Fall 2024 EE 313 Linear Systems and Signals Prof. Evans

Mini-Project #1: A Musical Illusion

Neev Mehra & Oscar Portillo

1. Introduction

A sum of sinusoids can be used to analyze and synthesize any periodic and non-periodic signals that satisfy Dirilecht's conditions, which specify that the signal is absolutely integrable over all real numbers, and for any range of values, there is a finite number of discontinuities and minima/maxima. More specifically, though, any periodic signal can be defined as a sum of sinusoids with harmonic frequencies, which we accomplish through the technique of Fourier Series analysis learned in class[1.0]. For non-periodic signals that satisfy Dirilecht's conditions, we can apply the Fourier Transform to decompose the non-periodic signal into a sum of sinusoids[1.1].

$$a_k = \frac{1}{T_0}\int_0^{T_0} x(t)e^{-j2\pi kf_0 t}dt$$

Exhibit 1.0: Fourier Series Analysis

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\,dt$$

Exhibit 1.1: Fourier Transform – Frequency Domain to Time Domain

To find the amplitude of each sinusoid, we typically use a Pythagorean formula to sum both the magnitudes of imaginary and real components or take the absolute value of our given magnitude to find our desired components.

$$A = |Y|$$

Exhibit 1.2: Amplitude Calculation

For phase, we apply a similar approach, in that we use the imaginary and real magnitudes with an inverse tangent function to discover our components' phase.

$$\phi = \tan^{-1}\frac{Im(Y)}{Re(Y)}$$

Exhibit 1.3: Phase Calculation

Conversely, we can apply Fourier synthesis to generate signals – whether periodic or non-periodic. This involves adding together the signals to generate your output signal.

$$x(t) = A_0 + \sum_{k=1}^{N} A_k \cos\left(2\pi f_k t + \phi_k\right) = X_0 + \sum_{k=1}^{N} \left\{ \frac{X_k}{2} e^{j2\pi f_k t} + \frac{X_k^*}{2} e^{-j2\pi f_k t} \right\}$$

Exhibit 1.4: Fourier Synthesis

2. Piano Keyboard & Gaussian Forms

2.1 Piano Keyboard

A piano is composed of keys that play different notes. Each note has its own pitch, and pitch is a measure of how "high" or "low" the note sounds. Pitch corresponds to the frequency of the sound wave, where higher frequencies correspond to higher pitches.

On the piano, the keys are divided into different octaves, where each octave has 12 notes, and for any given note, the same note in the next octave is twice the frequency. The piano has 7 white keys (A-G) and 5 black keys for any given octave, and each individual note is $2^{1/12}$ higher in pitch compared to the previous note.

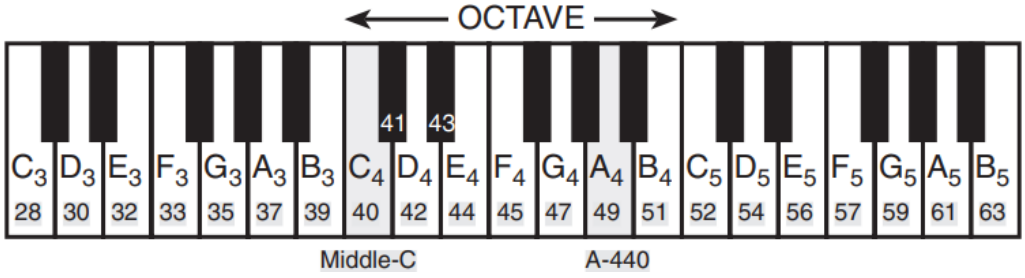


Exhibit 2.0: Piano Keys & Notes

In music, there are different time signatures, where the bottom number indicates which type of note the beat counts and the top indicates how many beats are in a measure, which is a unit of music. For example, if we had music playing at 120 beats per minute in 4/4, each beat corresponds to a quarter note, and there are four quarter notes per measure.

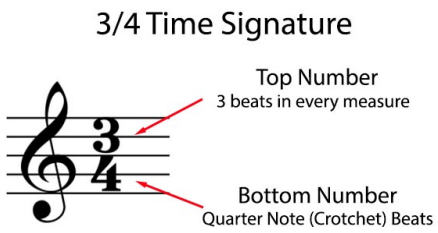


Exhibit 2.1: Time Signatures

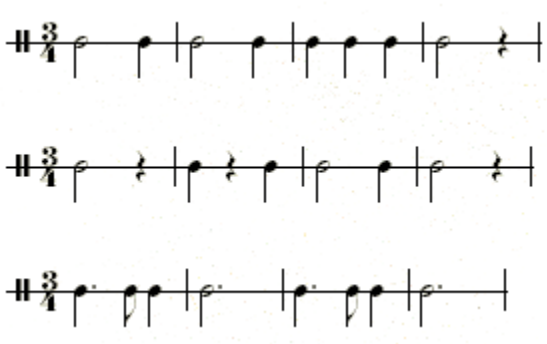


Exhibit 2.2: Notes in Measures

Additionally, the different types of notes – eighth, quarter, half, whole, etc. – correspond to different lengths.

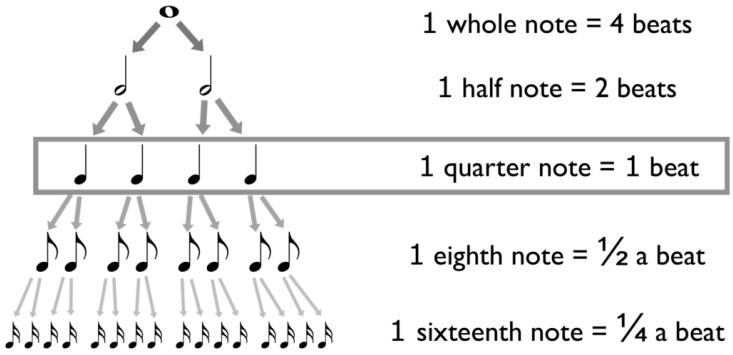


Exhibit 2.3: Different Types of Beats

2.2 Gaussian Forms

The Gaussian is a function with wide applications in engineering, math, science, etc. The formula is provided below where μ is the apex of the Gaussian distribution, ν is the independent variable, and σ is the variance, indicating how wide/spread out the curve is. The Gaussian distribution is commonly recognized as a “bell curve” centered at μ.

$$g(v) = \alpha e^{-(v-\mu)^2/2\sigma^2}$$

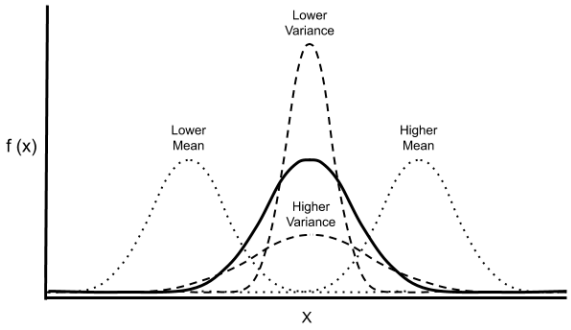
Exhibition 1.5: Gaussian distribution

In our context, we use Gaussian weighting to apply weights to the different harmonics of a given note, where the central frequency (corresponding to the A-G note in the C scale octave that includes A-440) is μ. We do this to appropriately achieve the "limitless" pitch increase characteristic of the Shepard tone.

```
% Array of harmonic frequencies
ff_discr = [f1, f2, f3, f4, fc, f6, f7, f8, f9];
% Generate Gaussian weighting for these frequencies
output = exp( (-(log2(ff_discr) - log2(500)).^2) / (2*sig^2) );
```

3. Warm Up

In the context of Shepard tones and signal processing, we need to apply weighting centered at central frequencies (those on the C scale including A-440) to create the endless pitch increase effect. While there is a commonly used cosine-weighting technique, this assignment requests the implementation of Gaussian weighting; this Gaussian weighting implements the algorithm/formula explained in Section 2.2. Harmonics nearer to the central frequency for a given note are weighted more heavily while those further from it receive less weighting, as is computed by the Gaussian weighting formula (e.g. the A's 4 octaves above or below A-440). The Gaussian, also known as the normal distribution curve, is recognized as a bell curve.



Exhibition 3.0: Gaussian weighting graph

3.1 Gaussian Weighting

For Section 3.1 of the lab, we synthesized the Gaussian using its formula, and we plotted it with a linear and logarithmic frequency scale. When plotted with the logarithmic scale, the Gaussian

reflected the desired bell shape, as its frequency axis was logarithmic, which matched how we defined the Gaussian logarithmically. When plotted with the linear scale, the Gaussian was distorted and concentrated around the central frequency in a non-bell shape.

3.2 2 Synthesize Octaves with Gaussian Weighting

For Section 3.2 of the lab, we implemented the Gaussian weighting on the 5 harmonic notes of C with center at 440Hz. We automated this process of generating the weighted sums of the harmonic sinusoids with the following algorithm that takes the central note frequency and variance as arguments (though there are small variations in implementation for 3.2b and 4.2 based on inputs and desired outputs).

```
function [weighted_signal] = MusicalWeightingC(f_c, sig)
% Define harmonic frequencies
f5 = f_c*2;
f6 = f5*2;
f3 = f_c/2;
f2 = f3/2;
fs = 8000;
fcenter = 440;

% Array of harmonic frequencies
ff_c = [f2, f3, f_c, f5, f6];

% Generate Gaussian weighting for these frequencies
output = exp( (-(log2(ff_c) - log2(fcenter)).^2) / (2*sig^2) );

% Define a time vector for constructing the signal (1 second duration)
t = 0:1/fs:2;

% Initialize the weighted signal
weighted_signal = zeros(size(t));

% Sum the weighted cosine terms for each frequency over time
for i = 1:length(ff_c)
   weighted_signal = weighted_signal + output(i) * cos(2*pi*ff_c(i)*t);
end
end
```

This same automation was implemented with slight tweaks to account for more harmonics in our Shepard Scale algorithm, as defined in Section 4.

4. A Musical Illusion

The Shepard Scale is a musical illusion that plays a repeated scale of weighted harmonics, which ends up sounding like a never-ending rise in pitch.

For our frequency, we found the frequencies for all of the notes in the C scale (and later for all the chromatic notes in this octave), and we doubled or halved them accordingly in the following fashion:

```
function [weighted_signal] = MusicalWeightingDiscrete(fc, sig)
% Define harmonic frequencies
f6 = fc*2;
f7 = f6*2;
f8 = f7*2;
f9 = f8*2;
f4 = fc/2;
f3 = f4/2;
f2 = f3/2;
f1 = f2/2;
```

For amplitude, we extracted the amplitudes for each of the harmonics by applying Gaussian weighting, and then, we performed element-wise multiplication on the harmonic sinusoids so that it was weighted appropriately. All of these sinusoids for each note were added together.

```
% Define sampling frequency
fs = 22050;

% Array of harmonic frequencies
ff_discr = [f1, f2, f3, f4, fc, f6, f7, f8, f9];

% Generate Gaussian weighting for harmonics
output = exp( (-(log2(ff_discr) - log2(500)).^2) / (2*sig^2) );

% Define a time vector for signal (1s duration)
t = 0:1/fs:1;

% Initialize the weighted signal
weighted_signal = zeros(size(t));

% Sum weighted cosine terms for each frequency over time
for i = 1:length(ff_discr)
   weighted_signal = weighted_signal + output(i) * cos(2*pi*ff_discr(i)*t);
end
```

Our phase for the harmonic sinusoids was 0, as the illusion of the Shepard tone is due to the transition between harmonic sinusoids (sinusoids separated by octaves), leading to the perception

of constantly increasing pitch. As such, managing amplitude – which is weighted – and frequency is critical, but phase isn't strictly necessary. Using this phase value of 0 generated an appropriate illusion.

Our code and answers for all of the parts within Section 4 have been added to the Appendix, and the plots have been attached in a separate pdf.

5. Conclusion

Our project creating Shepard's Tone heavily relies on the underlying principles of sound synthesis, Fourier series, and Gaussian weighting.

1. Fourier Series
    - We use Fourier series to generate the harmonic components of the C-scale which is a practical application of the sinusoids. When summing these different components, we effectively synthesize the sound of a piano playing the C-scale.
2. Gaussian Weighting
    - The application of Gaussian weighting successfully creates the illusion of the continuously ascending pitch, which is characteristic of the Shepard Tone. This function makes the frequencies around the center louder while suppressing the outliers, crafting our tone.
3. Piano Notes
    - The project digitally manipulated the C-scale on a piano, aligning with the visual representation mentioned earlier as the foundation of the Shepard Tone.
4. Musical Illusion
    - By creating the Shepard Tone, the project shows how the human brain can be tricked into hearing patterns that aren't physically present in the generated sound, a testament to the success of synthesizing Gaussian-weighted notes.

Bibliography

1. “Fourier Transform – Representation and Condition for Existence.” *Tutorialspoint.com*, 2021,www.tutorialspoint.com/fourier-transform-representation-and-condition-for-existence. Accessed 23 Sept. 2024.
2. AI. “The Science of Machine Learning & AI.” *The Science of Machine Learning & AI*, 2014, www.ml-science.com/normal-distribution. Accessed 23 Sept. 2024.
3. Deutsch, Diana. “Some Musical Illusions Are Discovered.” *Oxford University Press EBooks*, Oxford University Press, June 2019, pp. 24–45, https://doi.org/10.1093/oso/9780190206833.003.0003. Accessed 23 Sept. 2024.
4. Titeux, Nicolas. “The Shepard Tone: An Audio Illusion | Nicolas Titeux.” *Nicolas TITEUX, Sound Designer, Sound Mixer and Composer*, 15 Nov. 2020, www.nicolastiteux.com/en/blog/shepard-and-risset-audio-illusions/. Accessed 23 Sept. 2024.
5. Singh, Meghna, et al. “Gaussian and Laplacian of Gaussian Weighting Functions for Robust Feature Based Tracking.” *Pattern Recognition Letters*, vol. 26, no. 13, Elsevier BV, Oct. 2005, pp. 1995–2005, https://doi.org/10.1016/j.patrec.2005.03.015. Accessed 23 Sept. 2024.
6. “Using the Visual Piano Keyboard | Soundslice Help | Soundslice.” *Soundslice*, 2024, www.soundslice.com/help/en/player/advanced/20/visual-keyboard/. Accessed 23 Sept. 2024.
7. E, Matt. “How to Read Music - Part 2: How Rhythm Really Works | School of Composition.” *School of Composition*, 22 Sept. 2017, www.schoolofcomposition.com/music-rhythm/. Accessed 23 Sept. 2024.
8. “2.6: Introduction to Subdivisions in Simple Meters.” *Humanities LibreTexts*, 23 Jan. 2020, human.libretexts.org/Bookshelves/Music/Music_Theory/Music_Fundamentals_%28Ewell_and_Schmidt-Jones%29/02%3A_Rhythm_and_Meter/2.06%3A_Introduction_to_Subdivisions_in_Simple_Meters. Accessed 23 Sept. 2024.
9. Ben. “Time Signatures - Music Theory Academy - What Is a Time Signature?” *Music Theory Academy*, 18 Jan. 2013, www.musictheoryacademy.com/how-to-read-sheet-music/time-signatures/. Accessed 23 Sept. 2024.
10. Fourier Series Lecture
11. CT Fourier Transform
12. Periodic Signals

13. Assignment Doc
14.

Code Appendix Starts Here:

```
function [weighted_signal] = MusicalWeightingDiscrete(fc, sig)

% Define harmonic frequencies

f6 = fc*2;

f7 = f6*2;

f8 = f7*2;

f9 = f8*2;

f4 = fc/2;

f3 = f4/2;

f2 = f3/2;

f1 = f2/2;

fs = 22050;

% Array of harmonic frequencies

ff_discr = [f1, f2, f3, f4, fc, f6, f7, f8, f9];

% Generate Gaussian weighting for these frequencies

output = exp( (-(log2(ff_discr) - log2(500)).^2) / (2*sig^2) );

% Define a time vector for constructing the signal (1 second duration)

t = 0:1/fs:1;

% Initialize the weighted signal

weighted_signal = zeros(size(t));

% Sum the weighted cosine terms for each frequency over time

for i = 1:length(ff_discr)

   weighted_signal = weighted_signal + output(i) * cos(2*pi*ff_discr(i)*t);

end
```

```
end


function [output, sampling_freq] = NoteHarmonics(base_freq)

% Harmonics

f6 = base_freq*2;

f7 = f6*2;

f8 = f7*2;

f9 = f8*2;

f4 = base_freq/2;

f3 = f4/2;

f2 = f3/2;

f1 = f2/2;

% Sampling Info

sampling_freq = f9 * 10;

Ts = 1/sampling_freq;

% function output

t = [0 : Ts : 2];

output = cos(2*pi*f1*t) + cos(2*pi*f2*t) + cos(2*pi*f3*t) + cos(2*pi*f4*t) +
cos(2*pi*base_freq*t) + cos(2*pi*f6*t) + cos(2*pi*f7*t) + cos(2*pi*f8*t)  +
cos(2*pi*f9*t);



function [output, ff] = MusicalWeighting(fc, sig)

ff = linspace(log2(fc/16), log2(fc*16), 100);

output = exp( (-(ff - log2(fc)).^2) / (2*sig^2) );

end



function [weighted_signal] = MusicalWeightingC(f_c, sig)
```

```
% Define harmonic frequencies

f5 = f_c*2;

f6 = f5*2;

f3 = f_c/2;

f2 = f3/2;

fs = 8000;

fcenter = 440;

% Array of harmonic frequencies

ff_c = [f2, f3, f_c, f5, f6];

% Generate Gaussian weighting for these frequencies

output = exp( (-(log2(ff_c) - log2(fcenter)).^2) / (2*sig^2) );

% Define a time vector for constructing the signal (1 second duration)

t = 0:1/fs:2;

% Initialize the weighted signal

weighted_signal = zeros(size(t));

% Sum the weighted cosine terms for each frequency over time

for i = 1:length(ff_c)

   weighted_signal = weighted_signal + output(i) * cos(2*pi*ff_c(i)*t);

end

end
```

```
% Tune-Up #3
```

```
% Copy this file into a Matlab script window, add your code and answers to the

% questions as Matlab comments, hit "Publish", and upload the resulting PDF
file

% to this page for the tune-up assignment.  Please do not submit a link to a
file

% but instead upload the file itself.   Late penalty: 2 points per minute late.

% Shepard Scale Demo

% https://www.illusionsindex.org/i/shepard-scale-illusionsLinks to an external
site.

% Part A.  We will answer the question in Section 2 of mini-project #1 on

% Shepard Scale Synthesis.

%
https://users.ece.utexas.edu/~bevans/courses/signals/homework/fall2024/miniproj
ect1.pdfLinks to an external site.

% Write a few lines of MATLAB code to make a plot of

% 10 exp( -(v-1)^2 / (2 (3)^2) over the range -10 <= v <= 10.

% This is g(v) where alpha = 10, mu = 1, and sigma=3.

% This question is in Section 2.2 of the lab assignment

% https://dspfirst.gatech.edu/chapters/DSP1st2eLabs/ShepardScaleLab.pdfLinks to
an external site.

fs = 8000;

Ts = 1/fs;

v = -10 : Ts : 10;

alpha = 10;

mu = 1;

sigma = 3;

g = alpha * exp( -(v - mu).^2 / (2 * (sigma^2)) );

figure;

plot(v, g);
```

```
ylim( [-1, alpha+1] );

title('Class Gaussian Plot');

xlabel('v');

ylabel('Gaussian g(v)');

grid on;

% Part B. We will answer the questions in Section 3.1 of mini-project #1 on

% Shepard Scale Synthesis.

%
https://users.ece.utexas.edu/~bevans/courses/signals/homework/fall2024/miniproj
ect1.pdfLinks to an external site.

ff_log = 2.^(log2(55): 1/12 : log2(1760));

ff_normal = 55: 2 : 1760;

fc = 440;

sig = 1; % width equal to one according to log scale (each octave is 2x prev)

% 3.1 part c

% Creating Frequency Weighting Objects

g_1 = FrequencyWeighting(fc, sig, ff_log);

g_2 = FrequencyWeighting(fc, sig, ff_normal);

g_3 = FrequencyWeighting(fc, sig, ff_normal);

figure;

semilogx(ff_log, g_1);

xlabel('Frequency (Hz)');

ylabel('Gaussian Weighting');

title('Gaussian centered at 440Hz w/ logarithmic inputs & semilogx');

% 3.1 part d, the Gaussian appears distorted because the plot command is

% on a linear basis regarding frequency. However, the Gaussian function

% utilizes a logarithmic scale and since the bell shape is compressed
```

```
% towards the center, distortion is created.

%However, when you use semilogx, the x-axis becomes logarithmic which

%aligns with how the Gaussian function is defined. This creates the desired
bell

%shape because the x-axis has proper frequency spacing by Gaussian

%distribution.

figure;

plot(ff_normal, g_2)

xlabel('Frequency (Hz)');

ylabel('Gaussian Weighting');

title('Gaussian centered at 440Hz w/ normal inputs & no semilogx');

figure;

semilogx(ff_normal, g_3);

xlabel('Frequency (Hz)');

ylabel('Gaussian Weighting');

title('Gaussian centered at 440Hz w/ normal inputs & semilogx');




% Q3.2

% Gaussian weighting function & parameters

sig_c = 1;

fs_c = 8000;

fcenterc = 261.63;

weighted_c = MusicalWeightingC(fcenterc, sig_c); % Gaussian function

% Spectrogram parameters

window_c = hamming(2048);
```

```
noverlap_c = 1024;

nfft_c = 2048;

figure;

spectrogram(weighted_c, window_c, noverlap_c, nfft_c, fs_c, 'yaxis'); %
Plotting spectrogram

ylim([0 1.2]);
```

Appendix Part 4: Code

```
% Q4a - Synthesize a C major scale

% Note frequencies derived from
https://www.intmath.com/trigonometric-graphs/music.php

[C261_63, fs261_63] = NoteHarmonics(261.63);

[D293_66, fs293_66] = NoteHarmonics(293.66);

[E329_63, fs329_63] = NoteHarmonics(329.63);

[F349_23, fs349_23] = NoteHarmonics(349.23);

[G392, fs392] = NoteHarmonics(392);

[A440, fs440] = NoteHarmonics(440);

[B493_88, fs493_88] = NoteHarmonics(493.88);

[C523_25, fs523_25] = NoteHarmonics(523.25);

% Q4b - Play the C-Major scale five times

for i = 1:5

sound(C261_63, fs261_63);
```

```
pause(3);

sound(D293_66, fs293_66);

pause(3);

sound(E329_63, fs329_63);

pause(3);

sound(F349_23, fs349_23);

pause(3);

sound(G392, fs392);

pause(3);

sound(A440, fs440);

pause(3);

sound(B493_88, fs493_88);

pause(3);

sound(C523_25, fs523_25);

pause(3);

end

sigma = 2;

% Q4c - Introduce amplitude weighting using a Gaussian form

[GC261_63, logf261_63] = MusicalWeighting(261.63, sigma); % Gaussian notes so
log2

[GD293_66, logf293_66] = MusicalWeighting(293.6, sigma);

[GE329_63, logf329_63] = MusicalWeighting(329.63, sigma);

[GF349_23, logf349_23] = MusicalWeighting(349.23, sigma);

[GG392, logf392] = MusicalWeighting(392, sigma);

[GA440, logf440] = MusicalWeighting(440, sigma);

[GB493_88, logf493_88] = MusicalWeighting(493.88, sigma);
```

```
[GC523_25, logf523_25] = MusicalWeighting(523.25, sigma);

figure;

plot(logf261_63, GC261_63);

xlabel('Time (s)');

ylabel('Frequency (Hz)');

title('Gaussian Note Frequency vs. Time');

figure;

plot(logf293_66, GD293_66);

xlabel('Time (s)');

ylabel('Frequency (Hz)');

title('Gaussian Note Frequency vs. Time');

figure;

plot(logf329_63, GE329_63);

xlabel('Time (s)');

ylabel('Frequency (Hz)');

title('Gaussian Note Frequency vs. Time');

figure;

plot(logf349_23, GF349_23);

xlabel('Time (s)');

ylabel('Frequency (Hz)');

title('Gaussian Note Frequency vs. Time');

figure;

plot(logf392, GG392);

xlabel('Time (s)');

ylabel('Frequency (Hz)');

title('Gaussian Note Frequency vs. Time');
```

```
figure;
plot(logf440, GA440);
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Gaussian Note Frequency vs. Time');
figure;
plot(logf493_88, GB493_88);
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Gaussian Note Frequency vs. Time');
figure;
plot(logf523_25, GC523_25);
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Gaussian Note Frequency vs. Time');
% Q4d - Produce the illusion by using the Gaussian weight function
sig = 0.75;
Fs1 = 22050;
[DC261_63] = MusicalWeightingDiscrete(261.63, sig);
[DDf271_18] = MusicalWeightingDiscrete(271.18, sig);
[DD293_66] = MusicalWeightingDiscrete(293.66, sig);
[DEf311_13] = MusicalWeightingDiscrete(311.13, sig);
[DE329_63] = MusicalWeightingDiscrete(329.63, sig);
[DF349_23] = MusicalWeightingDiscrete(349.23, sig);
[DGf369_99]     = MusicalWeightingDiscrete(369.99, sig);
[DG392]     = MusicalWeightingDiscrete(392, sig);
```

```
[DAb415_30]     = MusicalWeightingDiscrete(415.30, sig);

[DA440]     = MusicalWeightingDiscrete(440, sig);

[DBb466_16]  = MusicalWeightingDiscrete(466.16, sig);

[DB493_88]  = MusicalWeightingDiscrete(493.88, sig);

silVector = zeros(1, round(Fs1 * 0.4));

audiovector = [DC261_63, silVector, DD293_66, silVector, DE329_63, silVector,
DF349_23, silVector, DG392, silVector, DA440, silVector, DB493_88, silVector];

repeatedAudio = repmat(audiovector, 1, 3);

soundsc(repeatedAudio, Fs1);

% Q4e - Make a spectrogram of playing the scale three times, there is a

% clear illusion when observing the spectrogram. You can see that the

% frequencies increase as the scale progresses, and that there is an

% overlap between the higher pitched notes vs. the lower ones, this overlap

% is producing the effect because the listener perceives the overlap as a

% continuous sound climbing the frequency ladder.

window = hamming(1024);

noverlap = 512;

nfft = 1024;

spectrogram(repeatedAudio, window, noverlap, nfft, Fs1, 'yaxis');

% Q4f - Variations: Play every note within the octave, the illusion does sound

% better because it has more frequencies in between, reducing the step

% makes the illusion more convincing because it's not as much as an

% auditory jump so the listener perceives the differences as marginal.

audiovector1 = [DC261_63, silVector, DDf271_18, silVector, DD293_66, silVector,
DEf311_13, silVector, DE329_63, silVector, DF349_23, silVector, DGf369_99,
silVector, DG392, silVector, DAb415_30, silVector, DA440, silVector, DBb466_16,
silVector, DB493_88, silVector];

repeatedAudio1 = repmat(audiovector1, 1, 3);
```

```
soundsc(repeatedAudio1, Fs1);

spectrogram(repeatedAudio1, window, noverlap, nfft, Fs1, 'yaxis');

filename = 'mywav.wav';

normrepeatedaudio1 = repeatedAudio1 / (max(abs(repeatedAudio1)));

audiowrite(filename, normrepeatedaudio1, Fs1);
```