Deconvolution Assignment

Programming and Synthesis - Tutorial 1

Roberto Becerra

November 12, 2025

This tutorial and assignment explains the basics of Fourier analysis, Convolution and Deconvolution through a Python code example that demonstrates their working principle. By now you are probably well acquainted with the concepts of Fourier Transform / Analysis and Convolution. But just in case, let us go through some of them as we prime the way for this assignment.

This text contains a few formulas, but don't be intimidated by them!! as I will try to explain them bit by bit. If you still had any issues, we can discuss during the next lecture. All in all, they are here for the sake of building a conceptual framework. Anyway, you can skip to the code assignment section directly if you wish to.

The task in this assignment will be to record a sound file and complete the code that you will be given together with this document. The code imports a *sweep.wav* file and a *sweep-wet.wav* file, performs Fourier Transforms on them and uses Deconvolution to find the *impulse response* that you will be able to use in whichever *convolution reverb* plugin you may know or use.

How to work on this assignment

You will be given this document along 2 more files: - deconvolution.py - sweep.wav

Your task will be to record another wav file *sweep-wet.wav* and to complete the code in *deconvolution.py* step by step in the instructions further down this text.

You will only have to edit the code that is between this comments:

```
########################
# START YOUR CODE HERE #
########################
```

and

```
########################
#  END YOUR CODE HERE  #
########################
```

Once you have completed the task, please upload your resulting *ir.wav* file along with your *sweep-wet.wav* and your resulting python file *deconvolution.py* to our moodle assignment activity: https://moodle.lmta.lt/mod/assign/view.php?id=16272

The Fourier Theorem and Fast Fourier Transform (FFT) - intuition

The Fourier theorem states that any periodic signal can be expressed as the sum of a series of **sine** and **cosine** waves, each with specific amplitudes, frequencies, and phases. This decomposition is fundamental in signal processing and provides the foundation for operations like convolution and deconvolution.

The digital version Fourier transform, the Discrete Fourier Transform (DFT), converts a time-domain signal $x[n]$ into its frequency-domain representation $X[f]$:

$$X[f] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi fn/N} \tag{1}$$

Here, x is the signal that is dependent on time t. X upper case is the resulting Fourier transform of small x , and f stands for frequency. We could say that X is the amplitude per each sample (a moment in time) n, whereas X is the energy per each frequency f present in the signal x.

The summing symbol $\sum_{n=0}^{N-1}$ means that we scan and sum through every sample n from $n = 0$ and to $n = N - 1$, where N is the length of the input signal buffer. Effectively, when scanning all samples $x[n]$, we multiply them with the term $e^{-j2\pi fn/N}$, which represents a sine and a cosine at frequency f as they would behave over the same number of samples. The 2π means 360° or a whole cycle at frequency f. If this multiplication between x and the *sine* and *cosine* is like a comparison: if there is a match, the summation will accumulate to a large number, otherwise the summation will cancel itself and give values very close to zero for that frequency f.

A thing to remember is that $[n]$ is the *n-th* sample in a buffer of length N taken from signal x. This buffer size is the processing chunk in real time digital audio, such as the one you configure in your DAW when you want to adjust latency.

The term $e^{-j2\pi fn/N}$ is used since it is a *simpler*, more compact way to express the presence of both a *cosine* and a *sine* component in the signal we are analysing. This is because of an equivalence called the Euler's formula:

$$e^{-j2\pi fn/N} = \cos\left(\frac{2\pi fn}{N}\right) - j\sin\left(\frac{2\pi fn}{N}\right) \tag{2}$$

This means that the result of energy on the spectrum X at frequency f is represented by a bit of cosine and a bit of sine waves, which can be plotted in an x, y Cartesian unit circle, from where we can obtain information about the accumulated sum of the amplitude or the amount of energy (the size of the blue line), and phase of the frequency f (the phasee is the angle theta θ).

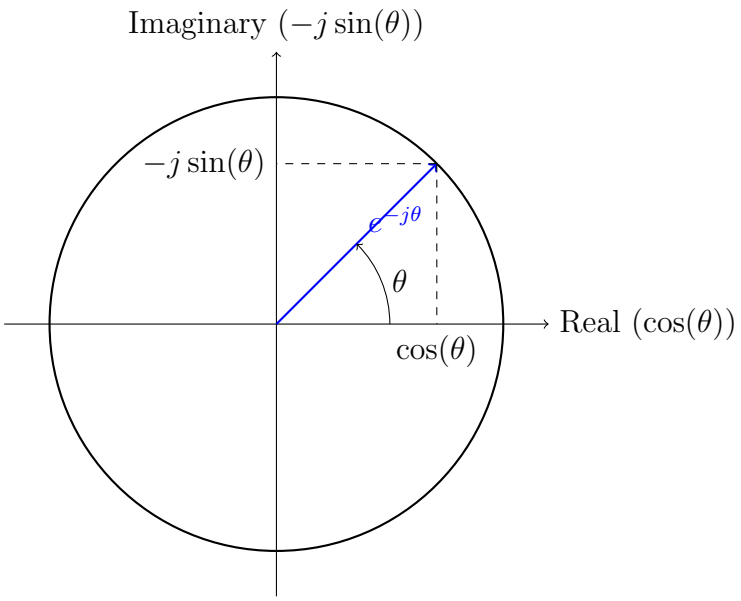

Figure 1: The unit circle showing the relationship between $\cos(\theta)$ and $-j\sin(\theta)$.

We can imagine as the Fourier transform generating a circle like the one in Figure 1 once for each of the frequency bins f in the spectrum buffer X.

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the DFT, significantly reducing the computation time, which is especially important for high-resolution audio signals. This is what we find in most modern software, like python or Max.

Applications in Audio Processing

The Fourier Transform and its efficient computation through the FFT have numerous applications in audio processing, sound design, and musical composition. Some key examples include:

- Frequency Analysis: Analyze the spectral content of a sound.
- Convolution and Deconvolution: Used in creating reverb, where the characteristics of a real acoustic space can be applied to a dry recording.
- Sound Design: Modify and manipulate the spectral components of sounds to create textures.
- Spectral Processing: Implement effects such as spectral filtering, spectral delay, etc.

- Pitch Shifting and Time Stretching: Use FFT information to modify the pitch or duration of a sound.

- Noise Reduction and Restoration: Isolate unwanted noise in the spectral domain .

- Additive Synthesis: Create complex sounds by summing sine waves of varying frequencies, amplitudes, and phases.

- Cross-Synthesis: Combine the spectral characteristics of one sound with the time-domain envelope of another.

- Spectral-Based Compositional Techniques**: Explore structure of sounds for spectral composition.

In this tutorial, we will use the FFT to perform deconvolution, as division in the frequency domain which is the opposite of convolution. This operation is particularly useful for applications such as impulse response processing, removing the effect of a room from a recording, or extracting the reverberation characteristics of a specific space for use in creative or corrective audio tasks. This is why it is also known as *dereverberation*.

What is Convolution?

Convolution is a mathematical operation that combines two signals to produce a third signal. In audio, this is often used to apply the characteristics of an impulse response (e.g., a room's acoustics) to another signal.

The good news is that Convolution can be calculated by simply multiplying the spectrum of one signal by the spectrum of another. In our case by multiplying the spectrum of x (X) by the spectrum of h (H), where h and H stand for *impulse response* signal and spectrum respectively.

$$Y[f] = X[f] * H[f] \tag{3}$$

This means that we first have to get the spectrum of the signals, i.e. by using an FFT and then we multiply them.

Convolution, impulse responses, reverb and sweeps.

As mentioned briefly before, convolutions is widely used to apply the reverb of a space to a dry sound source. This is done by *convolving* the sound source with the *impulse response* of a space.

This so called *impulse response* is short piece of audio that contains the characteristics of the reverb of a space (or system, i.e. a delay effect). It is called *impulse response* because it is a short recording of an *impulse* (something like a bang) inside of a space or system. The recording captures the *frequency response* and decay of a sound in the space or system. This bang could be quite literally a gunshot bang, because a very short impulse like this theoretically contains all frequencies, and we want to excite the space with all possible frequencies and record how it behaves.

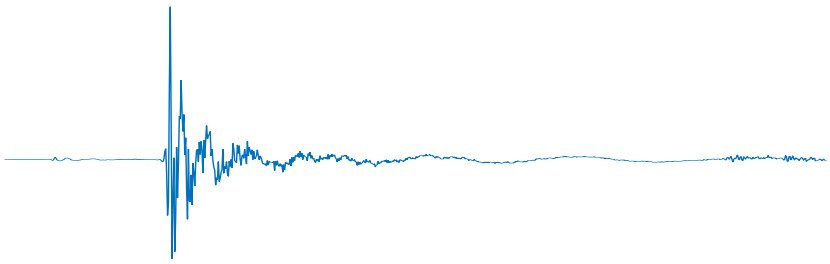

Figure 2: Example of an Impulse Response Signal. The plot shows the amplitude of the response over time.

In practice, the gunshot approach is not that accurate, because it its not exactly repeatable. Instead, all the frequencies are synthesized and played back on the space one after the other, in a fast sequence, using a so called *sine sweep*.

Attached together with this document you will have also a file *sweep.wav* which you will use for this assignment.

The process you have to do is as follows: You play the *sweep.wav* file over a loudspeaker (i.e. your laptop speakers) in the space you want to capture the *impulse response* from, and at the same time you record the sound on the space (i.e. with your laptop microphone). Then you have a recording, i.e. *sweep-wet.wav*.

Notice that this is NOT the impulse response. To actually get the impulse response you will need to perform a *Deconvolution*, since in essence what happened now is that you acoustically *convolved* the original *sweep.wav* sound with the *impulse response* of the space, and got the resulting *sweep-wet.wav* sound. So now we need to extract that *impulse response*.

Just one thing: when we say space, we may as well say system or any other transformation, for example a delay effect. In such case you would not record the sound of the space, but rather the sound out of the effect plugin/device. Finally, if you do decide to use your laptop speakers and microphone, what you will capture is mostly the impulse response of your laptop speakers.

Deconvolution

Deconvolution is the process of reversing the effects of convolution, commonly used in signal processing to extract useful information or recover an original signal. In music and in our case, deconvolution can be applied to analyse impulse responses of spaces, create unique reverberation effects, or generate experimental sounds.

Deconvolution attempts to reverse this process by estimating the impulse response's spectrum H given the output signal's Y and the input signal's X spectra . Mathematically:

$$X(f) = \frac{Y(f)}{H(f)} \tag{4}$$

where $X(f)$, $Y(f)$, and $H(f)$ are the Fourier transforms of the input signal, output signal, and impulse response, respectively.

In this way Deconvolution is typically performed in the frequency domain, as a simple division, simply the opposite of Convolution.

Exercise: Deconvolution in Python

For this assignment, you will be completing the following code to perform first convolution and then deconvolution and to obtain the impulse response of your own room! The following Python code implementes a basic deconvolution process using the `numpy` and `scipy` libraries. The python file and *sweep.wav* file is provided along this assignment file. The code is copied here as reference.

0.1 Step 1.

- Use np.fft.fft to compute the FFT of the sweep and sweep-wet signals. - Replace *None* with your answer - https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html

0.2 Step 2.

- Calculate the appropriate division to get the impulse response FFT - Replace *None* with your answer

0.3 Step 3.

- Use np.fft.ifft to compute the Inverse FFT to convert the impulse response FFT back to a time-domain signal - Replace *None* with your answer - https://numpy.org/doc/stable/reference/generated/numpy.fft.ifft.html

0.4 Step 4.

Once you have correctly completed the code, you will be able to run it and it will compute the impulse response and save it as a .wav file. Please upload this file along with your sweep-wet.wav file, and your python file. If moodle does not allow to upload python files, copy it as a .txt file.

The code is copied in here for reference, but you will find the original python file attached to this PDF.

Make sure all your file are inside the same folder for the script to work, and don't forget about your virtual environment! now lets go!:

```
import numpy as np
import scipy.io.wavfile as wavfile
from scipy.signal import fftconvolve

# Load the sweep and sweep-wet audio files
sweep_rate, sweep     = wavfile.read("sweep.wav")
wet_rate,   sweep_wet = wavfile.read("sweep-wet.wav")

# Ensure the sampling rates are the same
if sweep_rate != wet_rate:
    raise ValueError("Sampling rates don't match")

# Ensure the input lengths are appropriate
if len(sweep_wet) < len(sweep):
    raise ValueError("'length of sweep-wet' must be longer than
    ↪  'sweep.wav'.")

# Convolution and Deconvolution operations need a specific
```

```
# length of the buffers. Here we make sure the lengths
# are correct, and add zeros at the end if they are not.
# Estimating the original length of the impulse response
# L = lenght of sweep-wet, M = lenght of sweep,
# N = estimated lenght of impulse response
# if  L = M + N - 1 ,then

M =  len(sweep)
L =  len(sweep_wet)

N = L - M + 1

# We make the length of L to be a power of 2.
# It makes the algorithm more efficient.
closest_power_2 = np.ceil(np.log2(L))
new_L           = 2**closest_power_2

sweep     = np.pad(sweep,    (0,int(new_L-M)))
sweep_wet = np.pad(sweep_wet,(0,int(new_L-L)))

########################
# START YOUR CODE HERE #
########################

# STEP 1. Peform Fast Fourier Transform on both signals
sweep_fft   = None
wet_fft     = None

# STEP 2. Perform deconvolution in the frequency domain
impulse_fft = None

# STEP 3. Convert back to the time domain
impulse_response = None

########################
#  END YOUR CODE HERE  #
########################


impulse_response = impulse_response[0:N]
```

```
# Normalize the impulse response to prevent clipping
impulse_response /= np.max(np.abs(impulse_response))

# Save the impulse response to a new file
ir_filename = "ir.wav"
wavfile.write(        ir_filename,
                        sweep_rate,
                        (impulse_response*32767).astype(np.int
                        ↪  16))

# Plot the impulse response for visualization
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.title("Impulse Response")
plt.plot(impulse_response)
plt.xlabel("Samples")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()

print(f"Impulse response saved to '{ir_filename}'")
```

Congratulations! you made it this far. If you managed to follow all the way through, you will be now more familiar with the depths of digital sound that make possible our music making using computers. Knowledge is power and this can translate into creative ideas.

Challenges and Considerations

- Noise Sensitivity: Deconvolution is sensitive to noise. And this method in fact is a simplified way of doing it. Regularization techniques, such as Wiener filtering, can help mitigate this issue. But those is far more advanced and beyond this assignment. (https://en.wikipedia.org/wiki/Wiener_deconvolution)

Conclusion

Deconvolution is a tool for signal analysis, enabling processing of signals and impulse responses. With Python, we can explore and experiment with these

techniques and eventually think of new creative ways to use this knowledge.

0.5 Notes

- Disclaimer: I used chat GPT to generate about 20 percent of this text, and most of the formatting of the equations and the format of the file.