Task 1 - Numerical Music

Overview

What is sound, after all?

Purpose

The purpose of this task is to familizarize the students with the field of signal processing, specifically audio signal processing, and provide insight into how programs such as FL Studio or Ableton work.

We will explore:

- How do we store an analog signal? What is stereo and mono?
- How do we create sounds artificially using an oscillator?
- How to "see" sound?
- I want to become a producer, how can I apply filters to sound?

studio.m

This is a script that you will use to manually test your functions. It creates different signals, applies filters and plot the spectrograms. You can edit this file however you like.

You will have to comment on the results of marked functions in the README.md, comparing them.

Storing signals. From stereo to mono

How do we store an analog signal? What is stereo and mono?

Analog to Digital

Analog signals are continuous signals, while digital signals are discrete signals. In order to store an analog signal, we need to convert it to a digital signal. This process is called **analog-to-digital conversion**. The process involves two main steps:

1. **Sampling**: The analog signal is sampled at regular intervals. The frequency at which the signal is sampled is called the **sampling rate**. The sampling rate is measured in **Hertz** (Hz). The higher the sampling rate, the more accurately the digital signal represents the original analog signal.
2. **Quantization**: The sampled signal is quantized. This means that the signal is divided into discrete levels. The number of levels is determined by the **bit depth**. The bit depth is measured in **bits**. The higher the bit depth, the more accurately the digital signal represents the original analog signal.

Sampling

It is obvious that in order to store a signal, we need to measure it at regular intervals. We would need infinte storage to store a continuous signal, so how can we work with less?

An important theorem in signal processing is the **Nyquist-Shannon sampling theorem**. It states that in order to accurately reconstruct a signal, the sampling rate must be at least twice the highest frequency present in the signal. This is known as the **Nyquist frequency**. If the sampling rate is lower than the Nyquist frequency, **aliasing** occurs, which distorts the reconstructed signal. The Nyquist frequency is calculated as:

$$f_{N_{quist}} = 2 \times f_{max}$$

where f_{max} is the highest frequency present in the signal.

Thus, what you will notice if you analize some audio files is that most of them have a sampling rate of 44100 Hz. This is because the human ear can hear frequencies up to around 20 kHz.

Quantization

Since the amplitude of a sine wave can be any real number, we need to define a "resolution" for our signal. This is done by quantizing the signal. As a very simple example, consider the number 3.14159. If we quantize it to 2 decimal places, we get 3.14. This is the same idea, but applied to the amplitude of the signal.

.wav files

The .wav file format is a common format for storing audio. Audio in this format can also be called uncompressed audio. This is because the audio is stored in a raw format, without any compression. This makes it easy to work with the audio, but the file sizes can be quite large. In MATLAB, you can read a .wav file using the audioread function.

The function returns two values: the audio data and the sampling rate. The audio data is a matrix containing the samples of the audio signal. The sampling rate is the number of samples per second. Each column in the matrix represents a channel of audio data.

Stereo vs. Mono

Audio can be stored in two main formats: **stereo** and **mono**. Stereo audio contains two channels: a left channel and a right channel. You can associate the left channel with the left speaker and the right channel with the right speaker. In mono, there is only one channel, which is played through both speakers.

Task

During this assignment, we will work only with mono since it is easier. Your first task is to write the function stereo_to_mono that takes in a matrix representing a signal with multilpe channels and returns a column vector according to the formula:

$$\text{mono}[i] = \frac{1}{n}\sum_{j=1}^{n} \text{signal}[i][j]$$

Look into the mean MATLAB function. Normalize the signal at the end.

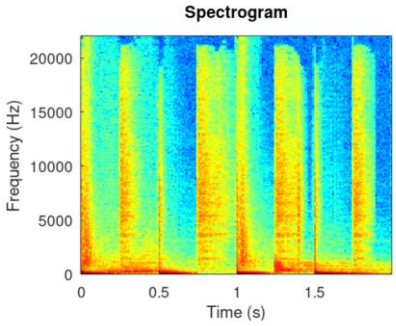
Spectrogram

How to "see" sound?

Introduction

A **spectrogram** is a visual representation of the spectrum of frequencies of a signal as it varies with time. It is a two-dimensional plot of the frequency w.r.t. time. It is a colored plot, where the color represents the amplitude of the signal at that time and frequency.

For example, the spectrogram of the loop.wav file included, will look like this:



Red indicates high amplitude, while blue indicates low amplitude, how present each frequency is at a specific time. The bread and butter of the spectrogram is the **Short-Time Fourier Transform**.

Short-Time Fourier Transform

The **Short-Time Fourier Transform** (STFT) is a technique used to analyze the frequency content of a signal as it changes over time. If we take the Fourier Transform of the entire signal, we would get all frequencies in the signal, but there is no way of knowing when each frequency appears, how many times it appears and how often it lasts.

The solution is simple: divide the signal into smaller pieces (windows) and compute the Fourier Transform of each window.

In practice, the computation is more complex, involving overlapping windows for smoother transitions between windows and a window function (think of it as interpolation). In this assignment, we will only apply the window function and have no overlap.

We will use the **Hann window** function. The Hann window is a weighted function that tapers the edges of the window to zero. This reduces the effect of the discontinuities at the edges of the window, which would otherwise introduce artifacts in the Fourier Transform.

Task

The function spectrogram takes as parameters the signal, sampling rate and the window size. It returns three values: the spectrogram matrix, the frequency vector and the time vector.

The spectrogram matrix is a 2D matrix where each row represents a frequency and each column represents a time. The value at position (i, j) represents the amplitude of the frequency i at time j.

The frequency vector is a column vector containing all the frequencies present in the signal.

The time vector is a column vector containing all the timestamps at which the STFT was computed.

These are the steps to compute the spectrogram:

1. Get the size of the signal and compute the number of windows. Use floor.
2. For each window:
    i. Apply the Hann function to it (hanning in MATLAB).
    ii. Compute the Fourier Transform of the window with a resolution twice the size of the window (fft in MATLAB).
    iii. Discard the conjugate part of the Fourier Transform.
    iv. Store the result in the spectrogram matrix.
3. Compute the frequency vector. Remember that the Fourier Transform is symmetric, so you only need the first half and look into what frequencies can the DFT represent.
4. Compute the time vector. The time between each window is the size of the window divided by the sampling rate. It starts from 0.

Comments

You will have to comment on the results of the spectrogram function in the README.md, for the marked functions in the studio.m. Compare them and explain why do they look like that.

Oscillator

How to create sounds artificially using an oscillator?

Introduction

An **oscillator** is an electronic circuit that produces a periodic, oscillating signal. In the context of audio signal processing, an oscillator generates a sound wave with a specific frequency and amplitude. Oscillators are used in synthesizers, audio effects, and other audio processing applications to create musical sounds.

Sine Wave Oscillator

In this was we will only consider the simplest type of oscillator: the **sine wave oscillator**. A sine wave oscillator generates a pure sine wave signal with a specific frequency and amplitude. The sine wave is the simplest type of waveform, consisting of a single frequency with no harmonics or overtones.

In practice, other types of waveforms are also used, such as square waves, sawtooth waves, and triangle waves. These waveforms have more complex harmonic content and can be used to create a wider variety of sounds.

Envelope

A very important feature in sound synthesis is the **envelope**. The envelope describes how the amplitude of the sound changes over time. It is usually described by four parameters:

- **Attack**: The time it takes for the sound to reach its maximum amplitude after the note is triggered.
- **Decay**: The time it takes for the sound to decrease from its maximum amplitude to the sustain level.
- **Sustain**: The amplitude level at which the sound is held after the decay phase.
- **Release**: The time it takes for the sound to fade out after the note is released.

This is called an **ADSR envelope**.

Task

The function oscillator takes as parameters the frequency of the sine wave, the duration of the created sound, the sampling rate and the envelope. It returns the generated sine wave as a column vector.

1. Create a time vector t from 0 to duration with a step of 1/sampling_rate. This is the time vector that will be used to generate the sine wave.
2. Create the sine wave using the formula sin(2*pi*frequency*t). This will generate a sine wave with the specified frequency.
3. Compute the number of attack samples, decay samples, sustain samples and release samples based on the envelope parameters and the sampling rate. Use floor to ensure that the number of samples is an integer. Notice that in order to obtain the number of sustain samples, you will need to subtract the number of attack, decay, and release samples from the total number of samples.
4. Compute the attack envelope using a linear ramp from 0 to 1 over the attack samples.
5. Compute the decay envelope using a linear ramp from 1 to the sustain level.
6. Compute the sustain envelope using a constant value equal to the sustain level.
7. Compute the release envelope using a linear ramp from the sustain level to 0 over the release samples.
8. Concatenate the attack, decay, sustain, and release envelopes to create the final envelope.
9. Multiply the sine wave by the envelope to apply the amplitude modulation (Hadamard product).

In practice, the envelope can be generated with other types of curves, not only linear ramps. For simplicity, we use only linear ramps in this assignment.

Created sound

By calling the create_sound function in studio.m, you can generate a sound based on music.csv. This is how it will sound:

You can play with music.csv as much as you want. Try to create different sounds by changing the frequency, duration, and envelope parameters.

High Pass Filter

How to remove high frequencies from a signal? Or, in extension, remove any kind of frequency from a signal?

Introduction

A **high-pass filter** is a filter that allows signals with a frequency higher than a certain cutoff frequency to pass through, while attenuating signals with frequencies lower than the cutoff frequency. This is useful in many applications, such as audio processing, where we might want to remove low frequency noise from a signal.

A very primitive filter

Knowing that the FFT of a signal gives us the frequency content of the signal, we can use this to filter out certain frequencies. The idea is to set to zero the FFT coefficients corresponding to the frequencies we want to remove, and then apply the inverse FFT to obtain the filtered signal.

Task

The function high_pass takes as parameters a signal x, a sampling rate fs, and a cutoff frequency fc. It returns the filtered signal as a column vector.

1. Compute the Fourier Transform of the signal x.
2. Compute all possible frequencies of the signal, similar to the spectrogram function.
3. Create a vector mask that is 1 for frequencies higher than the cutoff frequency fc, and 0 for frequencies lower than the cutoff frequency. Keep in mind the symmetry of the FFT. If you suppress a frequency in the first half, you also have to suppress it in the second half. For example, if you suppress the frequency X(1), you also have to suppress X(N).
4. Apply the Hadamard product between the Fourier Transform of the signal and the mask.
5. Compute the inverse Fourier Transform to get the filtered signal.
6. Normalize the filtered signal.

Some of you might notice that this is very similar to a convolution. In fact, filters can be seen as a convolution of the signal with the filter's impulse response. More of this in the next task. For now, this is good enough.

Reverb

How to make a sound feel like it's in a large room?

Introduction

Reverberation is the persistence of sound in a particular space after the original sound is produced. It is caused by the reflection of sound waves off surfaces in the environment (think of it like an echo). Reverberation is an important aspect of the perception of sound, as it can affect the timbre, loudness, and spatial characteristics of a sound. In music production, reverb is often used to create a sense of space and depth in a recording.

Impulse Response

One way to simulate reverb is by using an **impulse response**. An impulse response is the response of a system to an impulse signal. In the context of reverb, the impulse response represents the way sound decays in a particular space. By convolving the impulse response with a sound signal, we can simulate the effect of that space on the sound. You can find a list of impulse responses here, under IR data.

If you want to hear how your music would sound like in a church or a cave, download the respective IR and use it in the studio.m script.

This is what the sound obtained from the previous task sounds like with the s1r1.wav impulse response which came from a cave:

What is a convolution?

The convolution of two signals is a mathematical operation that combines the information from both signals. Think of it like applying a function represented by one signal to another signal. This is an awesome video explaining what convolutions are: https://youtu.be/KuXjwB4LzSA

Task

The function apply_reverb takes as parameters the signal, and the impulse response. It returns the signal with the impulse response applied.

1. Make sure that the impulse response is in mono by using stereo_to_mono.
2. Compute the convolution between the signal and the impulse response. (Hint: fftconv).
3. Normalize the resulting signal.