Assignment 5

EC5.201 Signal Processing

Digital Music Synthesis
Deadline: Saturday, 20th Sep '25, 11.55PM IST

Instructions:

- All the questions are compulsory
- The questions contain both theory (analytical) and practical (MATLAB coding) parts
- The submission format is as follows:
  - **Theory (on Moodle)** – PDF file containing the handwritten theory assignment solutions
  - **Lab (on GitHub)**
    - **Code** – folder containing all the codes
    - **Images** – folder containing all the .png images
    - **Report** – PDF/text file containing observations for the MATLAB section

    The naming convention for the code and image files is q<qn no.> _ <sub-part no.>.
- **Late submission:** For the theory component, a 10% penalty per day will be applicable (accepted up to at most 3 days after the deadline). No late submissions will be accepted for the lab component.

Question 1: Familiar sounds

Write a function `xn = SineSum(A, F, P, td, fs)` that generates samples of the signal $x(t)$ where,

$$x(t) = \sum_{k=1}^{N} a_k sin(2\pi f_k t + \phi_k)$$

- A-vector of length N consisting of amplitudes a_k
- F-vector of length N consisting of the frequencies f_k (in Hertz)
- P-vector of length N consisting of the phases ϕ_k
- td - duration of the generated signal $x(t)$ in seconds
- fs - sampling frequency for generating samples
- xn - the output signal consisting of samples of $x(t)$.

Note: for faster code avoid writing for-loop over time variable.

Using the above function, write a script for generating the signals below. For each sinusoid below use an amplitude of 0.5 and phase of 0, set the sampling frequency to $f_s = 10$ kHz, and use the matlab command `sound()` to listen to the generated signal.

(a) Generate a signal $x_1[n]$ that is sum of two sinusoids with frequencies 350 Hz and 440 Hz in the time interval [0, 4] seconds. Listen to it.

(b) Generate a signal $x_2[n]$ which is composed of alternating copies of signals b1 and z1 (of 0.5 second duration each) repeated four times where: b1 is the sum of two sinusoids with frequencies 480 Hz and 620 Hz, and z1 is an all zero signal. In matlab, you can simply concatenate two row vectors b1 and z1 to get a longer row vector by writing `[b1, z1]`. Listen to the signal $x_2[n]$.

(c) Generate a signal $x_3[n]$ which is composed of alternating copies of signals b2 and z2 (of 2 second duration each) repeated four times where: b2 is the sum of two sinusoids with frequencies 440 Hz and 480 Hz, and z2 is an all zero signal. Listen to it.

(d) Do the above signals sound familiar?

(e) Plot these three signals in a 3x1 figure (plot the first 300 samples).

Question 2: Creating a signal with harmonics

Many musical instruments' sounds are well-modelled as the sum of harmonically related sinusoids. Each "note" played by a musical instrument (for ex. note of a piano) can be approximated as

$$x(t) = \sum_{k=1}^{N} a_k sin(2\pi k f_0 t + \phi_k)$$

where f_0 is the fundamental frequency of the note and there are N harmonics.

Write a function `xn = harmonics(A, f0, P, td, fs)` which calls the function `SineSum()` from previous part to generate a "note" of frequency fo. The remaining parameters are as before.

Write a script to call the function `harmonics()` for various inputs and listen to the generated signal using the `soundsc()` command (this scales your signal before playing). You can set phase P to be all zeros, sampling frequency of $f_s = 10$ kHz, and "note" duration to be `td` = 1 second. Try the following:

(a) $N = 5$, $f0 = 50$, $a_k = \frac{1}{k}$

(b) $N = 5$, $f0 = 50$, $a_k = exp(-k)$

(c) Repeat above with $N = \{10, 15\}$ and $f0 = \{100, 150, 200\}$

(d) Repeat with other amplitudes such as $a_k = 1 + sin(\frac{\pi k}{N})$, $a_k = log(k)$, $a_k = k$

(e) Plot the signal (first 300 samples) and observe (plot for a fixed value of N and fo, you don't have to plot for all above cases).

(f) Optional: Experiment/design a_k so that it sounds like your favourite instrument!

Question 3: Creating a signal envelope

The ADSR (attack, decay, sustain, release) envelope is used in music synthesizers to model how the amplitude of a note changes over time, see figure above (and read in Wikipedia). This makes the node sound more realistic and less abrupt.

Write a matlab function `[t_env, env] = envelope (a, d, s, sd, r, fs)`, that takes inputs

- a - attack duration in seconds

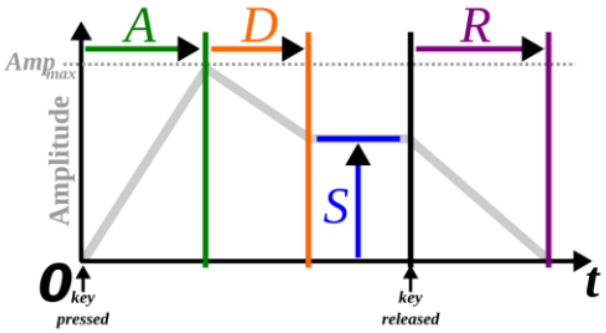


Figure 1: ASDR curve

- d- decay duration in seconds
- s - sustain level in [0,1]
- sd-sustain duration in seconds
- r - release duration in seconds
- fs-sampling frequency in Hz

and returns

- t_env - time vector sampled at fs Hz of length a+d+sd+r seconds
- env - the corresponding ADSR envelope.

You can assume that the durations a, d, sd, and r, will be integer multiples of sampling interval $(1/fs)$. See comments in the solution template for how to interpret the parameters. Use the given template and complete it.

```
function [t_env, env] = envelope (a, d, s, sd, r, fs)
% For each portion of the note, determine the corresponding piece of
% time vector and envelope.
% Attack: amplitude linearly increases from 0 to 1 in 'a' seconds
tattack = 0:1/fs:a;
env = ...;
t_env = tattack;
% Decay: amplitude linearly decreases from 1 to 's' in 'd' seconds
tdecay = (a+1/fs):1/fs:a+d;
t_env = [t_env, tdecay];
env = [env, ...];
% Sustain: amplitude stays at 's' for 'sd' seconds
tsustain = ...;
t_env = [t_env, tsustain];
env = [env, ...];
% Release: amplitude linearly decreases from 's' to 0 in 'r' seconds
trelease = ...;
t_env = [t_env, trelease];
env = [env, ...];
end
```

Make sure that when a+d+sd+r = 1, your code returns t_env & env of length fs.

Write a script to listen to the notes with and without ADSR envelop. Compare using `soundsc(xn, fs)` & `soundsc(xn.*env, fs)` to hear the effects of applying the envelope:

- use fs = 10000
- for xn use one of the notes generated in part 9.2 i.e. `xn = harmonics(...)`
- env is obtained by calling the function `envelope (0.2,0.2,0.7,0.4,0.2,fs)`
- note that you should have `td = a+d+sd+r` for this to work
- change envelope by changing values of a, d, s, sd, r and listen again.

In a 3x1 figure, plot xn, env, and xn.*env in the three panels.

Question 4: A simple music synthesiser

Now you can combine the results of parts 9.2 and 9.3 to create a simple music synthesizer function which constructs M notes in the specified sequence (a tune). Your synthesizer function should take the following inputs

- A-vector of length N consisting of amplitudes a_k (assumed same for all notes)
- F_notes-length M vector of note fundamental frequencies in Hz
- P-vector of length N consisting of the phases ϕ_k (assumed same for all notes)
- adsr-length 5 vector of (a, d, s, sd, r) (assumed same for all notes)
- td_notes-length M vector of note durations in seconds
- fs-sampling frequency in Hz

Since each note can be of different duration, we assume that in the input a+d+sd+r = 1 and for each note we accordingly scale them so that the envelop is of required duration.

Your function should produce an output signal y, so that `sound(y, fs)` produces the specified sequence of notes. Edit the given code to get the output.

```
function yn = my_synthesizer (A, F_notes, P, adsr, td_notes, fs)
% Initialize output as empty
y = [];
% Loop over the notes
for ii = 1:length(F_notes)
% scale a,d,sd,r so that they sum to required note duration
% Compute the time vector and ADSR envelope for this note
[t,env] = envelope(...);
% Compute the sum of harmonics for this note
xt = harmonics(...);
% Modulate the sum of harmonics with the envelope
xte = ...;
% Add the note to the sequence
y = [y,xte];
end
end
```

Write a script which calls the `my_synthesizer()` function with various inputs and listen to the synthesized signal using `soundsc()`. You can set $N = 5$, amplitudes phase P to be all zeros, sampling frequency of $f_s = 10$ kHz. Try the following:

(a) F_notes=50:5:100, all notes of same duration (1 second)

(b) F_notes=100:-10:40, all notes of same duration (1 second)

- $a_k = \frac{1}{k^2}$

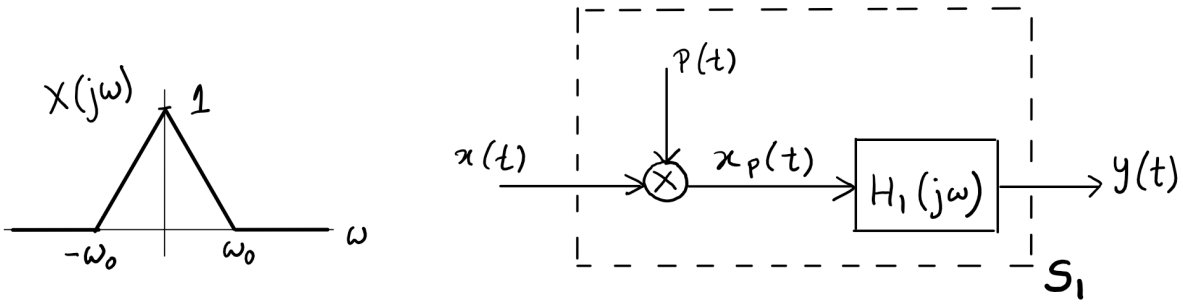
(c) Random tune: $M = 5$; F_notes=50 + 50*rand(1,M) td_notes = 0.5+rand(1,M);

(d) Use different a_k, F_notes and td_notes till you can synthesize/hear a unique tune. Try and see if you can generate your favourite tune!

(e) Use the command `audiowrite()` to save a tune of 9 seconds as wav file and submit.

Question 5

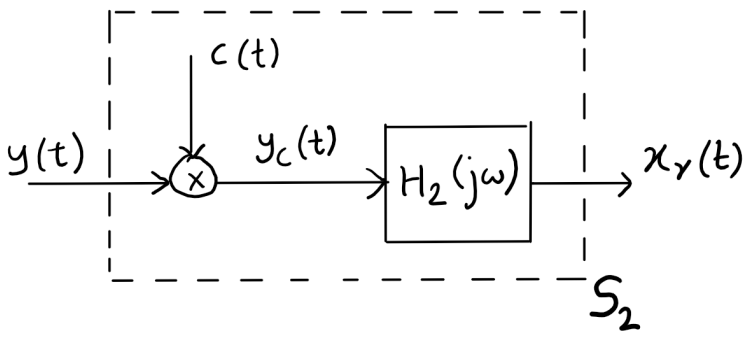
A system S_1 is designed to perform impulse train sampling followed by filtering with $H_1(j\omega)$ as shown below. The input to this system is a *bandlimited* continuous-time signal $x(t)$ with Fourier transform $X(j\omega)$ as shown, $X(j\omega) = 0$ for $|\omega| > \omega_0$.



(a) Write expression for the Fourier transform of the sampled signal $x_p(t) = x(t)p(t)$ and plot it. It is given that impulse train $p(t) = \sum_{-\infty}^{\infty} \delta(t - nT)$ and sampling period $T = \frac{2\pi}{3\omega_0}$.

(b) Due to an error in system design, $H_1(j\omega)$ was chosen to be an ideal high pass filter with cutoff frequency $\frac{3\omega_0}{2}$. Plot the Fourier transform of the output $y(t)$ of this filter.

To correct for the error, a system S_2 is designed with input $y(t)$ as shown below. The system S_2 consists of a multiplication step giving $y_c(t) = y(t)c(t)$ where $c(t) = e^{-j\omega_1 t}$, followed by application of a ideal low pass filter $H_2(j\omega)$ with cutoff frequency $\frac{3\omega_0}{2}$ and gain T.



(c) Give expression for Fourier transform of $y_c(t)$ and plot it.

(d) Find all possible values of frequency ω_1 so that the reconstructed signal $x_r(t)$ is exactly equal to the original signal $x(t)$.

(e) For *any* arbitrary input $x(t)$, is the system S_2 inverse of the system S_1 ? Explain.

Question 6

A discrete-time signal $x[n]$ given by

$$x[n] = \frac{\sin(\omega_0 n)}{\pi n}, \omega_0 \in (0, 2\pi).$$

(a) Show that, $x[n - n_1] * x[n - n_2] = x[n - (n_1 + n_2)]$ for any intergers n_1 and n_2 where the symbol $*$ denotes convolution.

(b) Let $\omega_0 = \frac{\pi}{2}\cdot$ By applying Parseval's relation to $x[n]$, prove an identity of the form

$$\sum_{m=0}^{\infty} \left(\frac{1}{2m+1} \right)^2 = \alpha \pi^2$$

and find the value of the constant α.

Question 7

Solve question 7.39 from the Oppenheim textbook.