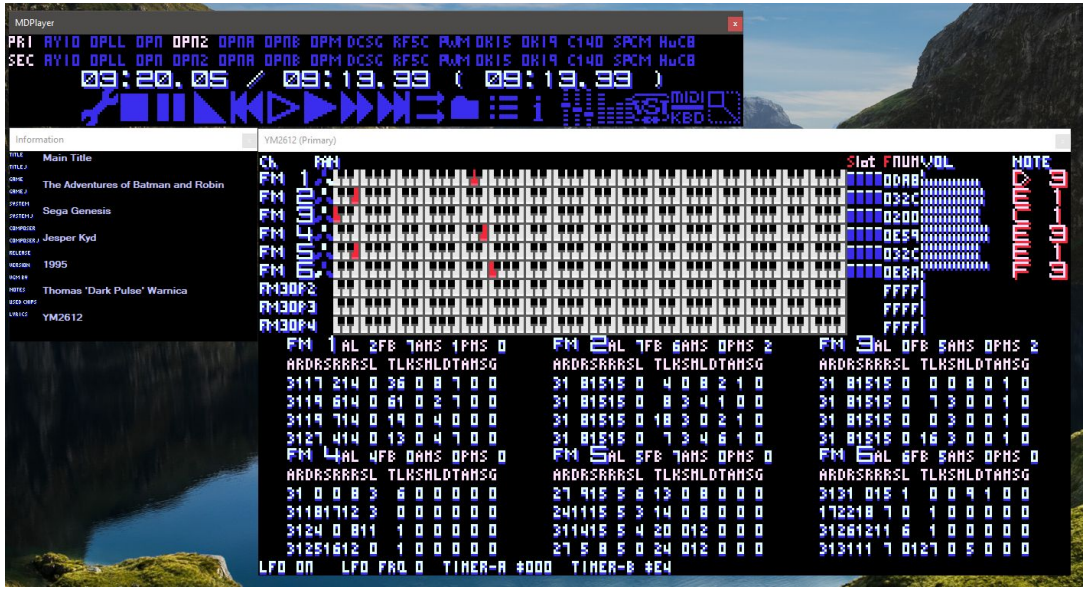
Represent music using the instructions used by an audio chip, e.g. the audio chip on an NES (from https://github.com/chrisdonahue/nesmdb):

```
clock,1789773  # NES system clock rat
fc mo,0        # Set frame counter to 4-step mod
ch no,1        # Turn on noise channe
ch tr,1        # Turn on triangle channe
ch p2,1        # Turn on Pulse 2 channe
ch p1,1        # Turn on Pulse 1 channe
w,13           # Wait 13 audio samples (13/44100 seconds)
tr lr,66       # Set the triangle linear counter load to 6
tr tl,172      # Set the lower 8 bits of the triangle clock divider to 1
w,1            # Wait 1 audio sampl
tr th,1        # Set the upper 3 bits of the triangle clock divider to
```

See e.g. https://irkenkitties.com/blog/2015/03/29/creating-sound-on-the-nes/

Assembly code

Tools also exist to play this type of music on modern devices, e.g. https://chipmusic.org/forums/topic/24423/searching-for-specific-vgm-player/



Assembly code

Why is this representation useful?

- Potentially promising as a language modeling (or “code”) representation of music
- Contains both musical notes and “performance attributes”, i.e., describes both *what* is played and *how* it should be played
- Reasonably expressive but also reasonably constrained (e.g. fixed set of instruments and sound profile)
- See NES-MDB (later)

Symbolic Representations

We saw that music can be represented via “code”; before proceeding further, let’s think about in what ways music is **similar to** language?

- Is inherently sequential, i.e., it can be represented as a sequence of (discrete) tokens
- Relatively small vocabulary of “words” (or tokens)
- Has its own syntax or “grammar” (i.e., what tokens generally follow others), that a language model could conceivably learn
- **Most** important relationships among notes (words) are relatively local
- Corpuses of symbolic music are available for training

Symbolic Representations

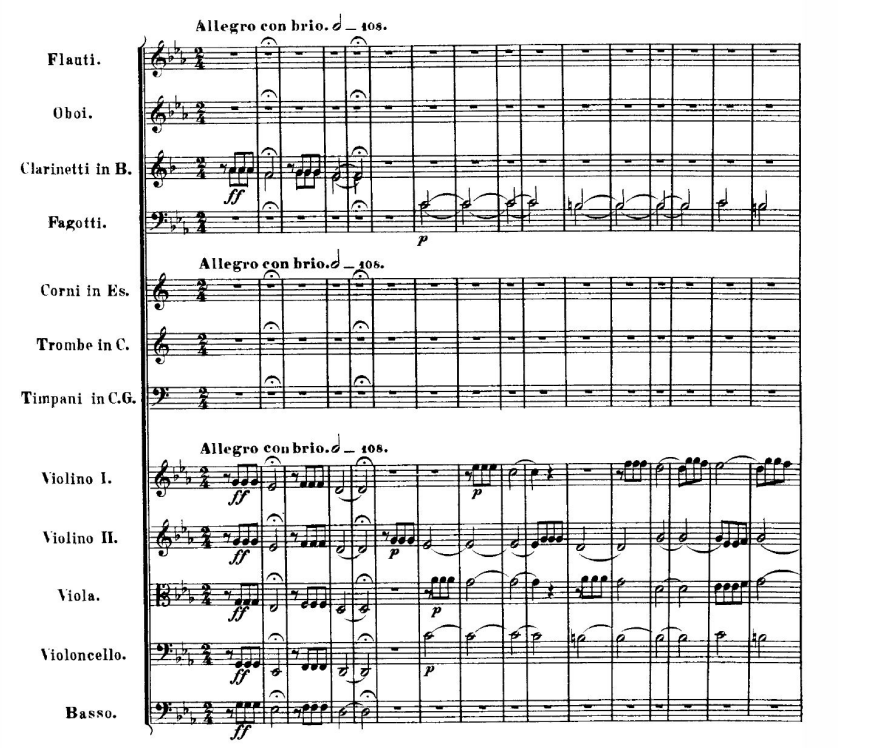
And in what ways is music **not** similar to language?

- Many events (“words”) can occur **simultaneously**
- Events have various types of metadata (e.g. velocity, duration, instrumentation)
- Precise beat and timing information has to be represented somehow
- Although many events are “local,” there is also very long context information that is hard to capture
- Training datasets are (relatively) extremely limited
- Generation is extremely difficult, even for highly-trained humans!

Sheet music

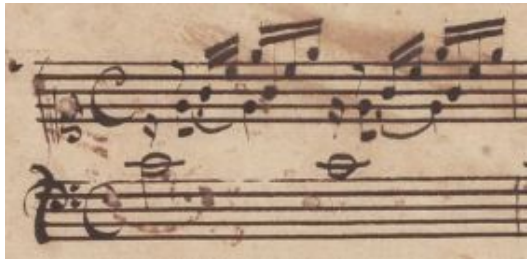
Assembly code provides instructions to a device to play a piece of music; *sheet music* and *musical notation* provides instructions to a human (or a group of people)

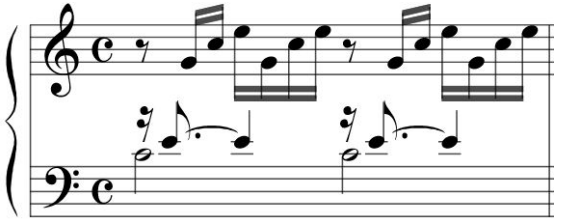
Arguably not really a “representation” in and of itself, but rather a *rendering* of some underlying musical language

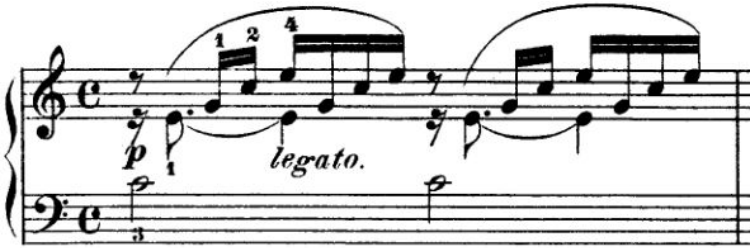


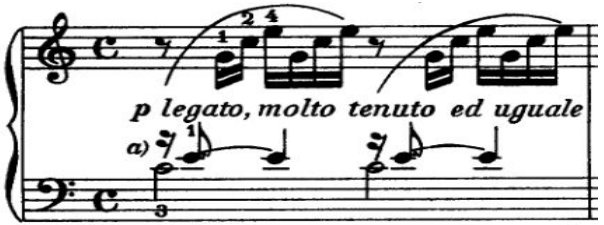
Sheet music

Lots of ambiguity!

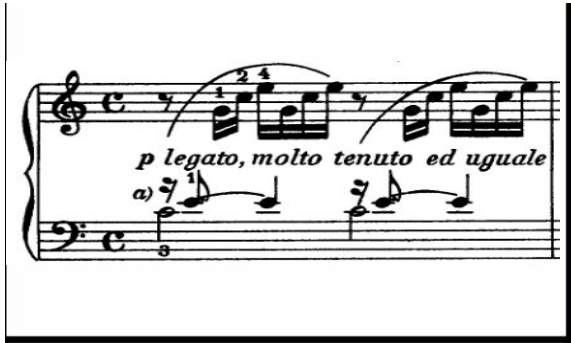


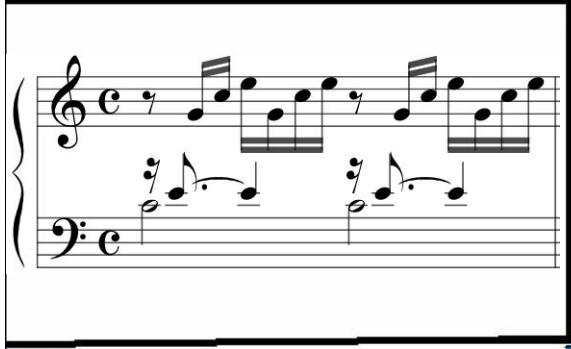


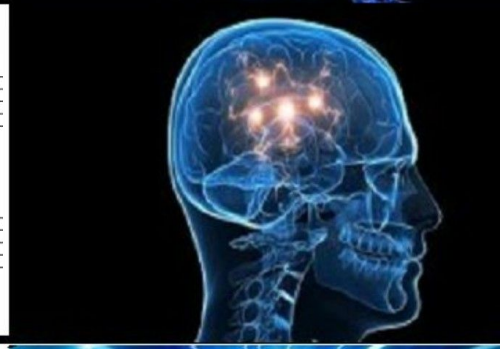


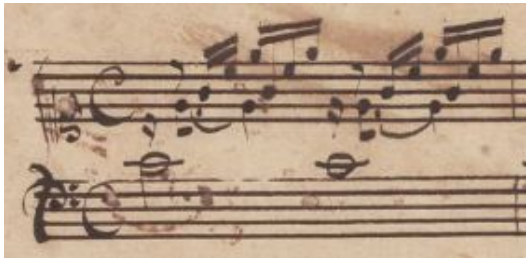


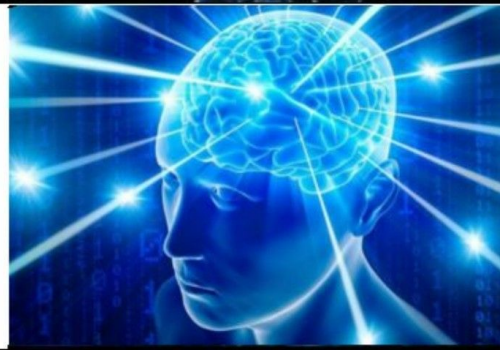
Sheet music





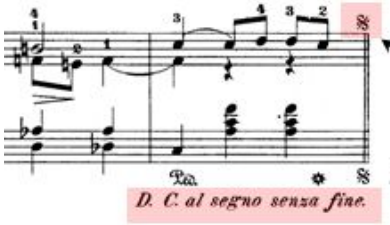






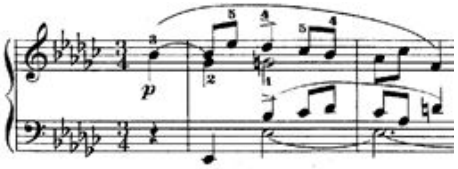
Sheet music

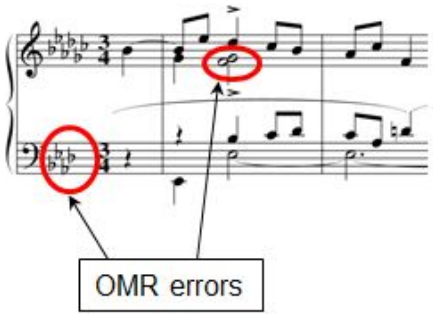
Lots of complexity!



Jump directives

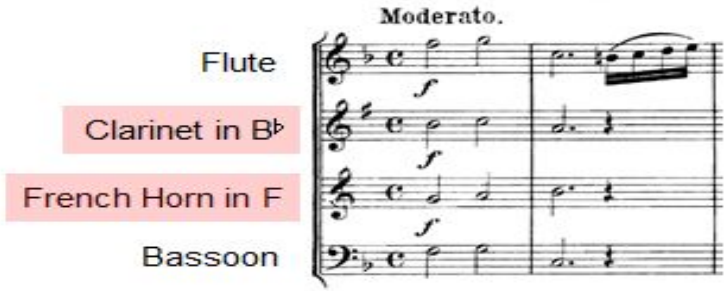
Repeat with alternative endings





OMR errors

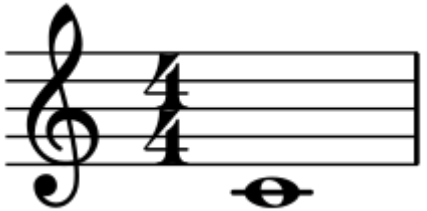
Transposing instruments



MusicXML

Music XML is a *markup language*, whose focus is on the representation and interchange of **musical notation;** that is, how the music should be *written*

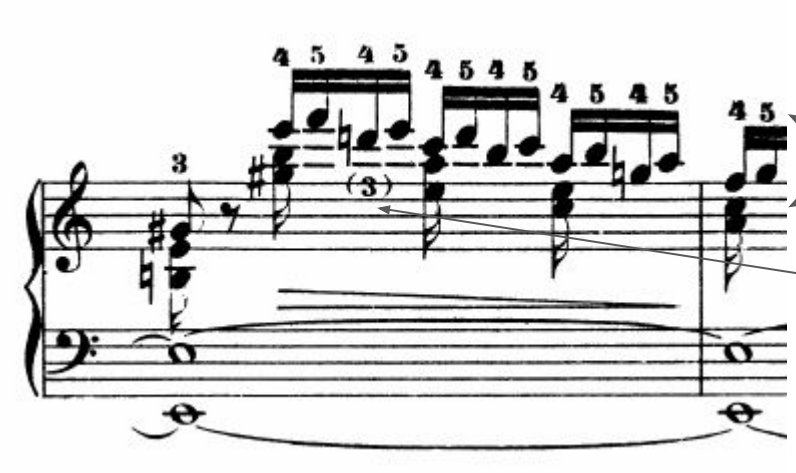
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 4.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="4.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
 </score-partwise>
```



MusicXML to render a (score of a) middle c (from wikipedia)

MusicXML

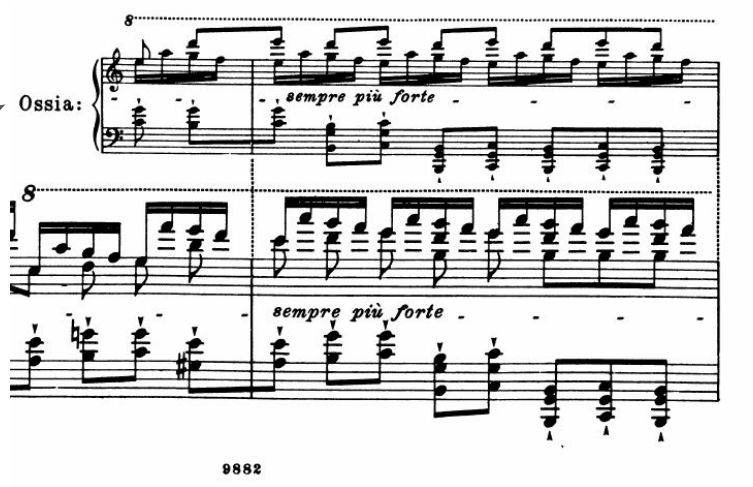
- Goal is to facilitate *rendering* rather than being read by a human (much like html), or being played by a device
- *Rendering* music is more about instructing readers *how to play something* rather than informing them about *what should be played*



MusicXML

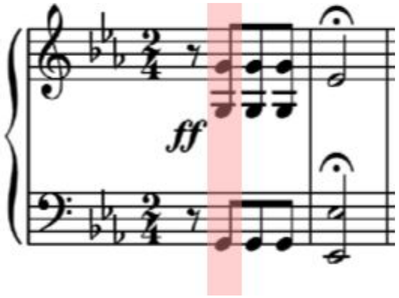
- Goal is to facilitate *rendering* rather than being read by a human (much like html), or being played by a device
- *Rendering* music is more about instructing readers *how to play something* rather than informing them about *what should be played*

Maybe play this instead if you like

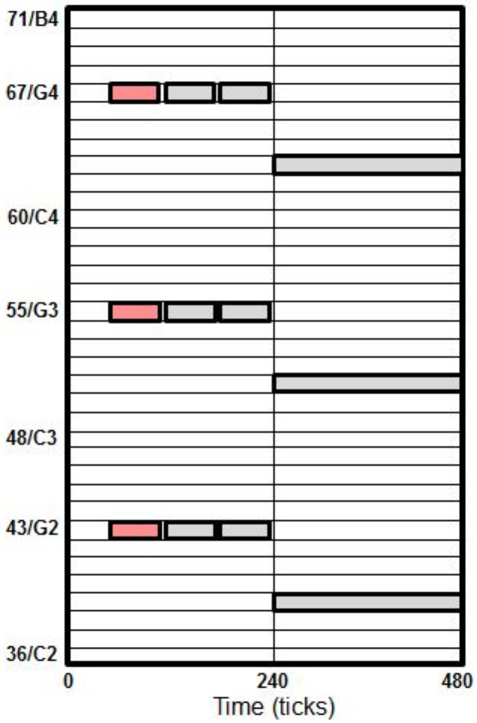


MusicXML

MusicXML is expressive enough to render sheet music, but our goal is probably to convert between formats



Time (Ticks)	Message	Channel	Note Number	Velocity
60	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	63	100
0	NOTE ON	2	51	100
0	NOTE ON	2	39	100
240	NOTE OFF	1	63	0
0	NOTE OFF	2	51	0
0	NOTE OFF	2	39	0



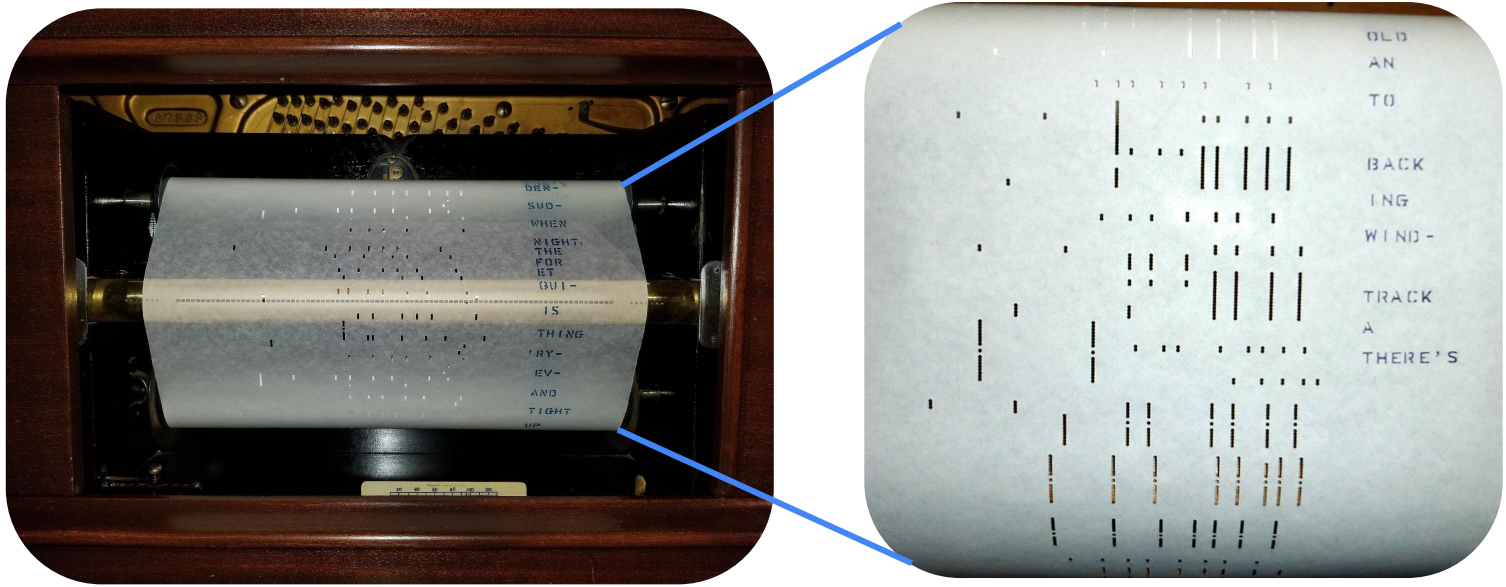
midi

Midi (Musical Instrument Digital Interface) is a way to track musical events for communication between devices; mostly just a way to *symbolically* record and control electronic instruments rather than a format we'd use for modeling (though there are models that generate midi files!)

(we'll explore this more in homework exercises and in workbook later on)

https://hackmd.io/@VinnyMan/understanding-the-in-depth-level-of-a-midi-file-format-for-developers

Piano rolls



(Source: Draconichiaro)

(Source: Tangerineduel)

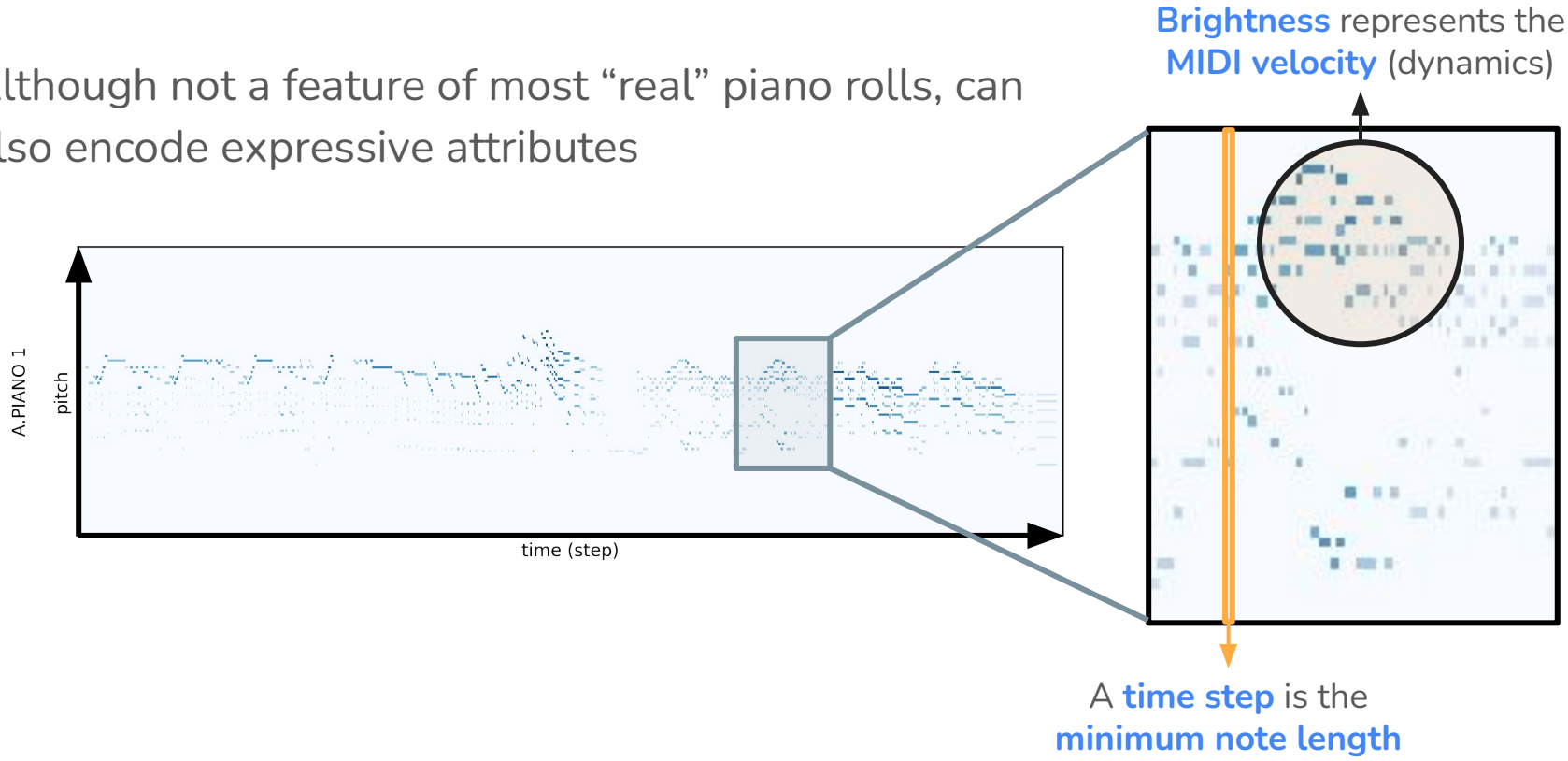
credit: Meinard Müller and Herman Dong

Player Pianos

https://www.youtube.com/watch?v=CfBUj7QKLKA

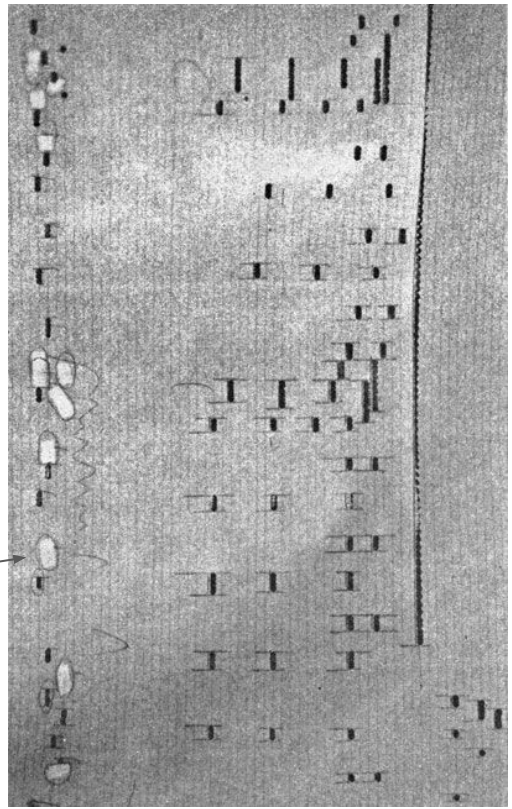
Piano roll representation

Although not a feature of most “real” piano rolls, can also encode expressive attributes



Piano roll representation

(some player pianos could reproduce dynamics, see e.g. https://www.pianola.org/reproducing/reproducing_welte.cfm for a lengthy discussion)

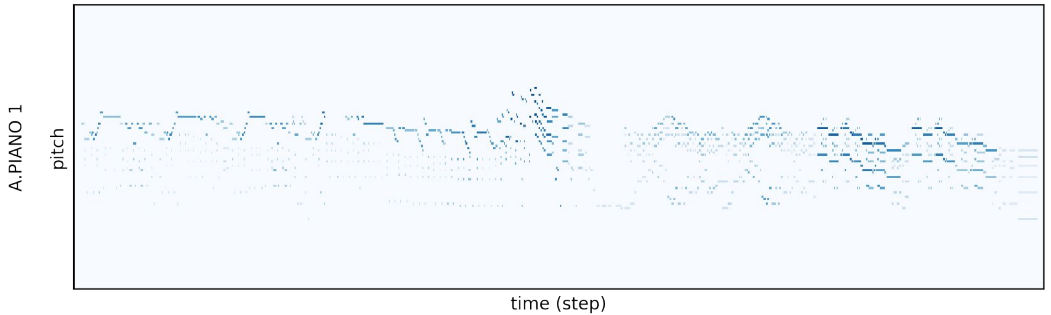


dynamics
(somehow)

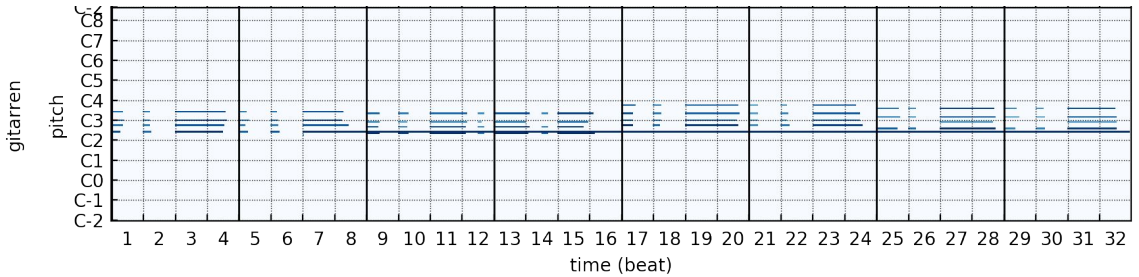
Piano roll representation

And may or may not be aligned to a beat:

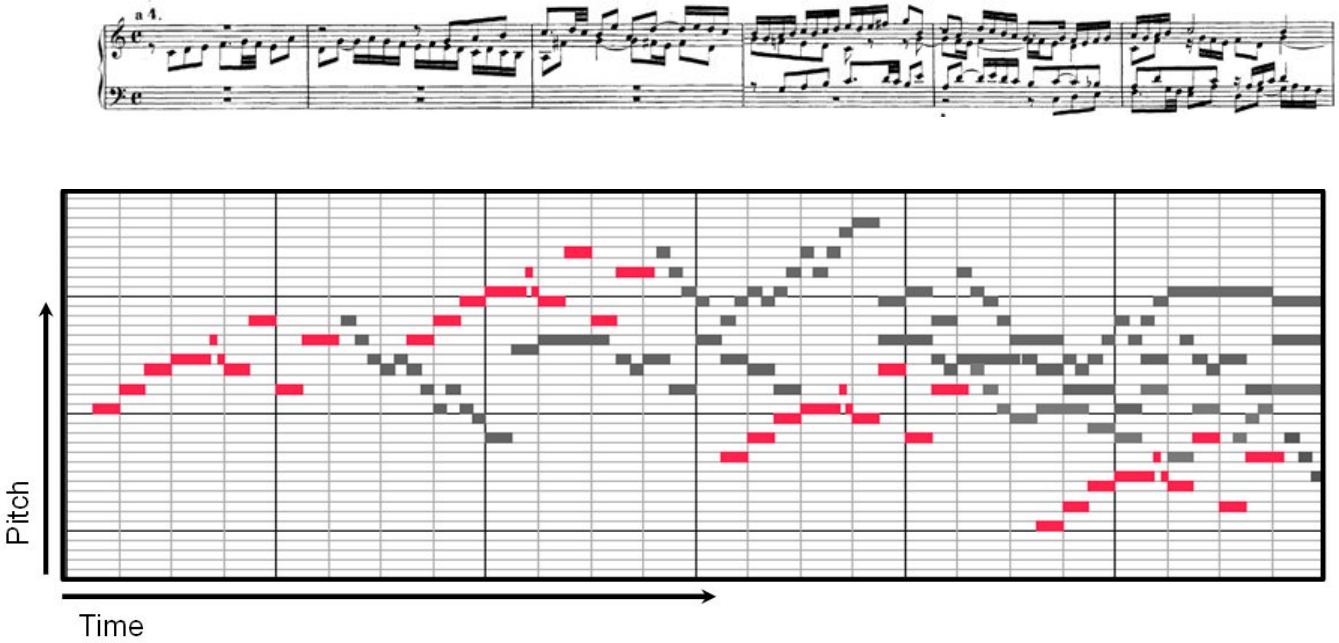
With expressive timing (and dynamics)



Without expressive timing

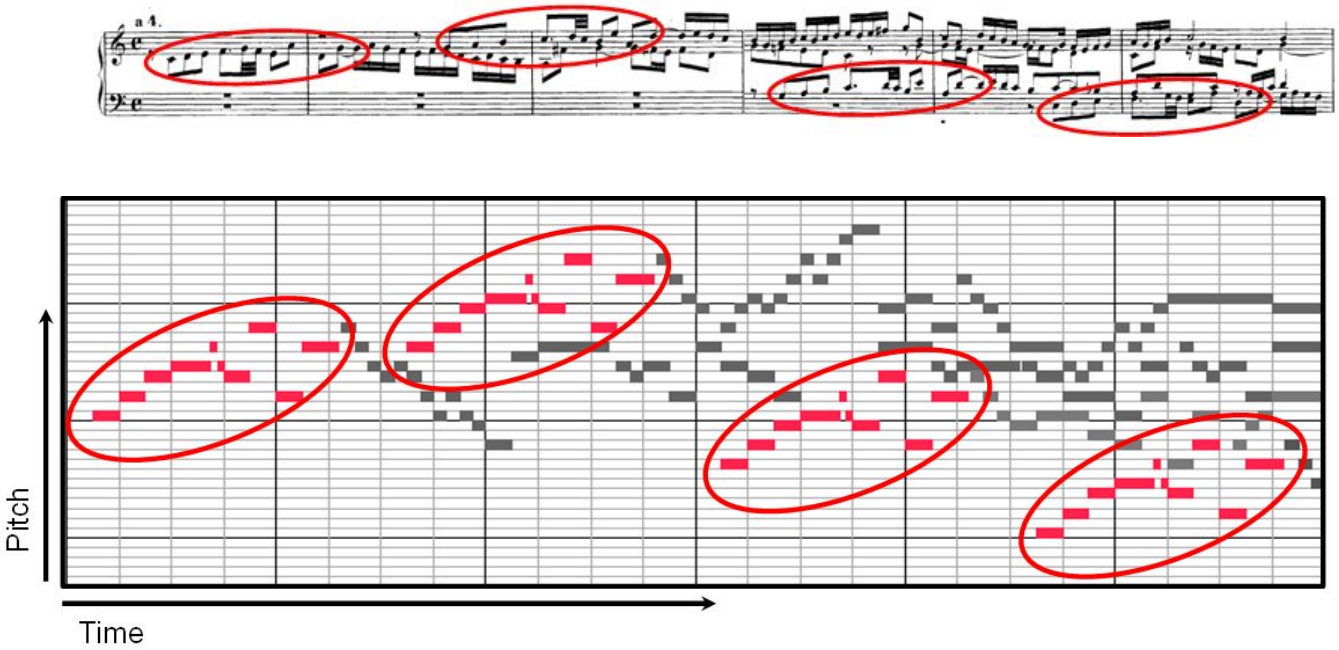


Piano roll representation



easy to see repeated patterns!

Piano roll representation



easy to see repeated patterns!

Piano roll representation

Note that in practice a “piano roll representation” is really a **matrix** representation of music:

x-axis: time (quantized)

y-axis: pitch (also quantized by scale degrees)

values: binary (when is the note on) or continuous (what is the velocity)

Piano roll representation

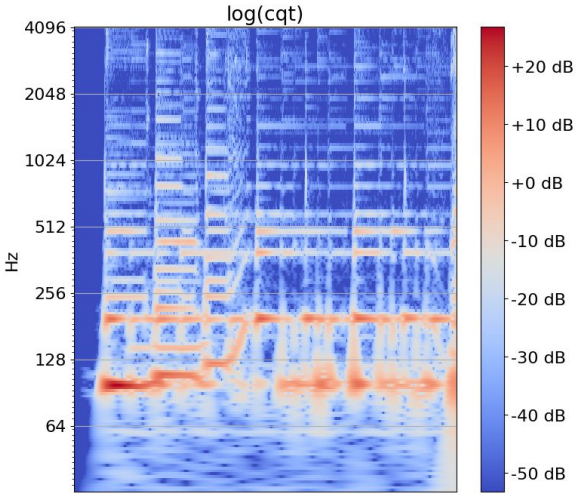
Why is this representation useful?

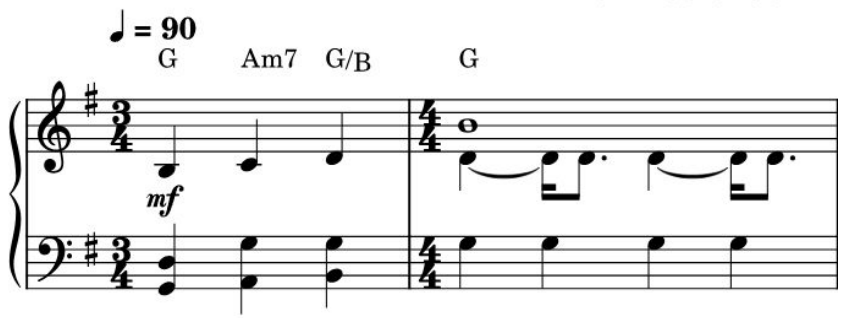
- Encodes time, pitch, and dynamics in a single object (a matrix)
- Straightforward to handle simultaneous (compared to e.g. language-ish representations)
- (later) Possibly will get the benefits we get from image models (CNNs): can learn invariances across pitch and time

Piano roll representation

Note that piano rolls are a *time-frequency* representation of music, i.e., they represent music much like a binary “image” whose x axis is time and whose y axis is frequency (or log_2 frequency)

As such while they are “symbolic” they are a precursor to continuous time-frequency representations that we’ll see later

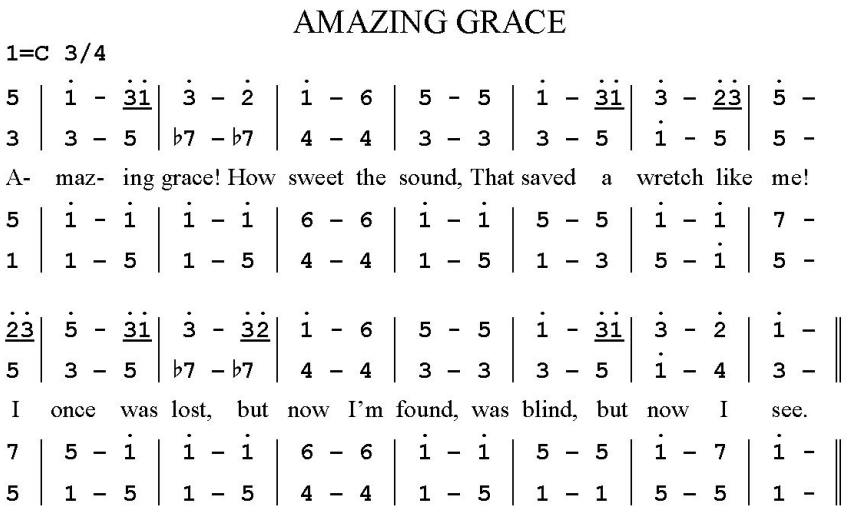




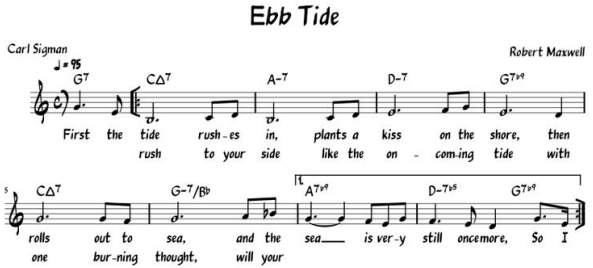
(see Module 2)

Custom formats designed for ML

For human-readable music, there are *lots* of alternative notation systems, some of which facilitate easy readability, compactness, pedagogy, etc.



numbered notation (from wikipedia)



lead sheet (from musescore)

Custom formats designed for ML

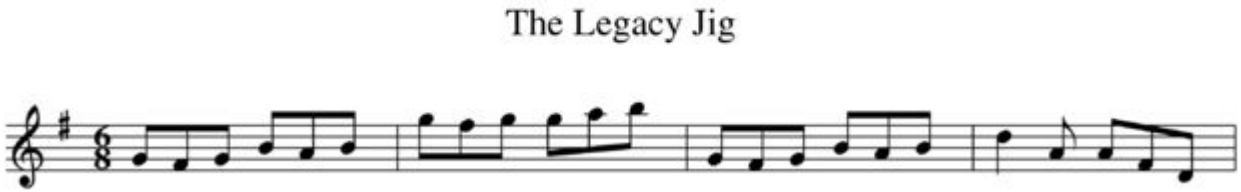
Likewise there are systems specifically designed to facilitate machine “readability”, which may (e.g.) facilitate downstream use by language models

ABC Notation (1997): https://abcnotation.com/

- Uses letters A-G for notes (e.g., C, D, E...)
- Supports rhythms, accidentals, and ornamentation
- a file (X:), the title (T:), the time signature (M:), the default note length (L:), the type of tune (R:) and the key (K:)

Custom formats designed for ML

ABC Notation Example (1997)

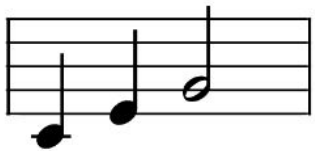



```
<score lang="ABC">
X:1
T:The Legacy Jig
M:6/8
L:1/8
R:jig
K:G
GFG BAB | gfg gab | GFG BAB | d2A AFD |
</score>
```

Custom formats designed for ML

REMI (Huang and Yang, 2020)

Encodes musical events with bar and position tokens to provide structured timing information

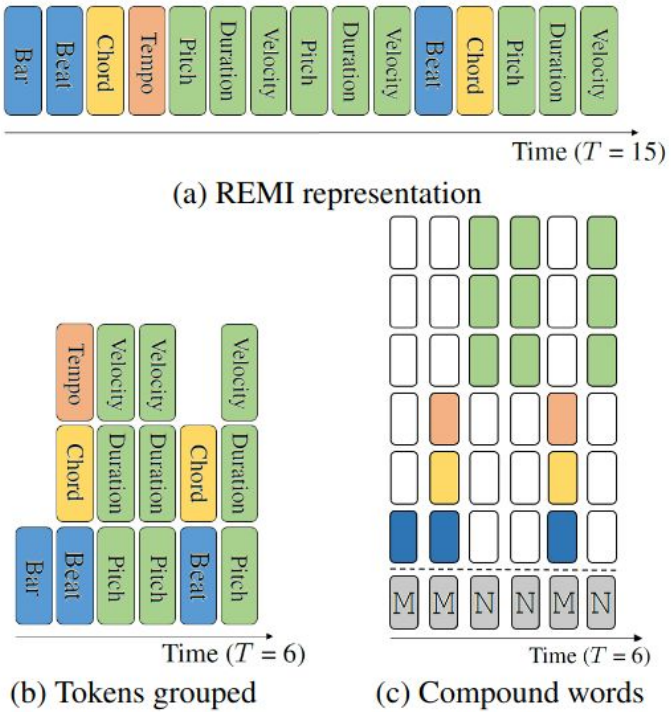


Bar, Position (1/16), Chord (C major),
Position (1/16), Tempo Class (mid),
Tempo Value (10), Position (1/16),
Note Velocity (16), Note On (60),
Note Duration (4), Position (5/16),
.....
Tempo Value (12), Position (9/16),
Note Velocity (14), Note On (67),
Note Duration (8), Bar

Custom formats designed for ML

Compound Word (Hsiao et al., 2021)

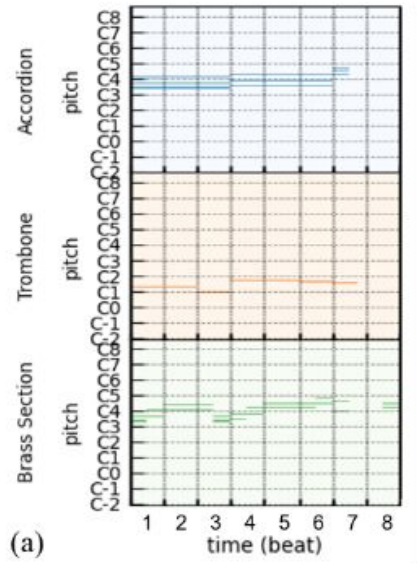
- REMI Tokens are grouped (for computational efficiency)
- `N`: note-related or `M`: metric-related



Custom formats designed for ML

Multitrack Symbolic Representation (Dong et al., 2022)

Representing melodies of multiple instruments for a song simultaneously



(a)

```
(0, 0,  0,  0,  0,  0)
(1, 0,  0,  0,  0, 15)
(1, 0,  0,  0,  0, 36)
(1, 0,  0,  0,  0, 39)
(2, 0,  0,  0,  0,  0)
(3, 1,  1, 41, 15, 36)
(3, 1,  1, 65,  4, 39)
(3, 1,  1, 65, 17, 15)
(3, 1,  1, 68,  4, 39)
(3, 1,  1, 68, 17, 15)
(3, 1,  1, 73, 17, 15)
(3, 1, 13, 68,  4, 39)
(3, 1, 13, 73,  4, 39)
(3, 2,  1, 73, 12, 39)
(3, 2,  1, 77, 12, 39)
   ...
(4, 0,  0,  0,  0,  0)
```

(b)

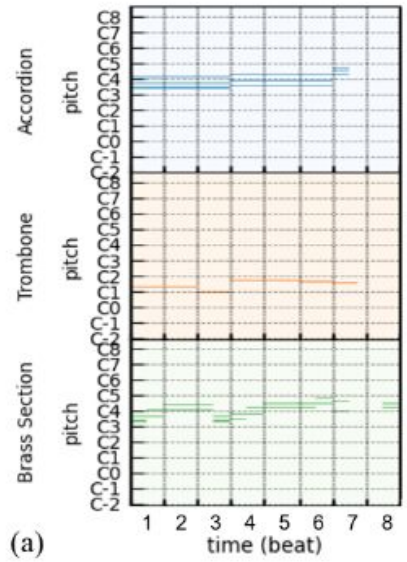
```
Start of song
Instrument: accordion
Instrument: trombone
Instrument: brasses
Start of notes
Note: beat=1, position=1,  pitch=E2, duration=48, instrument=trombone
Note: beat=1, position=1,  pitch=E4, duration=12, instrument=brasses
Note: beat=1, position=1,  pitch=E4, duration=72, instrument=accordion
Note: beat=1, position=1,  pitch=G4, duration=12, instrument=brasses
Note: beat=1, position=1,  pitch=G4, duration=72, instrument=accordion
Note: beat=1, position=1,  pitch=C5, duration=72, instrument=accordion
Note: beat=1, position=13, pitch=G4, duration=12, instrument=brasses
Note: beat=1, position=13, pitch=C5, duration=12, instrument=brasses
Note: beat=2, position=1,  pitch=C5, duration=36, instrument=brasses
Note: beat=2, position=1,  pitch=E5, duration=36, instrument=brasses
...
End of song
```

(c)

Custom formats designed for ML

Multitrack Symbolic Representation (Dong et al., 2022)

Representing melodies of multiple instruments for a song simultaneously




```
(0, 0,  0,  0,  0,  0)
(1, 0,  0,  0,  0, 15)
(1, 0,  0,  0,  0, 36)
(1, 0,  0,  0,  0, 39)
(2, 0,  0,  0,  0,  0)
(3, 1,  1, 41, 15, 36)
(3, 1,  1, 65,  4, 39)
(3, 1,  1, 65, 17, 15)
(3, 1,  1, 68,  4, 39)
(3, 1,  1, 68, 17, 15)
(3, 1,  1, 73, 17, 15)
(3, 1, 13, 68,  4, 39)
(3, 1, 13, 73,  4, 39)
(3, 2,  1, 73, 12, 39)
(3, 2,  1, 77, 12, 39)
   ...
(4, 0,  0,  0,  0,  0)
```

(b)

```
Start of song
Instrument: accordion
Instrument: trombone
Instrument: brasses
Start of notes
Note: beat=1, position=1,  pitch=E2, duration=48, instrument=trombone
Note: beat=1, position=1,  pitch=E4, duration=12, instrument=brasses
Note: beat=1, position=1,  pitch=E4, duration=72, instrument=accordion
Note: beat=1, position=1,  pitch=G4, duration=12, instrument=brasses
Note: beat=1, position=1,  pitch=G4, duration=72, instrument=accordion
Note: beat=1, position=1,  pitch=C5, duration=72, instrument=accordion
Note: beat=1, position=13, pitch=G4, duration=12, instrument=brasses
Note: beat=1, position=13, pitch=C5, duration=12, instrument=brasses
Note: beat=2, position=1,  pitch=C5, duration=36, instrument=brasses
Note: beat=2, position=1,  pitch=E5, duration=36, instrument=brasses
...
End of song
```

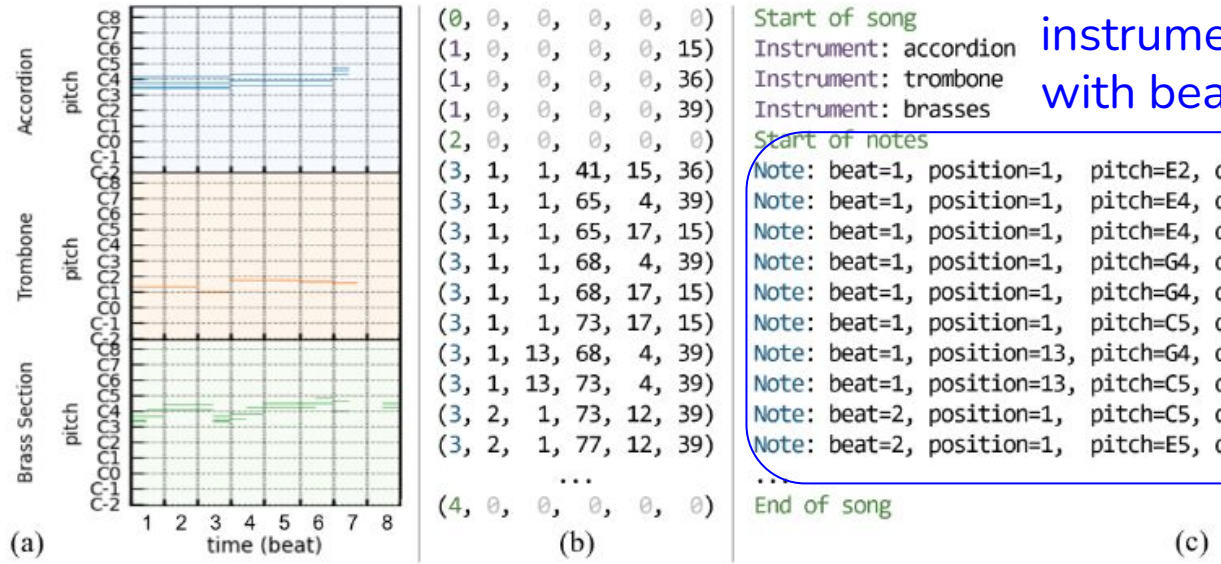
(c)

List every instrument at the beginning of symbols

Custom formats designed for ML

Multitrack Symbolic Representation (Dong et al., 2022)

Representing melodies of multiple instruments for a song simultaneously

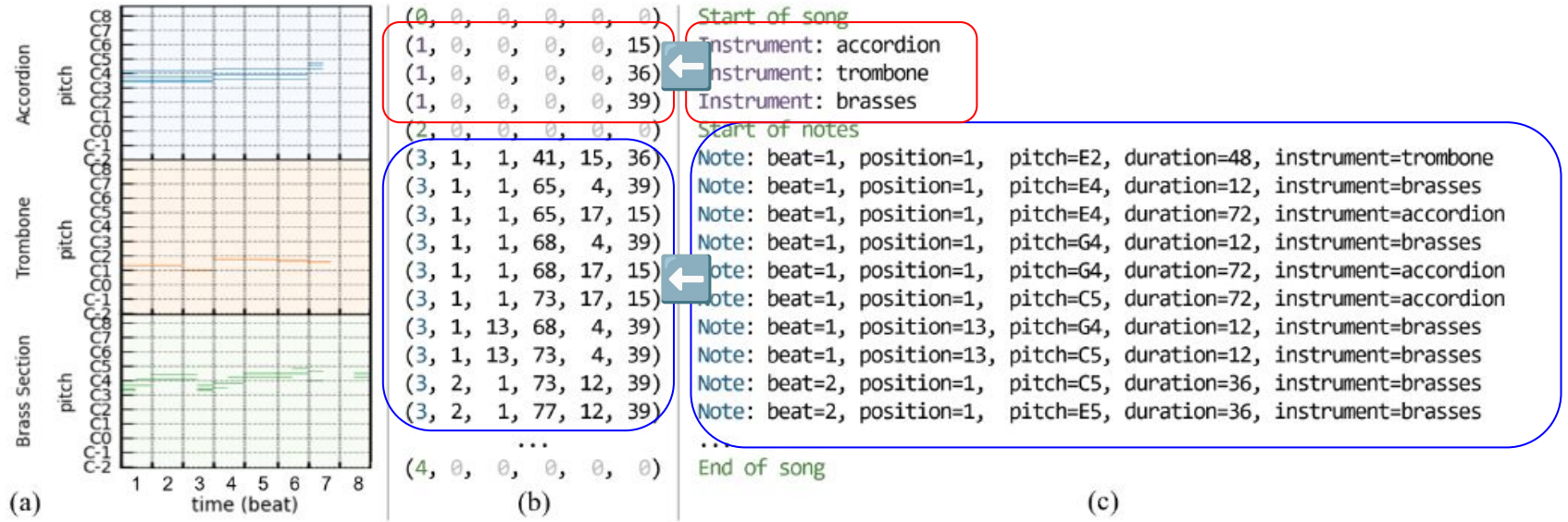


Each note is labeled with instrument information along with beat, position, pitch

Custom formats designed for ML

Multitrack Symbolic Representation (Dong et al., 2022)

This information is ultimately represented numerically (i.e., using “tokens”)



Custom formats designed for ML

Lots more in Module 3!

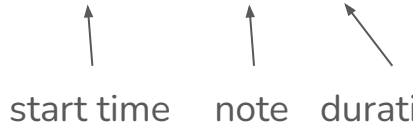
Some popular symbolic datasets

- PDMX: https://arxiv.org/abs/2409.10831
- Bach Chorales: https://archive.ics.uci.edu/dataset/25/bach+chorales
- Piano-midi.de
- Nottingham folk tune collection: https://abc.sourceforge.net/NMD/
- MuseData library: https://musedata.org/
- Magaloff Corpus: https://depts.washington.edu/icmpc11/ICMPC11/PDF/AUTHOR/MP100017.PDF
- Lakh MIDI: https://colinraffel.com/projects/lmd/
- EMOPIA: https://annahung31.github.io/EMOPIA/
- NES MDB: https://github.com/chrisdonahue/nesmdb

Some popular symbolic datasets

Bach Chorales: https://archive.ics.uci.edu/dataset/25/bach+chorales

```
(1 ((st 8)  (pitch 67) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 12) (pitch 67) (dur 8) (keysig 1) (timesig 12) (fermata 0))
   ((st 20) (pitch 74) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 24) (pitch 71) (dur 6) (keysig 1) (timesig 12) (fermata 0))
   ((st 30) (pitch 69) (dur 2) (keysig 1) (timesig 12) (fermata 0))((st 32) (pitch 67) (dur 4) (keysig 1) (timesig 12) (fermata 0))
   ((st 36) (pitch 67) (dur 6) (keysig 1) (timesig 12) (fermata 0))((st 42) (pitch 69) (dur 2) (keysig 1) (timesig 12) (fermata 0))
   ((st 44) (pitch 71) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 48) (pitch 69) (dur 8) (keysig 1) (timesig 12) (fermata 1))
```



Some popular symbolic datasets

Lakh Midi: https://colinraffel.com/projects/lmd/

"The Lakh MIDI dataset is a collection of 176,581 unique MIDI files, 45,129 of which have been matched and aligned to entries in the Million Song Dataset. Its goal is to facilitate large-scale music information retrieval, both symbolic (using the MIDI files alone) and audio content-based (using information extracted from the MIDI files as annotations for the matched audio files)."

LMD-full: The full collection of 176,581 deduped MIDI files

LMD-matched: A subset of 45,129 files from LMD-full which have been matched to entries in the Million Song Dataset (http://millionsongdataset.com/, not a symbolic dataset)

LMD-aligned: All of the files in LMD-matched, aligned to the 7digital preview MP3s from the Million Song Dataset

Some popular symbolic datasets

Magaloff Corpus:
https://depts.washington.edu/icmpc11/ICMPC11/PDF/AUTHOR/MP100017.PDF

"Magaloff performed Chopin's entire work for solo piano in strictly chronological order in six public appearances. The concerts were performed and recorded on a Bösendorfer computer-controlled grand piano. The recorded data comprises over 10 hours of continuous playing, over 150 pieces or more than 320,000 performed notes, precisely documenting the temporal and dynamic information for each played note"

Category	Pieces	Score Notes	Played Notes	Matches	Insertions	Omissions	Substitutions
Ballads	4	19511	20223	18971	1001	496	251
Etudes	24	40894	40863	38684	1615	1681	561
Impromptus	3	7216	7310	7150	96	159	64
Mazurkas	41	47312	47043	45260	1129	1669	470
Nocturnes	19	31109	32016	30943	671	873	302
Pieces	7	39759	41068	38249	1728	1487	916
Polonaises	7	27873	28301	26232	1597	1189	436
Preludes	25	20067	20239	19234	683	631	321
Rondos	3	18250	18331	17347	324	441	440
Scherzos	4	21951	22633	20849	1369	707	376
Sonatas	12	38971	40450	37015	1651	1498	731
Waltzes	8	18651	18876	18178	461	675	237

Some popular symbolic datasets

NES MDB: https://github.com/chrisdonahue/nesmdb

5278 songs, represented as assembly code, from the soundtracks of 397 NES games:

```
clock,1789773  # NES system clock rat
fc mo,0        # Set frame counter to 4-step mod
ch no,1        # Turn on noise channe
ch tr,1        # Turn on triangle channel
ch p2,1        # Turn on Pulse 2 channe
ch p1,1        # Turn on Pulse 1 channe
w,13           # Wait 13 audio samples (13/44100 seconds)
tr lr,66       # Set the triangle linear counter load to 66
tr tl,172      # Set the lower 8 bits of the triangle clock divider to 172
w,1            # Wait 1 audio sampl
tr th,1        # Set the upper 3 bits of the triangle clock divider to 1
```

Interesting in the sense that it is both symbolic and contains detailed expressive information

Data ingestion (and output)

Finally, let’s say a few words about reading and writing data… all code in **workbook1.ipynb**

Data ingestion (and output)

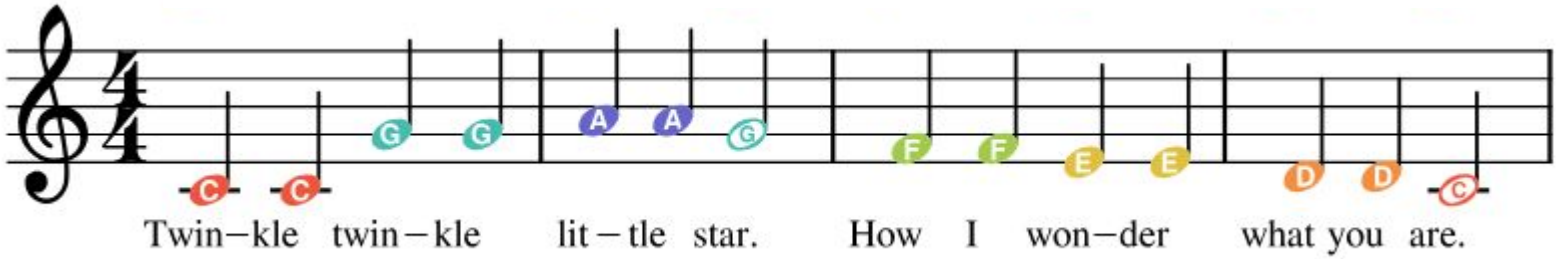
Make a new MIDI file

```
from midiutil import MIDIFile#Import library
midi = MIDIFile(1)#Create a MIDI file that consists of 1 track
track = 0#Set track number
time = 0#Just starting point
tempo = 120#Set track
midi.addTempo(track, time, tempo)#Add tempo information
```

Data ingestion (and output)

Write a new MIDI file (twinkle twinkle little star)

```
notes = [60, 60, 67, 67, 69, 69, 67,#CCGGAAG
         65, 65, 64, 64, 62, 62, 60]#FFEEDDA
durations = [1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2]
#beats
```



Data ingestion (and output)

Write a new MIDI file (twinkle twinkle little star)

```
current_time = 0
for pitch, duration in zip(notes, durations):
    midi.addNote(track, 0, pitch, current_time, duration, 100)
    current_time += duration
#add notes to MIDI file
with open("twinkle.mid", "wb") as f:
    midi.writeFile(f)#write MIDI file
```

note: you can drop the generated file to your DAW (e.g., logic pro, ableton live), or (probably) your system's default media player

Data ingestion (and output)

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-2	0	1	2	3	4	5	6	7	8	9	10	11
-1	12	13	14	15	16	17	18	19	20	21	22	23
0	24	25	26	27	28	29	30	31	32	33	34	35
1	36	37	38	39	40	41	42	43	44	45	46	47
2	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71
4	72	73	74	75	76	77	78	79	80	81	82	83
5	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107
7	108	109	110	111	112	113	114	115	116	117	118	119
8	120	121	122	123	124	125	126	127				

MIDI note numbers (“middle C” = 60)

Data ingestion (and output)

Convert MIDI file to audio file with a synthesizer

```
from midi2audio import FluidSynth#Import library
fs = FluidSynth('FluidR3Mono_GM.sf3')#Initialize FluidSynth
fs.midi_to_audio('twinkle.mid', 'twinkle.wav')#convert MIDI to audio
```

*Downlodable at https://github.com/musescore/MuseScore/blob/master/share/sound/FluidR3Mono_GM.sf3

Data ingestion (and output)

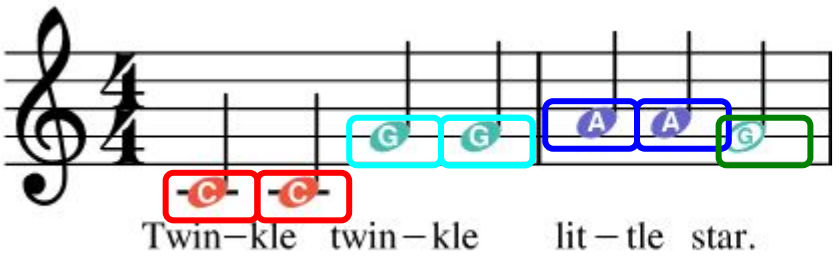
Create a melody stream and add metadata

```
from music21 import *
melody = stream.Stream()
melody.append(metadata.Metadata())
melody.metadata.title = 'Twinkle Twinkle Little Star'
melody.metadata.composer = 'Unknown'
melody.append(meter.TimeSignature('4/4'))
melody.append(key.Key('C'))
```

Data ingestion (and output)

Define and add notes

```
notes = [
    ('C4', 'quarter'),
    ('C4', 'quarter'),
    ('G4', 'quarter'),
    ('G4', 'quarter'),
    ('A4', 'quarter'),
    ('A4', 'quarter'),
    ('G4', 'half'),
]
for pitch, duration in notes:
    n = note.Note(pitch)
    n.duration.type = duration
    melody.append(n)
```



Data ingestion (and output)

Write a musicXML file

```
melody.write('musicxml', fp='twinkle_star.musicxml')
```

File format

File name

Take-homes

Don’t stress the details too much for now, other than to get a sense of the tradeoffs and ambiguities involved in music representation

We’ll explore these ideas more in **Module 2**

Data structures for music and data ingestion

1.6: Continuous representations of music

Continuous representations of music

Some slides from **“Deep Learning for Music Analysis and Generation”:**

https://github.com/affige/DeepMIR

and **“Fundamentals of Music Processing (Meinard Müller, International Audio Laboratories Erlangen):**

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C0/C0.html

Continuous representations of music

Fairly introductory section: we'll get into this a lot more in Module 2 when we dig into the main continuous format we'll use (spectrograms)

Mostly treat these slides as a reference in case these concepts come up later

Waveform

A waveform in audio is a visual representation of how sound pressure changes over time

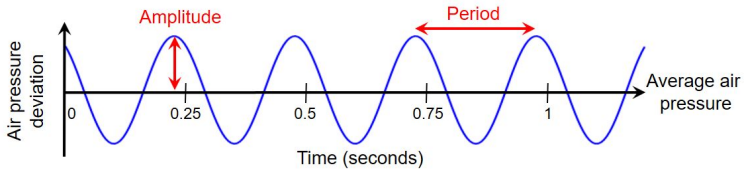
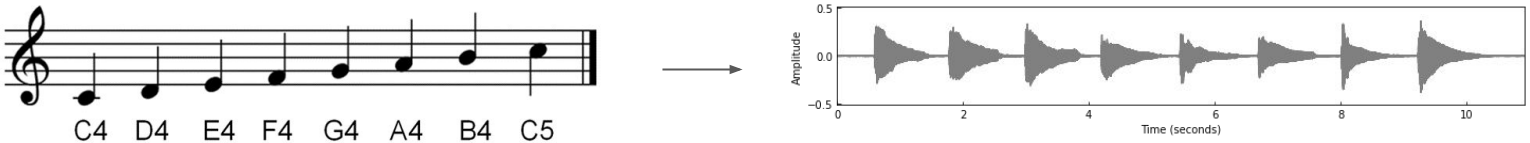


Figure 1.19 from [Müller, FMP, Springer 2015]



from: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Waveform.html

Stems

Music audio is a collection of audio that are mixed together. Each audio source (e.g., drum, vocal, guitar) is called a stem.

(Combined) music audio

bass

guitar

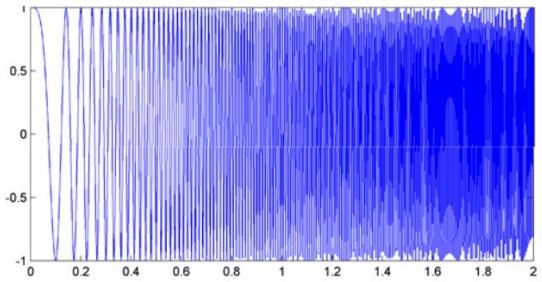
synth1

synth 2

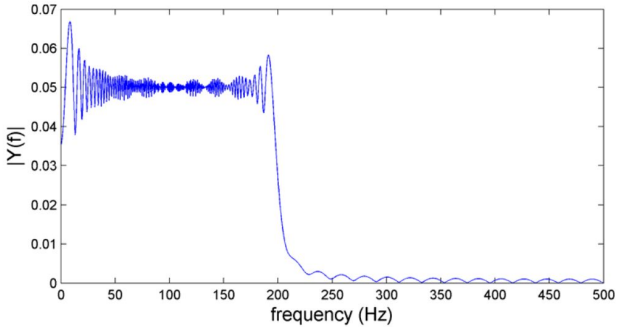
https://ummtone.com/what-are-stems-in-music-production/

Audio Representation

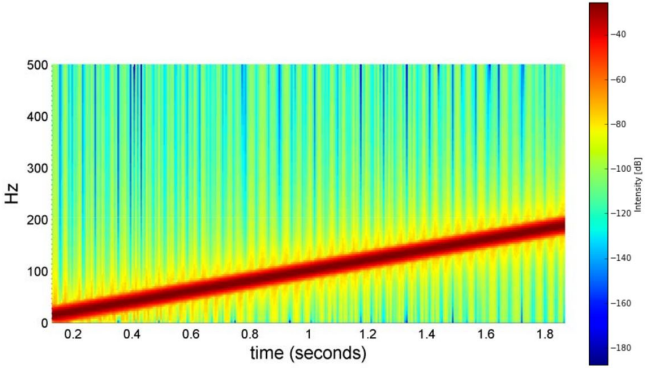
- **Waveform**: amplitude vs. time
- **Spectrum**: amplitude vs. frequency
- **Spectrogram**: time vs. frequency (color shows amplitude)



waveform



spectrum

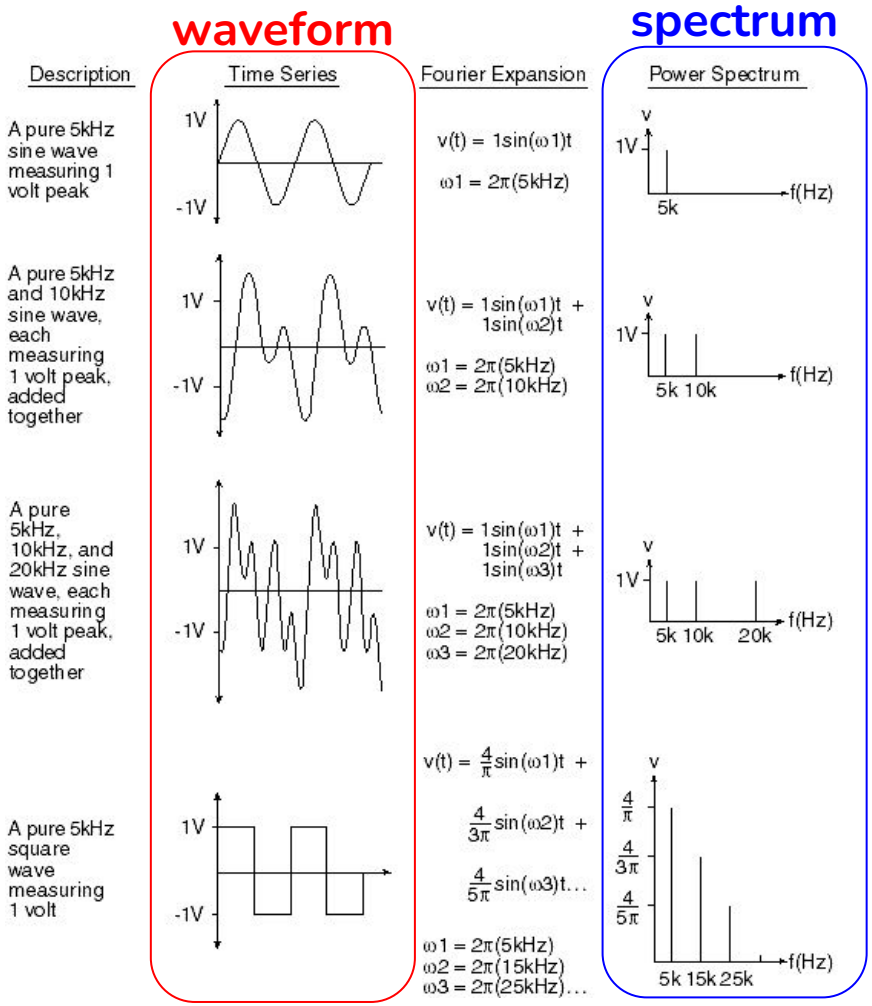


spectrogram

Spectrum

When you apply a **Fast Fourier Transform** to a waveform (time-domain signal), the result is its spectrum (frequency-domain representation)

(we'll cover this in **Module 2**)



Fundamental Frequency (F0)

What is f0?

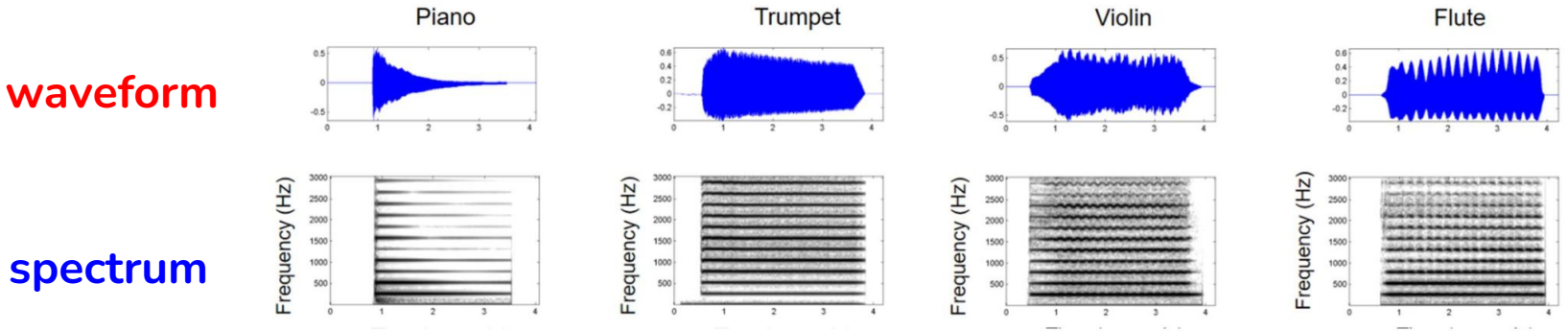
Fundamental frequency (f0) is the lowest frequency component of a periodic waveform; it represents the basic rate at which a signal repeats itself

Key Properties

- Determines the *perceived pitch* in speech and music
- Measured in Hertz (Hz)

Partials (harmonics)

- Partials (harmonics): usually at frequencies which are integer multiples of F0 (fundamental frequency)
- Different instruments have different patterns of partials.

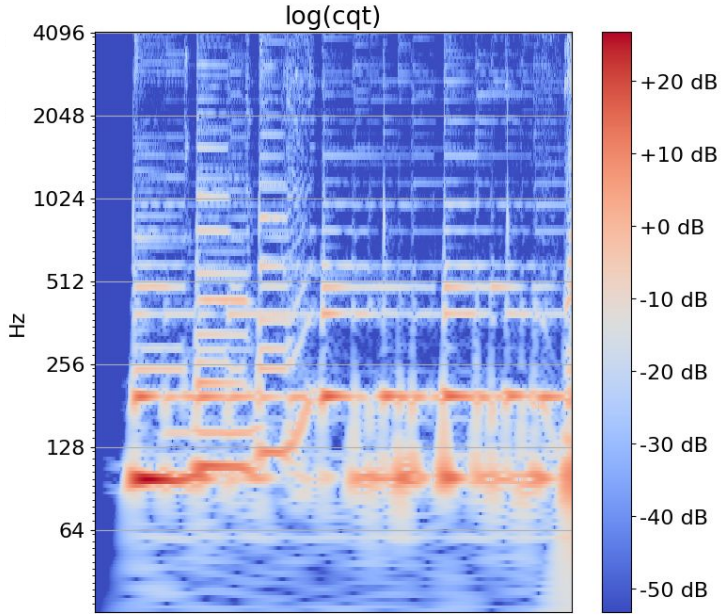


https://github.com/affige/DeepMIR/blob/main/2024/lecture02_fundamentals_audio.pdf

Spectrogram

- Time vs frequency representation
- color represents amplitude

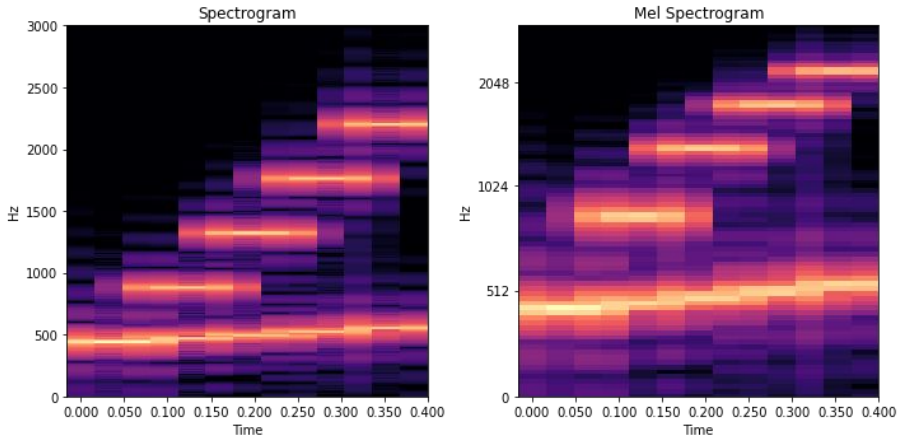
(don't worry too much about this for now, we'll cover in **Module 2**)



https://blogs.bl.uk/sound-and-vision/2018/09/seeing-sound-what-is-a-spectrogram.html

Mel-Spectrogram

- Variation of a spectrogram (modified to better match how human ears actually perceive sound)
- Higher frequencies are compressed, and lower range gets more space (because it's similar to how our ears work)



https://wikidocs.net/193588

Data structures for music and data ingestion

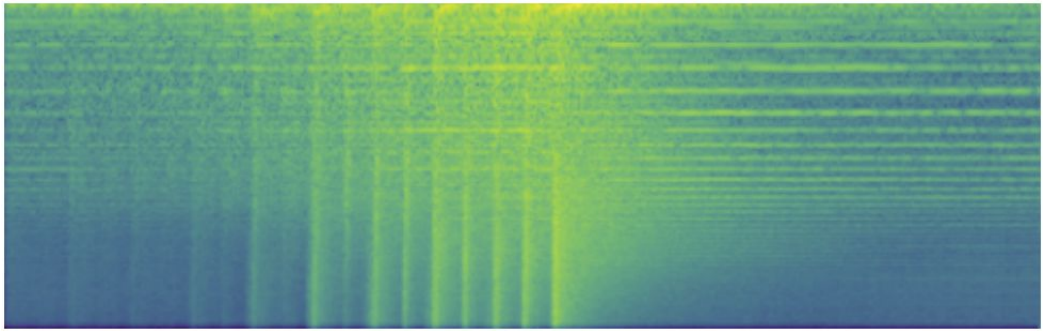
Conclusion

Is music discrete or continuous?

Based on what we've seen so far, do Music AI tasks seem closer to models from *vision* or from *language?*

Is music discrete or continuous?

Vision (continuous):



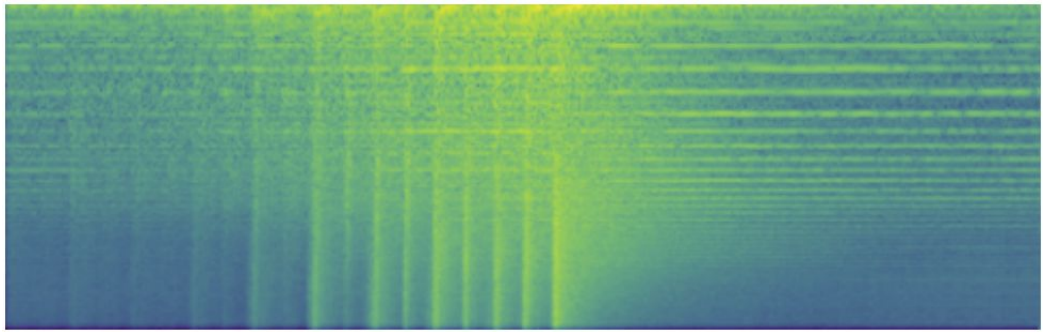
punk rock

(Note: while we conceptually treat audio as continuous, digital audio used in ML is actually discrete due to sampling and quantization)

Is music discrete or continuous?

Vision (continuous):

punk rock



Is music discrete or continuous?

Language (discrete):

```
(1 ((st 8)  (pitch 67) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 12) (pitch
67) (dur 8) (keysig 1) (timesig 12) (fermata 0))
   ((st 20) (pitch 74) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 24) (pitch
71) (dur 6) (keysig 1) (timesig 12) (fermata 0))
   ((st 30) (pitch 69) (dur 2) (keysig 1) (timesig 12) (fermata 0))((st 32) (pitch
67) (dur 4) (keysig 1) (timesig 12) (fermata 0))
   ((st 36) (pitch 67) (dur 6) (keysig 1) (timesig 12) (fermata 0))((st 42) (pitch
69) (dur 2) (keysig 1) (timesig 12) (fermata 0))
   ((st 44) (pitch 71) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 48) (pitch
69) (dur 8) (keysig 1) (timesig 12) (fermata 1))
```


classical

Is music discrete or continuous?

Language (discrete):

classical


```
(1 ((st 8)  (pitch 67) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 12) (pitch
67) (dur 8) (keysig 1) (timesig 12) (fermata 0))
   ((st 20) (pitch 74) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 24) (pitch
71) (dur 6) (keysig 1) (timesig 12) (fermata 0))
   ((st 30) (pitch 69) (dur 2) (keysig 1) (timesig 12) (fermata 0))((st 32) (pitch
67) (dur 4) (keysig 1) (timesig 12) (fermata 0))
   ((st 36) (pitch 67) (dur 6) (keysig 1) (timesig 12) (fermata 0))((st 42) (pitch
69) (dur 2) (keysig 1) (timesig 12) (fermata 0))
   ((st 44) (pitch 71) (dur 4) (keysig 1) (timesig 12) (fermata 0))((st 48) (pitch
69) (dur 8) (keysig 1) (timesig 12) (fermata 1))
```

Is music discrete or continuous?

All of these variants exist, e.g.:

- **Continuous, discriminative:** Dance Dance Convolution (Donahue et al., 2017)
- **Continuous, generative:** DITTO: Diffusion Inference-Time T-Optimization for Music Generation (Novack et al., 2024)
- **Discrete, discriminative:** Music Genre Classification Using MIDI and Audio Features (Cataltepe et al., 2007)
- **Discrete, generative:** Multitrack music transformer (Dong et al., 2023)

Is music discrete or continuous?

Arguments can be made for or against either representation:

Discrete (symbolic) – advantages:

- (Arguably) closer to the “raw” language of music
- More useful in various applications (e.g. if we want to provide playable music to humans)
- Less “superfluous” information; as an analogy, training a language model from *text* seems more straightforward than training a language model using *audio*
- (Potentially) allows us to leverage advances in LLMs, i.e., large-scale models for token generation, which can easily handle long-contexts etc.

Is music discrete or continuous?

Arguments can be made for or against either representation:

Discrete (symbolic) – disadvantages:

- Language models don't map to music *that* naturally: not easy to deal with simultaneous or overlapping events, additional information associated with "tokens" (dynamics, phrasing, etc.)

Is music discrete or continuous?

Arguments can be made for or against either representation:

Continuous – advantages:

- Closer to how humans *perceive* music
- Directly related to the inputs and outputs we might usually receive (i.e., actual audio files)
- Easy to deal with simultaneous events, dynamics, timbre, etc.
- (Potentially) allows us to leverage advances in image synthesis (e.g. diffusion models)

Is music discrete or continuous?

Arguments can be made for or against either representation:

Continuous – disadvantages:

- May be less useful to musicians as it doesn’t produce a playable score (is this a real issue?)
- May be harder to edit or control specific details (consider as an analogy editing the output of a language model vs a diffusion model)

Is music discrete or continuous?

In any case, the rest of this class will be concerned with both classes of approach:

Module 2: Discrete and continuous *input* models (i.e., “MIR”)

Module 3: Discrete *output* models

Module 4: Continuous *output* models