Tune-Up Tuesday for September 18, 2018

(a) Using the Matlab code below that generates a cosine signal $x_A(t) = \cos(2\,\pi\,f_A\,t)$ with f_A = 440 Hz for 3 seconds at a sampling rate of f_s = 8000 Hz:

```
fs = 8000;          % sampling rate
Ts = 1/fs;          % sampling time
t = 0 : Ts : 3;     % 3 sec
fA = 440;
xA = cos(2*pi*fA*t);
```

add to the above code to create and play an A major chord of A, C# and E

$x(t) = x_A(t) + x_{C\#}(t) + x_E(t)$

where $f_{C\#}$ = 554 Hz and f_E = 660 Hz. Comment on what you hear.

(b) Plot the spectrogram of the A major chord and comment on what you see:

```
spectrogram(x, hamming(1024), 512, 1024, fs, 'yaxis');
```

(c) Copy and paste your code for parts (a) and (b) into the Tune-up #3 page on Canvas.

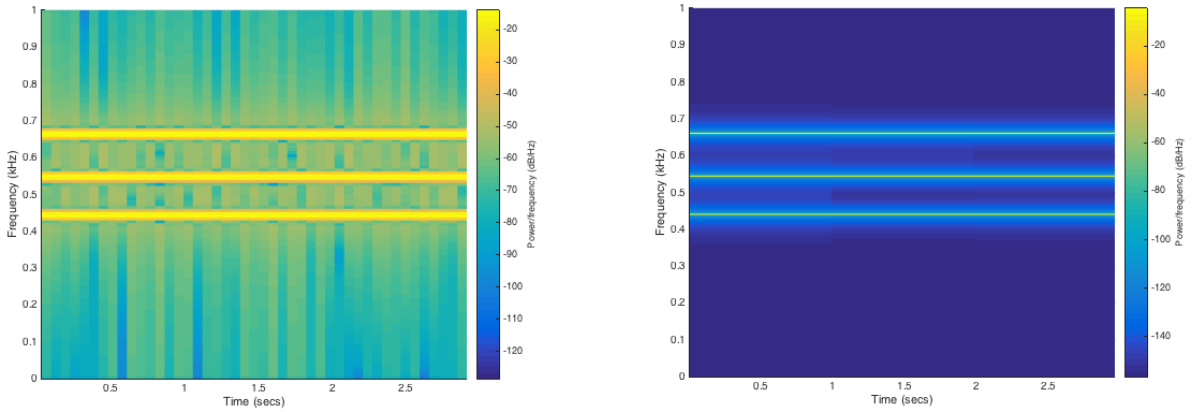
Solution

```
% (a) Generate the notes for an A major chord
fs = 8000;          % sampling rate
Ts = 1/fs;          % sampling time
t = 0 : Ts : 3;     % 3 second duration
fA = 440;
xA = cos(2*pi*fA*t);
fCsharp = 544;    % '#' is not a valid character for a Matlab variable
xCsharp = cos(2*pi*fCsharp*t);
fE = 660;
xE = cos(2*pi*fE*t);
x = xA + xCsharp + xE;
sound(x, fs);
pause(4);
soundsc(x, fs);

% I hear three notes being played.

% The sound command will clip any amplitude value greater than 1 to 1, and
% will clip any amplitude value less than -1 to -1.  This clipping sounds like
% distortion/noise.  The clipping affects 41% of the samples (see the
% Optional part for (a) at the end of this file).

% The soundsc command will make sure that amplitude values are not clipped
% when played out.  The soundsc command will map the range of amplitude
% values [a, b] to the range [-1, 1] before sending the signal to the audio playback
% system.  Playback using the  soundsc command sounds like three principal frequencies
% without much distortion/noise.
```

% The A major chord played as a sum of three note frequencies does not sound
% very pleasant. When a musical instrument plays a note, the frequency for that
% note is played along with harmonics of that note and noise/distortion characteristic
% of the instrument.

% (b) Plot the time-frequency components of x(t) using the spectrogram command.
% Optional: Zoom the frequency axis using ylim.
spectrogram(x, hamming(1024), 512, 1024, fs, 'yaxis');
ylim([0 1]);

% The above code generates the spectrogram on the left.



% The spectrogram contains three vertical lines across the time axis. These correspond
% to the principal frequencies of 440 Hz (A4), 544 Hz (C#) and 660 Hz (E). Each vertical
% line represents a small range of frequencies centered at a principal frequency.

% ***Optional for (b).*** Increasing the number of samples in a segment will increase the
% frequency resolution, and decreasing the shift from one segment to the next will
% give us more time resolution. The code below plots the spectrogram on the right.
figure;
spectrogram(x, hamming(8000), 128, 8000, fs, 'yaxis');
ylim([0 1]);

% ***Optional for (a):*** The Matlab command x > 1 will return a vector that is the same
% length of x with a 1 entry if that component of x is greater than 1 and 0 otherwise.
% We can then sum up the elements of 0s and 1s using the sum command:
sum(x > 1)
% returns the number of samples in x whose amplitudes are greater than 1.
sum(x < -1)
% The following returns the number of samples in x whose amplitudes are less than -1.
sum(x > 1) + sum(x < -1)
% gives 9949 samples out of the 24000 samples of x.