Lab 5 – Music synthesizer

5.1 Creating a signal with harmonics

Many musical instruments' sounds are well-modelled as the sum of harmonically related sinusoids. In this part we will build up a signal by adding terms of the form $a_k \sin(2\pi k f_0 t + \phi_k)$, where f_0 is the fundamental frequency and [amp(k), phase(k)] define the amplitude and phase of the k^{th} harmonic, where k = 1, 2, ... N. Let f_s be the sampling frequency.

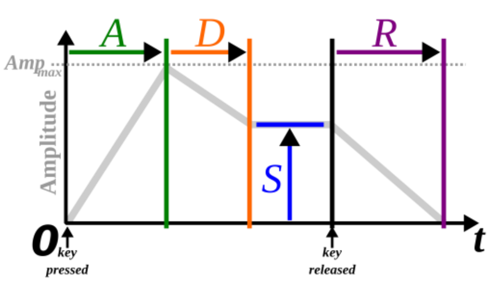
Write a matlab function, `harmonics`, that takes as input

- `time_vec`, a time vector whose samples are 1/fs apart
- `F0`, the fundamental frequency in Hz
- `harmamps`, a length N vector of harmonic amplitudes (the first entry corresponds to the fundamental frequency)
- `harmphase`, an optional length N vector of harmonic phases (if this input argument is omitted, all the phases should be 0)

The function should produce an output y, corresponding to the sum of harmonically related cosines above, which you can listen to using `sound(y,fs)`. You can set $f_s = 48000\ Hz$. Set `time_vec` to generate y of 2 seconds duration. Use the following template and complete your code.

```
function y = harmonics(t, f0, harmamps, harmphase)
% Initialize y to 0
y = zeros(size(t));

% Loop over harmonics, adding weighted versions to y
for i=1:length(harmamps)
    y = y+...
end
% Normalize maximum amplitude to 0.95 so that
% sound(y,fs) doesn't get distorted
y = y/max(y(:))*0.95;
end
```



ADSR curve

5.2 Creating a signal envelope

The ADSR (attack, decay, sustain, release) envelope is used in synthesizers to model how the amplitude of a note changes over time (you can read up about this in Wikipedia).

Write a function, envelope, that takes the positive scalar inputs

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

The function should return

- `time_vec`, a time vector sampled at fs Hz of length a+d+dur+r seconds
- `env`, the corresponding ADSR envelope (see comments in the solution template for how to interpret the parameters).

Thus, if harmonics is your solution to Problem 5.1, you can compute
[t,env] = envelope(fs, ...);
y = harmonics(t, f0, ...);

and then compare `sound(y,fs)` with `sound(y.*env, fs)` to hear the effects of applying the envelope. Use the following template and complete the code.

```
function [t,env] = envelope(fs,a,d,s,dur,r)
% In each phase of the signal, determine the corresponding piece
of time vector and envelope.
% Attack: signal linearly increases from 0 to 1 in a seconds
t = 0:1/fs:a;
e = ...
% Decay: signal linearly decreases from 1 to s in d seconds
tdelay = (a+1/fs):1/fs:a+d;
t = [t, tdelay];
e = [e, ...];
% Sustain: signal stays at s for dur seconds
tsustain = ...
t = [t, tsustain];
e = [e, ...];
% Release: signal linearly decreases from s to 0 in r seconds
trelease = ...
t = [t, trelease];
e = [e, ...];
end
```

5.3 A simple music synthesizer

Now you can combine the results of Problems 5.1 and 5.2 to create a simple music synthesizer. Your synthesizer function should take the inputs

- `notes`, a length *N* vector of note frequencies in Hz
- `durs`, a length *N* vector of note durations in seconds
- `harmamps`, a length M vector of harmonic amplitudes for each note
- `adsr`, a length 4 vector of (attack duration, decay duration, sustain level, release duration)
- `fs`, a sampling frequency in Hz

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

```
function y = synthesizer(notes,durs,harmamps,adsr,fs)
% Initialize output as empty
y = [];
% Loop over the notes
for i=1:length(notes)
    % Compute the time vector and ADSR envelope for this note
    [t,e] = envelope(...);

    % Compute the sum of harmonics for this note
    h = harmonics(...);

    % Modulate the sum of harmonics with the envelope
    n = ...;


    % Add the note to the sequence
    y = [y,n];
end
% Play the sound
sound(y,fs);
end
```