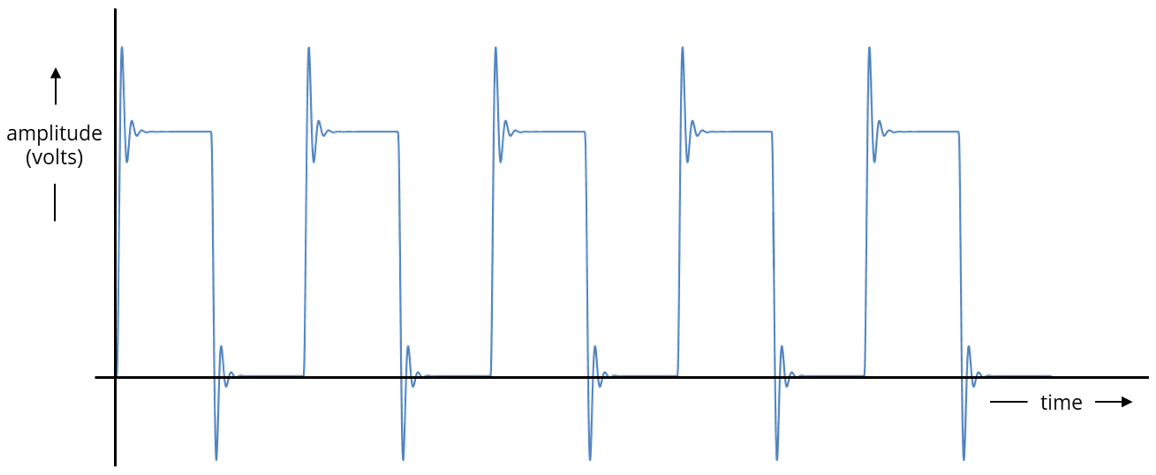
Research Notes

# Fast Fourier & Constant-Q Transform

## Time Domain

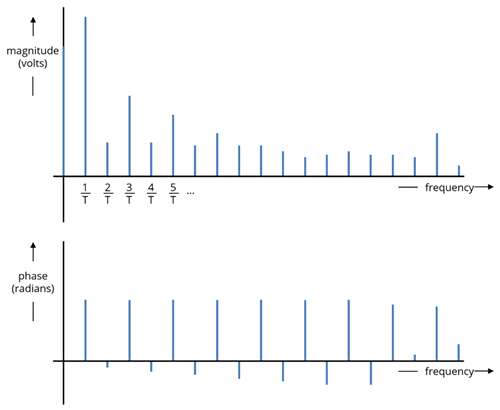
* Representation of a physical signal amplitude as a function of time
* A typical waveform looks like a signal represented in the time domain – displaying the amplitude (voltage) of the signal at a given time, across a certain timespan, as on an oscilloscope



* A real time domain graph would likely have more noise involved and would be far more uneven

## Frequency Domain

* Representation of a physical signal amplitude as a function of frequency
* Displays the distribution of signal frequencies of a signal over a given range of frequencies



* Essentially breaks down a signal into its different frequencies – extracting information such as identifying the dominant frequencies (fundemental pitch of a note), versus the presence of noise and harmonics in order to filter them out (i.e. using a low/high pass filter)

## (Fast) Fourier Transform (FFT)

* Fourier transform is a technique that can represent a signal as a collection of different frequency components, including amplitude and phase of these frequencies
* Transforms a signal from the **time domain** in order to be analysed in the **frequency domain**
* Used to analyse continuous signals (continuously changing values over time – analog signals, i.e. from a microphone)
* **Fast Fourier transform** is an **optimised version** of the standard Fourier transform, which is computationally expensive due to the amount of mathematical calculations being applied to potentially complex signals
* Is frequently used in audio processing, such as identification of pitch
* Some pre-written FFT implementations exist for C++, such as the **FFTW library**
* Accuracy of the algorithm is dependent on sampling rate and signal quality (ratio of actual sound content to background noise)
* FFT could be applied to microphone ADC to convert the time domain data into frequency domain data

### Discrete Fourier Transform (DFT)

* Discrete Fourier transform is the equivalent of FFT used for discrete signals (digital signals, like a .wav, .mp3)
* Discrete signals have distinct, identifiable values

### Frequency Bin

* Interval between samples in the frequency domain
* If sample rate is 100 Hz (number of samples, or peaks in the time domain), and the FFT size is 100 (number of frequencies in the frequency domain along the x axis), then you divide the sample rate by the number of frequencies to get frequency bin interval – 100 Hz / 100 samples = 1 (so you’d have **100** frequency bins each spanning a range of **1 Hz**)
  + If you instead took 200 samples – 100 Hz / 200 samples = 0.5 – you would have **200** bins of **0.5 Hz** in width
* Defines the **frequency resolution** (equal to individual bin width)
* Frequency ranges for musical notes, however, are not consistent and equally sized. For this reason, FFT may struggle to produce an accurate result

### Multi-Pitch Detection

* Very difficult with FFT – the overtones of one note may reinforce those of another note, resulting in a dominant frequency being identified that is not in fact a fundamental pitch

## Constant-Q Transform

* Like FFT, transforms a signal from the time domain to the frequency domain
* Unlike FFT, does not use a constant **window (bin) size** for all of the frequencies
  + The evenly-sized frequency bins do not map well to the geometrically spaced nature of note intervals on a musical instrument
* Constant-Q aims to allow bin widths to be defined as a **product** of the previous bin width, leading to progressively wider bin widths
* This transfers well to pitch detection – bins are wider the higher the frequency, and narrower when frequency is lower

# Other Pitch Detection Algorithms

## Additional: Interpolation & Piano Harmonics

* Estimates the values of unknown data points that fall between known data points
* Can be applied to audio to improve digital sound quality by filling the “gaps” in the discrete signals with predicted data
* Piano harmonics are usually integer multiples of the fundamental frequency (e.g. if the fundamental frequency of a middle C is 261.6 Hz, the first harmonic would be 2x this, 3rd would be 3x this etc.)

## Zero Crossing Rate (ZCR)

* Rate at which a signal moves from positive to zero to negative (or vice versa) within a time frame
* Could act as a primitive pitch detection algorithm – for single notes
* Operates in the time domain
* Typically operates better on percussive sounds rather than for identifying pitch

## Autocorrelation Algorithm

* Estimates the correlation between a signal and a copied, delayed version of itself
* Used to identify consistent patterns in a signal – if the signal is periodic, it will match itself and is said to have positive autocorrelation
* Computation is expensive
* Can be used to estimate a sound signal’s fundamental (lowest) frequency by using this pattern searching to identify pitch periods
* Operates in the time domain

## Harmonic Product Spectrum (HPS)

* Tone detection algorithm operating in the frequency domain
* Can detect fundamental frequency of a harmonic set
* Only works well for signals with a full set of harmonics – i.e. not simple sine waves
* Can use FFT output to produce a series of peaks corresponding to the fundamental frequency, plus the harmonic frequencies at integer multiples of the fundamental

# Rhythm Detection

* Could use a transform (Constant-Q or FFT) to obtain the different frequencies
* User could input BPM, time signature and number of bars into the system as parameters
* System could use these to split the audio signal into appropriately timed segments in order to calculate the correct note lengths for each new note played

# .midi VS .osc Files

* MIDI can only transmit 1-byte integers (0-255)
* OSC supports multiple data types, including 32-bit integers, floating points, strings
* MIDI is a binary message with limited fields
* OSC uses a more human-readable format, like a URL, in the form of words and numbers

# Pure Data Programming Environment

* Open source visual programming environment
* Can be used to process & generate sound and graphics, handle input devices and MIDI
* Can create software graphically (not manual coding-heavy)

# Sites Used

* <https://www.youtube.com/watch?v=piIujfV3Nsw>
* <https://www.youtube.com/watch?v=fYtVHhk3xJ0>
* <https://coertvonk.com/sw/arduino/pitch-detector/frequency-and-pitch-detection-31575>
* <https://iopscience.iop.org/article/10.1088/1742-6596/1722/1/012071/pdf#:~:text=Fast%20Fourier%20Transform%20Fast%20Fourier,voice%20and%20classic%20musical%20instruments>.
* <https://www.collimator.ai/reference-guides/what-is-a-fast-fourier-transform>
* <https://core.ac.uk/download/pdf/144846462.pdf>
* <https://surveillance9.sciencesconf.org/data/151173.pdf>
* <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6718976>
* <https://puredata.info>