EECS 3451 Lab3

Table of Contents

AUTHORS

- Jonathan Baldwin (212095691)
- Mark Savin (212921128)
- Sarwat Shaheen (214677322)

Some utility variables and functions

```
function y = notefreq(f,n)
%NOTEFREQ get the frequency of a note from it's musical notation
%   f   - base frequency
%   n   - note in musical notation (eg, 'A#')
    nf = containers.Map();
    nf('R')      = 0;
    nf('A')      = 2^(0/12);
    nf('A#')     = 2^(1/12);
    nf('Bb')     = 2^(1/12);
    nf('B')      = 2^(2/12);
    nf('C')      = 2^(3/12);
    nf('C#')     = 2^(4/12);
    nf('Db')     = 2^(4/12);
    nf('D')      = 2^(5/12);
    nf('D#')     = 2^(6/12);
    nf('Eb')     = 2^(6/12);
    nf('E')      = 2^(7/12);
    nf('F')      = 2^(8/12);
    nf('F#')     = 2^(9/12);
    nf('Gb')     = 2^(9/12);
    nf('G')      = 2^(10/12);
```

```
    nf('G#')     = 2^(11/12);
    nf('Ab')     = 2^(11/12);
    y = f*nf(n);
end
```

We store the notes and tempo in a .mat file instead of repeating them in every function.

```
nps   = .5;
notes   = {
    1/8, 'R';
    1/8, 'G';
    1/8, 'G';
    1/8, 'G';
    1/2, 'Eb';
    1/8, 'R';
    1/8, 'F';
    1/8, 'F';
    1/8, 'F';
    1/2, 'D'
    }';
save composition notes nps;
```

P1

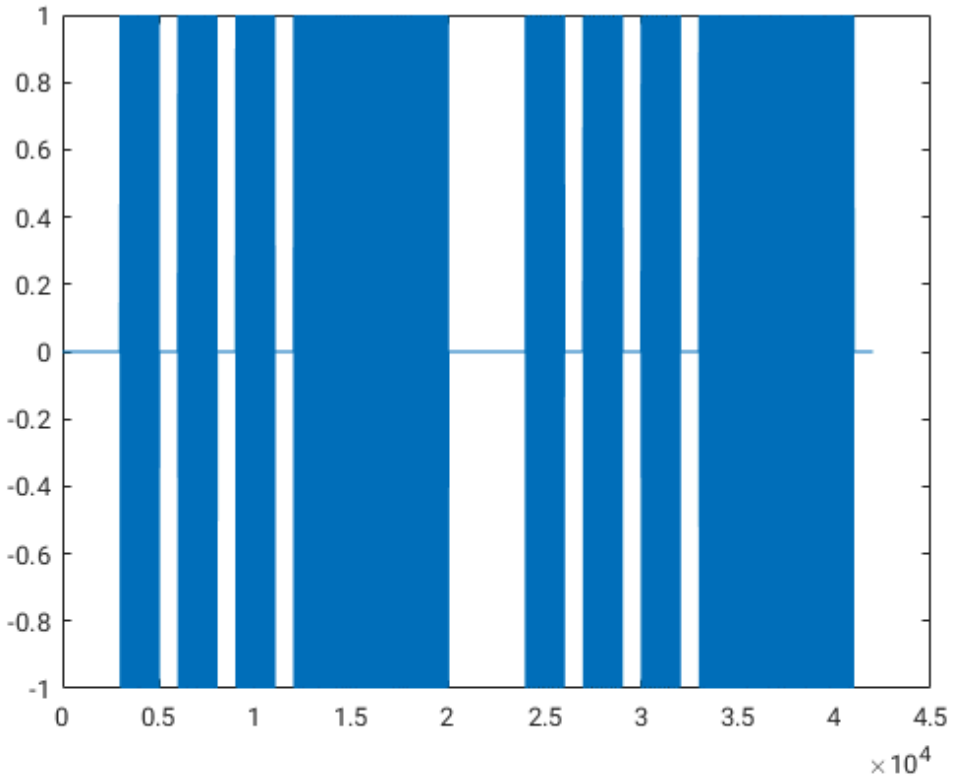
```
function y = make_note(f,fs,d,a)
%MAKE_NOTE make a note
%   f - frequency of the note
%   fs - sample rate
%   d  - length of the note, in seconds
%   a  - amplitude
    t = 0:1/fs:d-1/fs;
    y = [a.*sin(2*pi*f.*t) zeros(1,fs/8)];
end
```

```
function y = create_comp(f,fs,a)
%CREATE_COMP create a composition
%   f    - base frequency (frequency of the A note)
%   fs   - sample rate
%   a    - amplitude
    load composition notes nps;

    y = [];
    for n = notes
        y = [y make_note(notefreq(f, n{2,1}), fs, n{1,1}/nps,a)];
    end
end
```

```
y = create_comp(220,8000,1);
```

```
plot(y);
playsound(y, 8000);
audiowrite('composition.wav',y,8000);
```



P2

Running the composition through half doubles the pitch but also doubles the tempo.

Running the composition through dbl halves the pitch, but also halves the tempo.

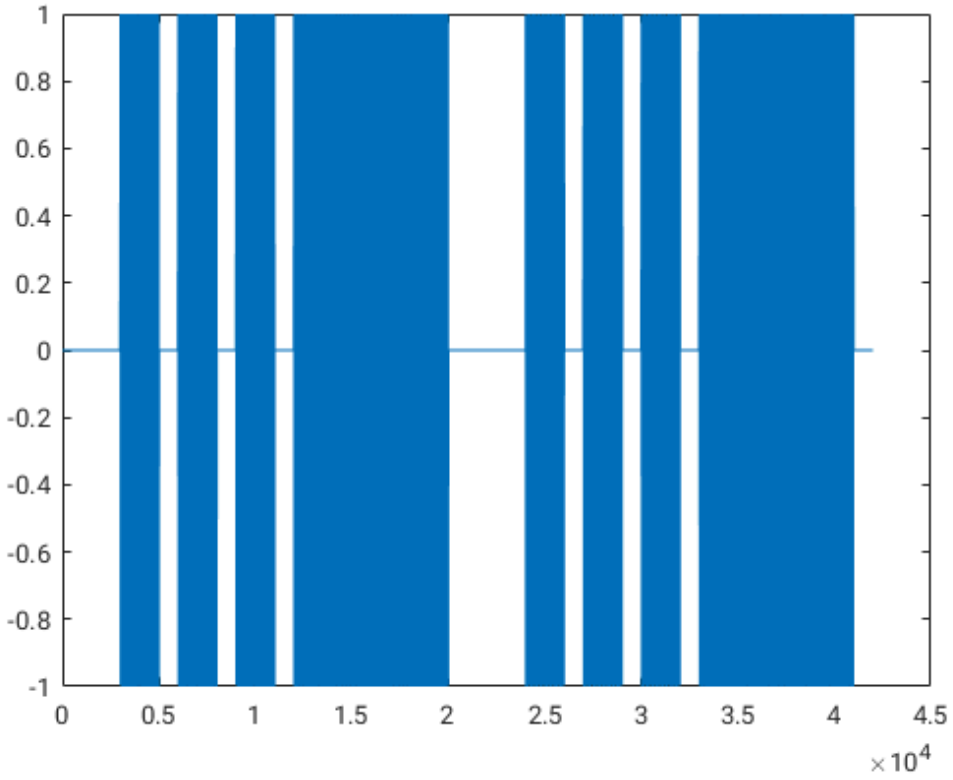
```
function out = half(in)
 out = in(:,1:2:end);
end
```

```
function out = double(in)
 tmp = 1:.5:length(in);
 out = (in(floor(tmp)) + in(ceil(tmp)))/2;
end
```

```
yhalf = half(y);
ydouble = dbl(y);

playsound(yhalf,8000);
```

```
playsound(ydouble,8000);
```



P3

Doubling the amplitude appears to quadruple the volume. The exponential dropoff effect is more noticeable on longer notes. The exponential dropoff effect seems to be more noticeable with a smaller tau/attenuation factor.

```
function y = make_note_exp(f,fs,d,a,tau)
%MAKE_NOTE_EXP make a note with exponential dropoff
%   f    - frequency of the note
%   fs   - sample rate
%   d    - length of the note, in seconds
%   a    - amplitude
%   tau - exponential attenuation factor
    t = 0:1/fs:d-1/fs;
    y = [exp(-t/tau).*a.*sin(2*pi*f.*t) zeros(1,fs/8)];
end
```

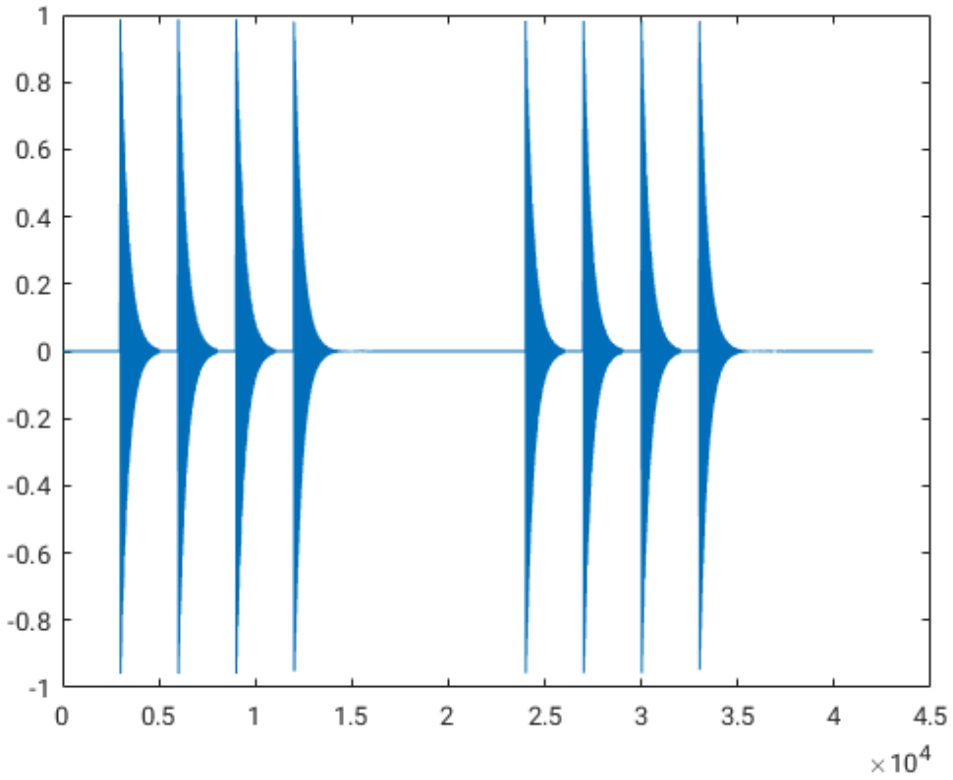
```
function y = create_comp_exp(f,fs,a,tau)
%CREATE_COMP_EXP create a composition with exponential dropoff on
 notes
%   f    - base frequency (frequency of the A note)
```

```
%   fs  - sample rate
%   a   - amplitude
%   tau - exponential attenuation factor
    load composition notes nps;

    y = [];
    for n = notes
        y = [y make_note_exp(notefreq(f,n{2,1}),fs,n{1,1}/nps,a,tau)];
    end
end


y = create_comp_exp(220,8000,1,0.05);
plot(y);

playsound(y, 8000);
```



P4

We double and half the pitch by doubling and halving the f value passed to create_comp_exp, respectively.

```
ydblpitch = create_comp_exp(440,8000,1,0.1);
yhalfpitch = create_comp_exp(110,8000,1,0.1);

playsound(ydblpitch,8000);
playsound(yhalfpitch,8000);
```

P5

We can shift the pitch up or down a half step by multiplying or dividing f by a factor of $2^{1/12}$, respectively.

```
yshiftup   = create_comp_exp(220*2^( 1/12),8000,1,0.1);
yshiftdown = create_comp_exp(220*2^(-1/12),8000,1,0.1);

playsound(yshiftup,8000);
playsound(yshiftdown,8000);
```

P6

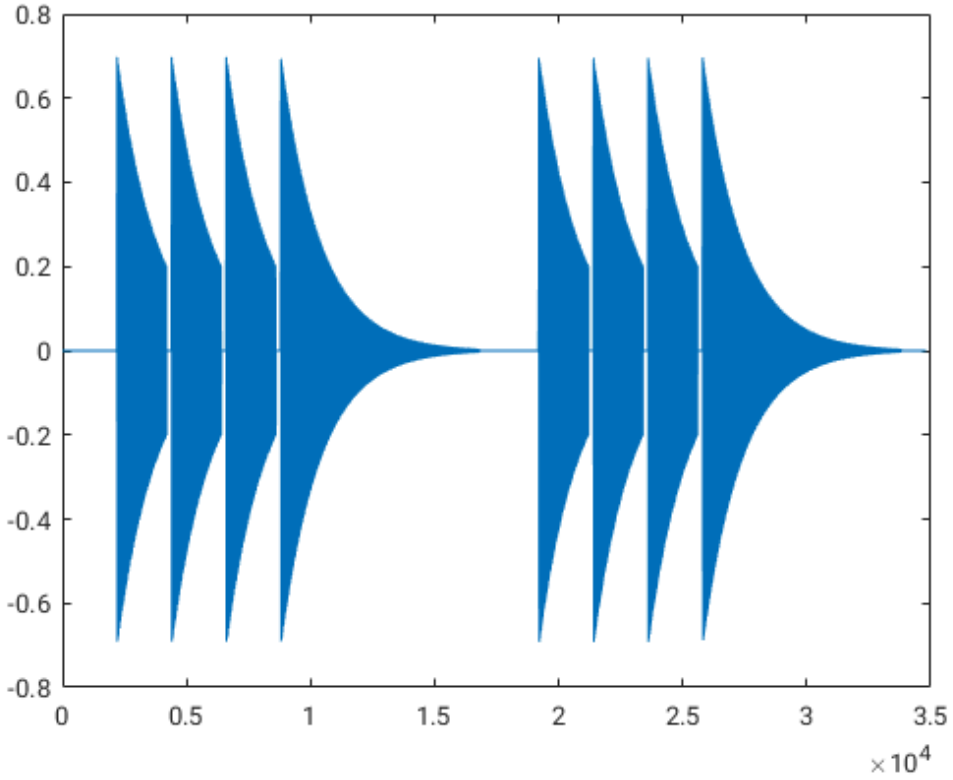
Introducing overlap causes the composition to play smoother and faster, but too much overlap causes notes to bleed together.

```
function y = vec_overlap(v1,v2,t)
%VEC_OVERLAP concatenate two vectors, summing an overlapping area
    if length(v1) < t || length(v2) < t
        y = [v1 v2];
    else
        y = [v1(1:length(v1)-t) v1(length(v1)-t+1:end)+v2(1:t) v2(t
+1:end) ];
    end
end
```

```
function y = create_comp_overlap(f,fs,a,tau,T)
%CREATE_COMP_OVERLAP create a composition, with overlap
%   f   - base frequency (frequency of the A note)
%   fs  - sample rate
%   a   - amplitude
%   tau - exponential attenuation factor
%   T   - amount of overlap to introduce in seconds
    load composition notes nps;

    y = [];
    for n = notes
        y = vec_overlap(y, make_note_exp(notefreq(f,n{2,1}),fs,n{1,1}/
nps,a,tau),T*fs);
    end
end
```

```
y = create_comp_overlap(220,8000,.7,0.2,0.1);
plot(y);
playsound(y,8000);
```

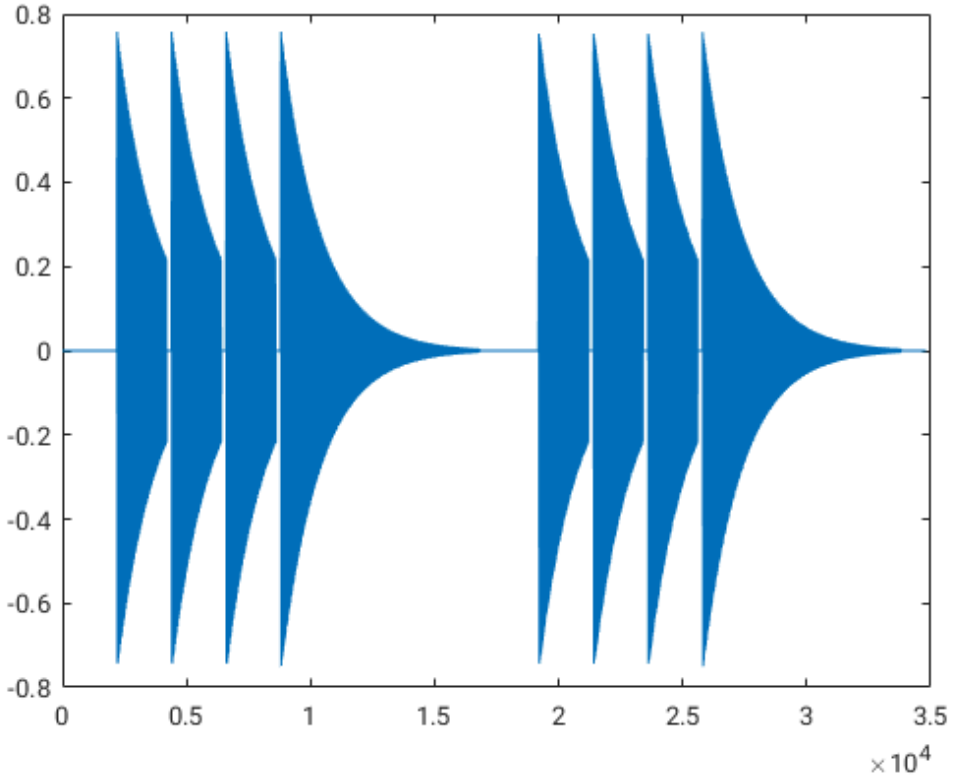


P7

Introducing harmonics causes the composition to have a richer sound, with different harmonics adding different character to the sound.

```
function y = create_comp_harm(f,fs,a,tau,T,h)
%CREATE_COMP_HARM create a composition, with harmonics
%CREATE_COMP_EXP create a composition with exponential dropoff on
 notes
%   f   - base frequency (frequency of the A note)
%   fs  - sample rate
%   a   - amplitude
%   tau - exponential attenuation factor
%   T   - amount of overlap to introduce in seconds
%   h   - vector of harmonics to introduce (eg. [3 5 7])
    y = create_comp_overlap(f,fs,a,tau,T);
    for i = h
        y = y + create_comp_overlap(f*i,fs,a*4^(1-i),tau,T);
    end
end
```

```
y = create_comp_harm(220,8000,.7,0.2,0.1,[2 4 6]);
plot(y);
playsound(y,8000);
```



P8

The delay T controls the delay of the echo. The attenuation factor a controls the volume of the echo added. mu controls how fast the exponentially decaying echo decays; small positive values yield faster decays. w controls the rate at which oscillating echo oscillates, higher values oscillate faster.

```
function y = add_const_echo(x, fs, T, a)
    echo = [zeros(1,fs*T) a.*x];
    y = [x zeros(1,fs*T)] + echo;
end
```

```
function y = add_exp_echo(x, fs, T, a, mu)
    t = 0:1/fs:(length(x)-1)/fs;
    echo = [zeros(1,fs*T) exp(-t/mu).*a.*x];
    y = [x zeros(1,fs*T)] + echo;
end
```

```
function y = add_osc_echo(x, fs, T, a, w)
    t = 0:1/fs:(length(x)-1)/fs;
    echo = [zeros(1,fs*T) cos(w.*t).*a.*x];
    y = [x zeros(1,fs*T)] + echo;
```

```
end


y = create_comp_harm(220,8000,1,0.2,0.1,[2 4 6]);

playsound(y,8000);

playsound(add_const_echo(y, 8000, 0.5, 0.35),8000);
playsound(add_exp_echo(y, 8000, 0.5, 0.35, 1),8000);
playsound(add_osc_echo(y, 8000, 0.1, 1, 5),8000);
```

What we learned

In this lab, we learned how to transform a harsh sinusoidal tone into pleasant sounding notes and compositions through the use of exponential decay and the introduction of harmonics.

We also learned how to apply the DRY (don't repeat yourself) principle to MATLAB code, how MATLAB's save and load functions work, and how to use cell arrays and loops effectively.

Published with MATLAB® R2019a