Signal Processing Lab Report -9

Team : 10

Aasrith Reddy Vedanaparti - 2023102031

Ritama Sanyal - 2023112027

International Institute of Information Technology Hyderabad

1 Familiar sounds

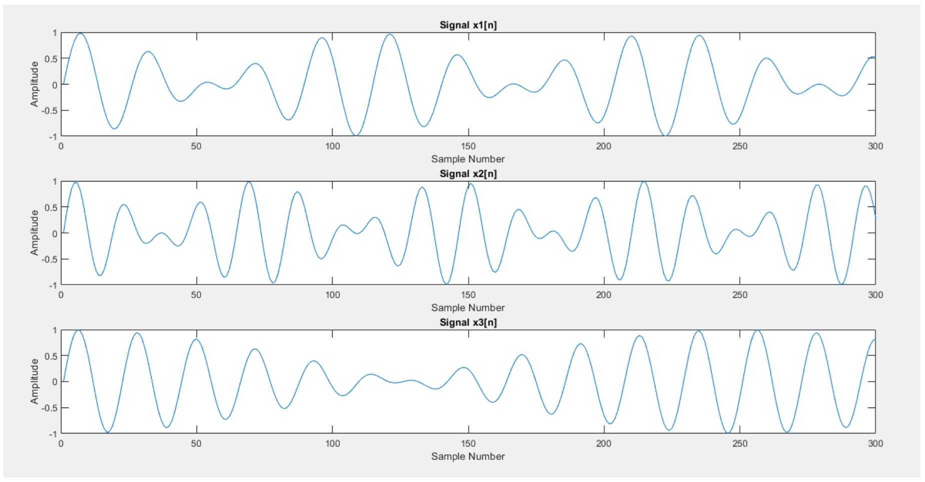

Figure 1.1: Plots of required signals

The sounds are familiar. They are heard during various situations during a phone call.

2 Creating a signal with harmonics

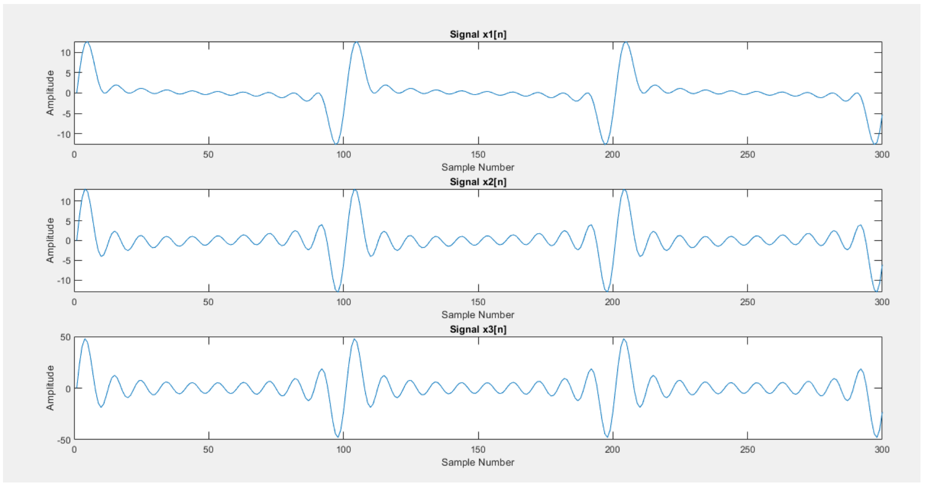

Figure 2.1: Plots of required signals

3 Creating a signal envelope

3.1 Description about ADSR Curve

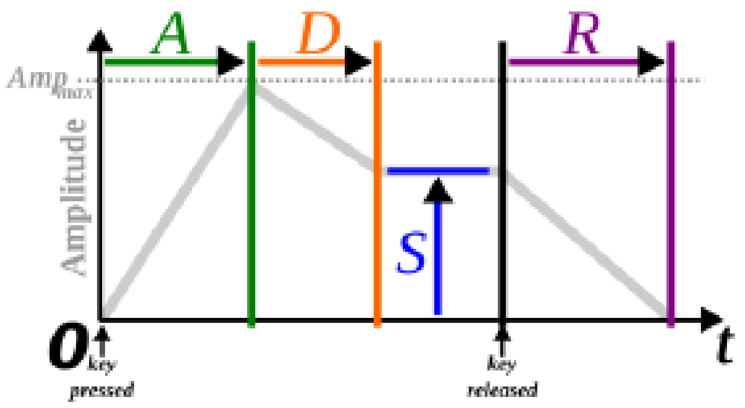

Figure 3.1: ADSR Curve

- **Attack (a)**: This phase represents the time it takes for the amplitude to rise from 0 to 1, typically the initial increase when the note is played.
- **Decay (d)**: After reaching the peak (amplitude = 1), the decay phase gradually lowers the amplitude to a specified sustain level s.
- **Sustain (sd)**: During the sustain phase, the amplitude remains constant at a specified level(s) until the note is about to end.
- **Release (r)**: The release phase occurs when the note is ending, where the amplitude decreases smoothly from the sustain level to 0.

3.2 Envelope Function

Duration of each ADSR segment :

- ***Attack Segment:*** Define a time vector t-attack from 0 to a seconds with increments of 1/fs. It will be $0 : \frac{1}{fs} : a$
- ***Decay Segment:*** Define a time vector t-decay from a to a + d seconds. It will be $a + \frac{1}{fs} : \frac{1}{fs} : a + d$
- ***Sustain Segment:*** Define a time vector t-sustain for the sustain duration sd.It will be $a + \frac{1}{fs} + d : \frac{1}{fs} : a + d + sd$

- ***Release Segment:*** Define a time vector t-release for the release duration r.It will be $a + \frac{1}{fs} + d + sd : \frac{1}{fs} : a + d + sd + r$

Envelope of each ADSR segment :

- ***Attack Segment:*** The envelope for this segment should linearly increase from 0 to 1.
- ***Decay Segment:*** The envelope should decrease linearly from 1 to the sustain level s.
- ***Sustain Segment:*** The envelope remains constant at s.
- ***Release Segment:*** The envelope decreases linearly from s to 0.

3.3 MATLAB Script Observation

When data $a = 0.2; d = 0.2; s = 0.7; sd = 0.4; r = 0.2;$

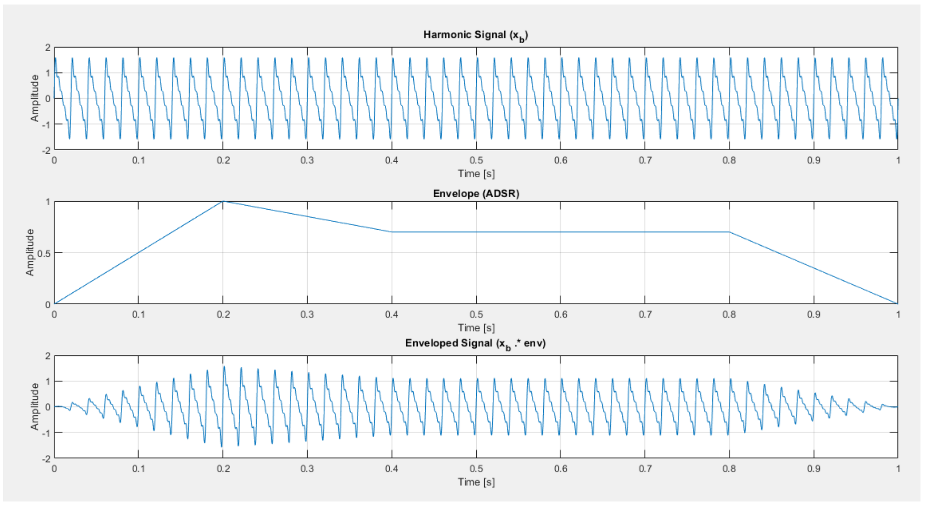

Figure 3.2: Envelope output

When data $a = 0.4; d = 0.2; s = 0.7; sd = 0.2; r = 0.2;$

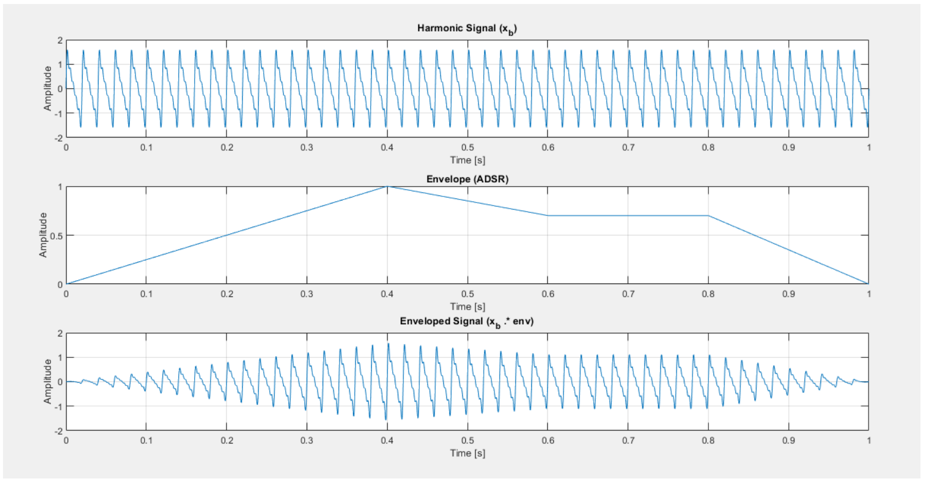

Figure 3.3: Envelope output

When data $a = 0.2; d = 0.4; s = 0.7; sd = 0.2; r = 0.2;$

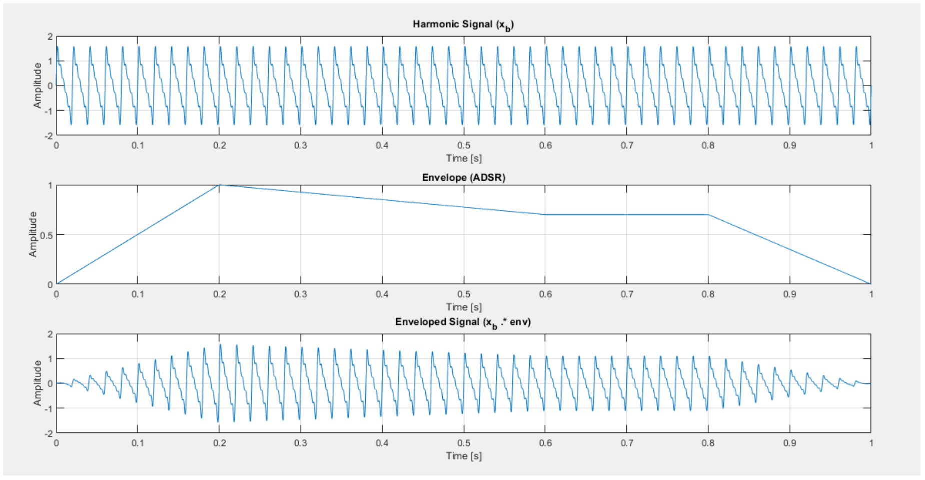

Figure 3.4: Envelope output

When data $a = 0.2; d = 0.4; s = 0.3; sd = 0.2; r = 0.2;$

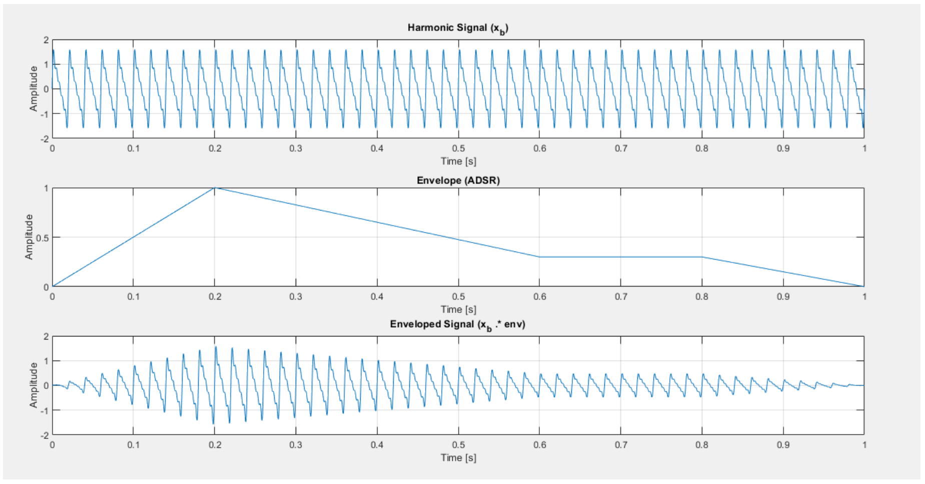

Figure 3.5: Envelope output

When data $a = 0.2; d = 0.2; s = 0.3; sd = 0.2; r = 0.4;$

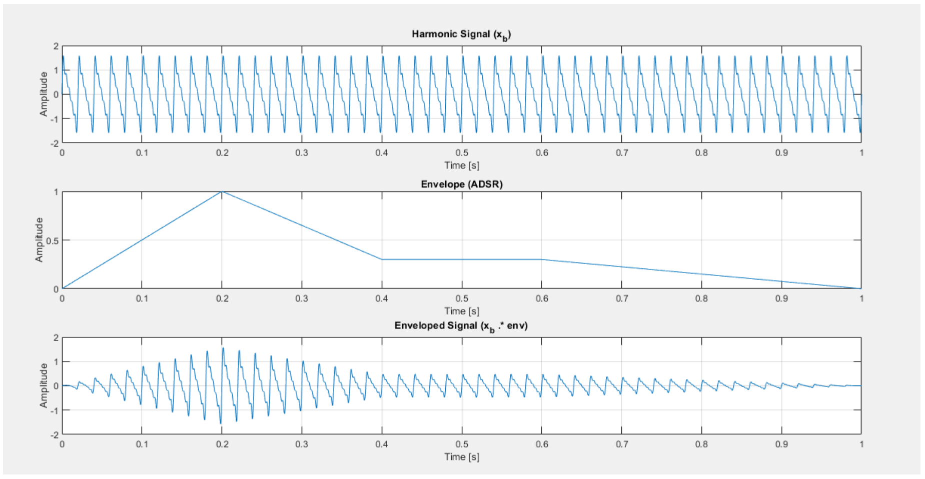

Figure 3.6: Envelope output

- **Attack (a)**:
  - ***Short Attack***: The sound reaches its peak amplitude almost immediately, giving a sharp or percussive onset, like a piano key strike or a snare drum hit.
  - ***Long Attack***: The sound fades in more gradually, creating a smoother, softer start, similar to how a violin or bow instrument builds up.
- **Decay (d)**:

  - ***Short Decay***: The sound quickly drops from the peak amplitude to the sustain level, resulting in a “plucked” feel, common in sounds like a plucked string.
  - ***Long Decay***: The sound holds its peak for a longer period before settling to the sustain level, giving the impression of a more gradual relaxation from the peak.

- **Sustain Level (s)**:
  - ***High Sustain***: The sound maintains a strong amplitude during the sustain phase, giving a more continuous or steady note, typical in instruments like an organ or a flute.
  - ***Low Sustain***: The sound fades more significantly after the initial attack, often used for sounds that decay naturally over time, like a struck guitar string or piano.

- **Sustain Duration (sd)**:
  - ***Short Sustain Duration***: The sound remains at the sustain level for only a brief time, which can make the note sound staccato or choppy, often used in fast-paced music.
  - ***Long Sustain Duration***: The sound stays steady for an extended period, creating a more “legato” effect, commonly found in melodic or ambient sounds.

- **Release (r)**:
  - ***Short Release***: The sound quickly drops to silence once the note ends, creating a clipped or abrupt cutoff, which can suit rhythmic or percussive sounds.
  - ***Long Release***: The sound gently fades out, giving a softer, more gradual ending, ideal for smooth transitions between notes in ambient or melodic music.

4 A simple music synthesizer

4.1 My Synthesizer Function

The goal is to design a function, `my_synthesizer`, that can generate multiple notes in sequence, each with an individual frequency, duration, and an ADSR (Attack, Decay, Sustain, Release) envelope applied to shape the amplitude. This synthesizer will take parameters such as amplitudes, frequencies, phases, ADSR values, durations, and sampling frequency.

- `A`: A vector of length N specifying the amplitudes a_k for each harmonic (assumed the same for all notes).

- `F_notes`: A vector of length M containing the fundamental frequencies of the notes to be played (in Hz).
- `P`: A vector of length N with the phases ϕ_k (assumed the same for all notes).
- `adsr`: A vector with the ADSR parameters (a, d, s, sd, r), which define the envelope.
- `td_notes`: A vector of length M containing the duration of each note (in seconds).
- `fs`: The sampling frequency in Hz.

The function will generate and sequence each note by iterating through `F_notes` and `td_notes`. Here's a breakdown of the key steps inside the function.

1. **Scaling ADSR Parameters**: For each note, scale the ADSR values (a, d, sd, r) so they add up to the duration `td_notes(ii)` of the current note. This ensures that each note has its own duration, while the relative proportions of attack, decay, sustain, and release are maintained.
2. **Generate the Envelope**: Call the `envelope` function (from question 9.3) with the scaled ADSR values to create an ADSR envelope for the note. This function returns a time vector `t_env` and the envelope vector `env`.
3. **Compute the Harmonic Sum for the Note**: Use the `harmonics` function (from question 9.2) to generate a note by summing sinusoids with harmonics based on the fundamental frequency `F_notes(ii)`. Pass the `A` and `P` vectors along with the fundamental frequency `F_notes(ii)`, scaled `td_notes(ii)`, and `fs` to this function. The output `xt` represents the sound wave of the note without amplitude shaping.
4. **Apply the Envelope to the Note**: Multiply the harmonic sum `xt` by the ADSR envelope `env` to shape the amplitude of the note over time, resulting in an enveloped note `xte`.
5. **Concatenate the Notes**: Append the current enveloped note `xte` to the output vector `y`, which will hold the entire synthesized tune as a continuous sequence.

The `my_synthesizer` function combines the principles of harmonic summing (to create musical notes with specified timbres) and ADSR envelopes (to add natural dynamics to each note) to synthesize a sequence of notes. This approach allows for creative experimentation to produce unique tunes or emulate real instrument sounds.

4.2 File 1

- **Sample Rate**: 10,000 Hz
- **Duration**: 11.0011 seconds
- **Mean Amplitude**: -0.88
- **Amplitude Range**: Maximum of 32,767 and Minimum of -32,768 (indicating full dynamic range usage)
- **Dominant Frequency**: 99.99 Hz

4.3 File 2

- **Sample Rate**: 10,000 Hz
- **Duration**: 5.1078 seconds
- **Mean Amplitude**: -0.76
- **Amplitude Range**: Maximum of 32,767 and Minimum of -32,768 (indicating full dynamic range usage)
- **Dominant Frequency**: 83.99 Hz

4.4 File 3

- **Sample Rate**: 10,000 Hz
- **Duration**: 5.2454 seconds
- **Mean Amplitude**: -1.31
- **Amplitude Range**: Maximum of 32,767 and Minimum of -32,768 (indicating full dynamic range usage)
- **Dominant Frequency**: 79.50 Hz

4.5 Observations

1. **Frequency Content**: Both sounds contain low-frequency components, with File 1 centered around 100 Hz and File 2 around 84 Hz. This suggests that each sound is likely a simple harmonic or a combination of low frequencies, possibly corresponding to musical notes or tones.
2. **Amplitude Dynamics**: Both files use the full 16-bit dynamic range, achieving a strong and clear signal. However, the slight negative mean amplitude in both files might indicate a minor offset or asymmetry in the waveform.

3. **Duration and Structure**: File 1 is longer than File 2, potentially indicating a more complex or repeated structure. File 2, being shorter, may represent a simpler or truncated sound pattern.

4. **Frequency Content**: Both files contain low-frequency components, with File 2 centered around 87.65 Hz and File 2 around 79.50 Hz. This suggests that each sound is likely a harmonic or combination of low frequencies, which may correspond to musical notes or tones.

5. **Amplitude Dynamics**: Both files use the full 16-bit dynamic range, achieving a strong signal. The slight negative mean amplitude in both files might indicate a minor offset or asymmetry in the waveform.

6. **Duration and Structure**: File 3 (‘Q3.wav‘) is slightly longer than File 2 (‘q2.wav‘), potentially indicating a more complex or extended structure. File 1, being shorter, may represent a simpler or truncated sound pattern.