Digital Signal Processing - Lecture Notes - 5

E. Yazıcı, SRH Uni Heidelberg

June 2023

Sinusoidal Generator and DFT Synthesis

In this section we will generate a simple sinusoidal signal. The equation for a continuous-time sinusoidal signal can be written as:

$$x(t) = A\sin(2\pi ft + \phi) \tag{1}$$

where A is the amplitude, f is the frequency in Hz, and ϕ is the phase in radians. For a discrete-time sinusoid sampled at a rate f_s, we use:

$$x[n] = A\sin\left(2\pi\frac{f}{f_s}n + \phi\right) \tag{2}$$

where n is the sample index. Now, let's generate a discrete-time sinusoid of frequency 5 Hz, sampled at 100 Hz, with an amplitude of 1 and zero phase.

Now, let's compute the DFT of this signal. The DFT for a sequence $x[n]$ is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{3}$$

where N is the length of the sequence and k is the frequency index. In Python, we can use the FFT function from the numpy library to compute DFT.

After calculating the DFT, we obtain a set of complex coefficients, which represent the frequency content of the signal. To recover our original signal from the DFT coefficients, we use the inverse DFT (IDFT), which is given by:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N} \tag{4}$$

Here is the Python code to generate a sinusoidal signal, compute its DFT and recover the signal:

```
import numpy as np
import matplotlib.pyplot as plt

# Signal parameters
f = 5     # Frequency in Hz
fs = 100  # Sampling rate in Hz
t = 1     # Duration in seconds
A = 1     # Amplitude
phi = 0   # Phase

# Generate the time array
time = np.arange(0, t, 1/fs)

# Generate the sinusoidal signal
x = A * np.sin(2 * np.pi * f * time + phi)

# Compute DFT
X = np.fft.fft(x)

# Recover the signal from DFT coefficients
x_recovered = np.fft.ifft(X)

# Plot the original and recovered signals
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(time, x, label='Original')
plt.title('Original Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)
plt.plot(time, x_recovered.real, label='Recovered')
plt.title('Recovered Signal from DFT Coefficients')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()
```

MP3 Data

MP3, which stands for MPEG Audio Layer III, is a method used for audio data compression. It is a lossy compression method, meaning it reduces the amount of data needed to represent the audio recording and still sounds like a faithful reproduction of the original uncompressed audio to most listeners.

The process of MP3 compression involves a number of steps, but at a high level, it can be seen as a process of removing parts of the sound that are considered to be beyond the auditory resolution ability of most people.

MP3 Compression

The MP3 compression algorithm involves several steps:

1. **Filter Bank:** The first step in MP3 compression is to break up the audio signal into 576-sample blocks, which are then transformed into the frequency domain using a type of Fourier Transform called a Modified Discrete Cosine Transform (MDCT). The use of the MDCT results in spectral lines that are equally spaced at approximately 34.13 Hz intervals.

2. **Psychoacoustic Model:** The next step is the application of a psychoacoustic model, which is used to estimate the sensitivity of human hearing at different frequencies. The model uses principles such as masking, which is a phenomenon where certain sounds make it difficult or impossible to hear other sounds in a certain frequency range.

3. **Quantization and Coding:** After the psychoacoustic model determines which parts of the audio can be removed, the remaining frequency coefficients are quantized and coded, which is a lossy process. The quantized values are then formatted into an MP3 frame that will be decoded by the MP3 player.

4. **Entropy Coding:** This final stage compresses the data further by removing statistical redundancy. The most common method used in MP3 is Huffman coding.

The Fourier Transform plays a crucial role in the first step of the MP3 compression algorithm. By transforming the audio signal into the frequency domain, it allows the algorithm to identify the frequencies where the audio signal has the most energy, and it is these parts of the signal that are preserved during the quantization and coding stage.

It's important to note that MP3 compression is a delicate balance between the desire for a smaller file size and the need to preserve the quality of the audio. The process inherently involves discarding data, so the challenge is to decide which data is least important and can be removed with the least impact on the perceived sound quality.

Regarding Python examples, I must clarify that implementing the MP3 algorithm from scratch in Python is complex and beyond the scope of a brief answer. However, Python has several libraries, like PyDub, which can be used to convert audio files into MP3 format, but they mostly use an already compiled binary like ffmpeg or libav for the actual encoding and decoding.

Short Time Fourier Transform and DTMF Signaling

The Fourier Transform provides a frequency domain representation of a signal by decomposing it into its sinusoidal components. However, for non-stationary signals where the frequency content changes over time, the Fourier Transform does not provide any information about the time at which a particular frequency component occurs. This limitation can be overcome by using the Short Time Fourier Transform (STFT).

The STFT addresses this by performing a series of Fourier transforms on chunks of data taken from different sections of the signal. Each chunk is multiplied by a window function which is zero-valued outside of the chunk. By varying the size of the window function, one can control the trade-off between time and frequency resolution.

DTMF Signaling

Dual-Tone Multi-Frequency (DTMF) signaling is a method used in touch tone dialing in telecommunication systems. It works by assigning a unique frequency pair consisting of one low-frequency tone and one high-frequency tone to each key on the dial pad.

When a key is pressed, the corresponding frequency pair is generated and sent over the communication channel. At the receiving end, these frequencies are detected to determine which key was pressed.

Example:

Consider a scenario where the number '1' is pressed on the dial pad. This corresponds to a frequency pair of 697 Hz and 1209 Hz. The signal sent is therefore the sum of two sinusoids with these frequencies.

In order to decode this, the receiver could use the STFT to analyse short segments of the received signal. If the window size is chosen appropriately, the STFT will reveal two distinct peaks in the frequency domain corresponding to the frequencies of the DTMF tone.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.signal import spectrogram

# Generate DTMF signal
fs = 8000  # Sample rate
t = np.arange(0, 1, 1/fs)  # Time array
f1, f2 = 697, 1209  # Frequencies for DTMF '1'
sig = np.sin(2*np.pi*f1*t) + np.sin(2*np.pi*f2*t)

# Compute and plot spectrogram
frequencies, times, Sxx = spectrogram(sig, fs)
plt.pcolormesh(times, frequencies, 10*np.log10(Sxx))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()
```

In this Python code, a DTMF signal is generated by summing two sinusoids with frequencies 697 Hz and 1209 Hz. The ‘spectrogram‘ function from the ‘scipy.signal‘ package is used to compute the STFT, which is then plotted to reveal the frequency content of the signal over time.

Spectrograms and DTMF Signaling

A spectrogram is a two-dimensional representation of a signal where the x-axis represents time, the y-axis represents frequency, and the color represents the amplitude of a particular frequency at a particular time. It is essentially a series of Short Time Fourier Transforms (STFTs) and provides a way to analyze both the frequency content and how that frequency content changes over time.

DTMF Signaling

As described in the previous section, Dual-Tone Multi-Frequency (DTMF) signaling is a method used in touch tone dialing in telecommunication systems. It works by assigning a unique frequency pair consisting of one low-frequency tone and one high-frequency tone to each key on the dial pad.

The spectrogram of a DTMF signal reveals the presence of these two frequency components over time. This makes spectrograms a useful tool in analyzing and debugging DTMF systems.

Example: DTMF Spectrogram

Consider the scenario where the number '1' is pressed on the dial pad. This corresponds to a frequency pair of 697 Hz and 1209 Hz. The signal sent is therefore the sum of two sinusoids with these frequencies.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.signal import spectrogram

# Generate DTMF signal
fs = 8000  # Sample rate
t = np.arange(0, 1, 1/fs)  # Time array
f1, f2 = 697, 1209  # Frequencies for DTMF ’1’
sig = np.sin(2*np.pi*f1*t) + np.sin(2*np.pi*f2*t)

# Compute and plot spectrogram
frequencies, times, Sxx = spectrogram(sig, fs, nperseg=1024)
plt.pcolormesh(times, frequencies, 10*np.log10(Sxx), shading=’auto’)
plt.ylabel(’Frequency [Hz]’)
plt.xlabel(’Time [sec]’)
plt.show()
```

In this Python code, a DTMF signal is generated by summing two sinusoids with frequencies 697 Hz and 1209 Hz. The ‘spectrogram‘ function from the ‘scipy.signal‘ package is used to compute the spectrogram, which is then plotted. The spectrogram clearly shows the two frequency components present in the signal when the ’1’ key is pressed.

Key	**Low Frequency (Hz)**	**High Frequency (Hz)**
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
	941	1209
0	941	1336
#	941	1477

Table 1: Frequency pairs for each key in the DTMF system.

Narrowband vs Wideband Spectrograms

Spectrograms provide a 2D representation of a signal where the x-axis represents time, the y-axis represents frequency, and the color or intensity represents the magnitude of the signal at a particular time and frequency.

A spectrogram is created by taking the Fourier Transform of small segments of a larger signal to show how the frequency content of the signal changes over time. The length of the segments used to compute the Fourier Transform determines the trade-off between temporal resolution and frequency resolution in the spectrogram. This leads to the concepts of narrowband and wideband spectrograms.

Trade-Off

In the context of signal processing and specifically time-frequency analysis like spectrograms, the term "trade-off" refers to the compromise that must be made between time resolution and frequency resolution.

Here's a more detailed explanation:

Time resolution is the ability to precisely determine the timing of events in a signal. High time resolution allows you to identify rapid changes in the signal, such as the start and end of a sound in a speech signal. However, high time resolution means that you are looking at a very short window of the signal, which limits your ability to precisely determine the frequencies present in the signal.

Frequency resolution is the ability to precisely determine the frequencies present in a signal. High frequency resolution allows you to separate and identify closely-spaced frequencies in a signal, such as the harmonics in a musical note. However, to achieve high frequency resolution you need to look at a longer window of the signal, which makes it harder to precisely determine when events in the signal occur.

So the "trade-off" is that you can't have both high time resolution and high frequency resolution at the same time - improving one reduces the other. You have to decide which one is more important for your specific application and choose your window length accordingly. This is a fundamental principle in signal processing, known as the uncertainty principle.

Narrowband Spectrograms

Narrowband spectrograms use longer segments of the signal when computing the Fourier Transform, which gives higher frequency resolution but lower time resolution. In other words, narrowband spectrograms can accurately represent the frequency content of the signal, but rapid changes in the signal over time may be blurred or missed entirely. Narrowband spectrograms are often used when the precise frequencies present in the signal are of interest, and the signal is not expected to change rapidly over time.

Wideband Spectrograms

Wideband spectrograms use shorter segments of the signal when computing the Fourier Transform, resulting in higher time resolution but lower frequency resolution. Wideband spectrograms can accurately represent rapid changes in the signal over time, but the frequency content of the signal may be blurred. These are often used when the way in which a signal changes over time is of interest, and the precise frequencies present in the signal are less important.

Choosing Between Narrowband and Wideband

The choice between narrowband and wideband spectrograms depends on the specifics of the signal being analyzed and the goals of the analysis. If precise frequency information is required, a narrowband spectrogram would be appropriate. If the way in which the signal changes over time is of interest, a wideband spectrogram would be a better choice.

Time-Frequency Tiling

In the field of signal processing, it is often useful to analyze how the frequency content of a signal changes over time. One common way to do this is by using a technique known as time-frequency tiling.

0.1 Concept of Time-Frequency Tiling

Time-frequency tiling refers to the division of a time-frequency plane into distinct regions or "tiles", each corresponding to a particular range of times and frequencies. These tiles are used to calculate the Fourier transform of the signal over short, overlapping windows of time, allowing us to observe how the frequency content of the signal changes over time. This technique forms the basis for spectrograms and similar visualizations.

0.2 Time-Frequency Tiling in Speech Analysis

Speech signals are non-stationary signals; their spectral content changes over time. When a person speaks, the frequencies present in the sound they produce can change rapidly. This makes time-frequency tiling an ideal technique for speech analysis.

In speech analysis, the tiles are often chosen to be larger (in the frequency domain) at lower frequencies and smaller at higher frequencies. This reflects the fact that human hearing is more sensitive to changes in pitch at lower frequencies than at higher ones.

Example: Narrowband vs Wideband Spectrograms of Speech

Let's take a look at an example of how the choice of tile size (i.e., the trade-off between time resolution and frequency resolution) can affect the analysis of a speech signal. We will use a recording of a person saying the phrase "Hello, World!".

In the narrowband spectrogram, the tiles are wide in the frequency domain and narrow in the time domain. This gives us a good frequency resolution (we can see individual harmonics of the voice), but poor time resolution (rapid changes in the speech are blurred).

In the wideband spectrogram, the tiles are narrow in the frequency domain and wide in the time domain. This gives us good time resolution (we can see rapid changes in the speech), but poor frequency resolution (the individual harmonics are blurred).

The choice of whether to use a narrowband or wideband spectrogram depends on what aspects of the speech we are interested in. If we want to analyze the pitch of the voice, we would choose a narrowband spectrogram. If we want to analyze the timing of the speech sounds, we would choose a wideband spectrogram.

1 Problem Set - 5

1. Explain in your own words the principle behind the Fourier Transform and how it relates to signal processing.

2. Write the Python code to generate a sinusoidal signal with frequency 440 Hz (the note 'A'), sampled at a rate of 8000 Hz for 2 seconds. Compute and plot its Fourier Transform.

3. Given the DTMF frequency table, write Python code to generate the DTMF signal for the number '5'. Use a sampling rate of 8000 Hz and make the duration of the signal 1 second.

4. Using the signal you generated in the previous problem, compute and plot its Short-Time Fourier Transform (STFT). Can you visually identify the frequencies of the signal in the spectrogram?

5. Suppose a wav file is sampled at a rate of 44100 Hz. What is the Nyquist frequency in this case?

6. Write Python code to read a .wav file, compute its Fourier Transform, and plot the magnitude of the transform in the frequency domain. Do you observe any specific patterns in the frequency content of the signal?

7. Consider a sinusoidal signal of frequency 100 Hz, contaminated with a noise of frequency 2000 Hz. Write Python code to apply a low-pass filter to this signal and filter out the noise. Plot both the original and the filtered signal.

8. Briefly explain the principle of operation of the MP3 compression algorithm and its relation to the Fourier Transform.