Fall 2024 EE 313 Linear Systems and Signals Prof. Evans

Mini-Project #1: Shepard Music Scale

Elyes Balti, Brian L. Evans, and Gordon Ta

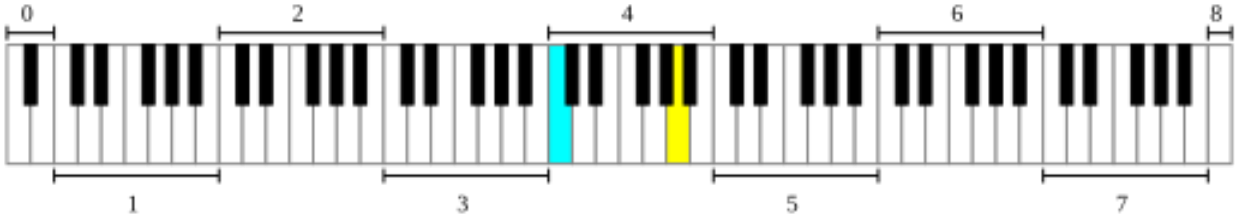
December 1, 2024, Version 2.0

1.0 Introduction (*3 points*)

This mini-project uses a sum of sinusoids to synthesize and play a Shepard Scale [1]. A Shepard Scale is an auditory illusion that sounds like the scale is always increasing in frequency. In the sum of sinusoids, we will use frequencies that are separated by octaves and amplitudes of different weights to achieve the auditory illusion.

2.0 Pre-Lab (6 *points*)

On a piano keyboard, the ratio between the frequencies of successive keys is $2^{1/12}$ Hz. 12 keys are contained in an octave, and the notes in one octave are twice the frequency of the notes in the previous octave. Here are the notes and octaves on an 88-key piano keyboard: [2]

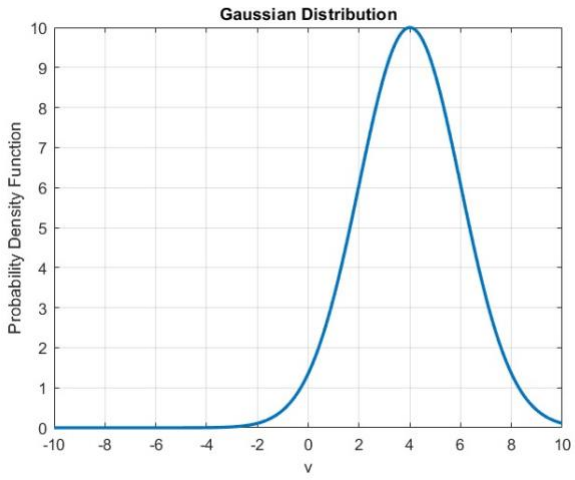


The key in Cyan is the 'C' note in the fourth octave on the Western scale, which has a principal frequency of 260 Hz. The key in yellow is the 'A' in the fourth octave (called middle 'A') which has a principal frequency of 440 Hz. When a note on the piano is played, a hammer strikes a string, and string vibrates at the principal frequency and its harmonics. This vibrates then causes vibrations in the piano body, which in turn emits the sound as pressure waves. This mini-project concerns the principal frequencies for each note on the piano keyboard and not the harmonics.

The Gaussian probability density function can be represented by the formula

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

where α specifies the amplitude, μ specifies the peak location, and σ specifies the width of the peak. The plot of the Gaussian distribution is a "bell curve." This plot is generated in Matlab below.




```
v = -10:.1:10;
sig=2;
alph=10;
mu = 4;
gausswave = alph*exp(-(v-mu).^2/(2*sig^2));

figure
plot(v,gausswave,'linewidth',2);
xlabel('v')
ylabel('Probability Density Function')
title('Gaussian Distribution')
grid on
```

3.0 Warm-Up

3.1 Gaussian Weighting

(a) We synthesize a Gaussian curve given a center frequency, variance, and vector of frequencies to be evaluated, and the Matlab code below is saved in FrequencyWeighting.m. *6 points.*

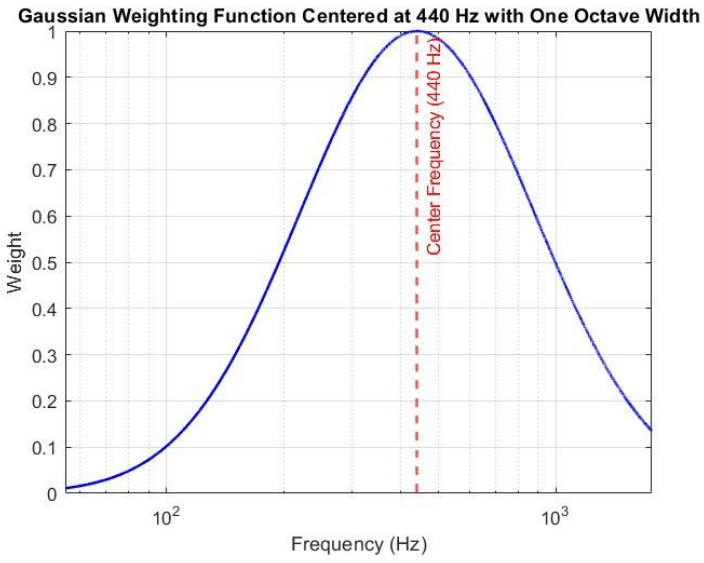
```
function output = FrequencyWeighting(fc,sig,ff)
output = exp(-(log2(ff) - log2(fc) ).^2/(2*sig^2));
%output = exp(-(ff - fc ).^2/(2*sig^2));
end
```

(b) The following Matlab code generates a vector of principal frequencies over five octaves, where there are 12 notes per octave and each note is a 1/12 step from the previous notes. *0 points*

```
ff = 2.^(5:1/12:10);
```

(c) The center frequency is 440 Hz and the width of frequencies is equal to one octave, and the plot of a Gaussian with these parameters is shown below using semilogx. *6 points*

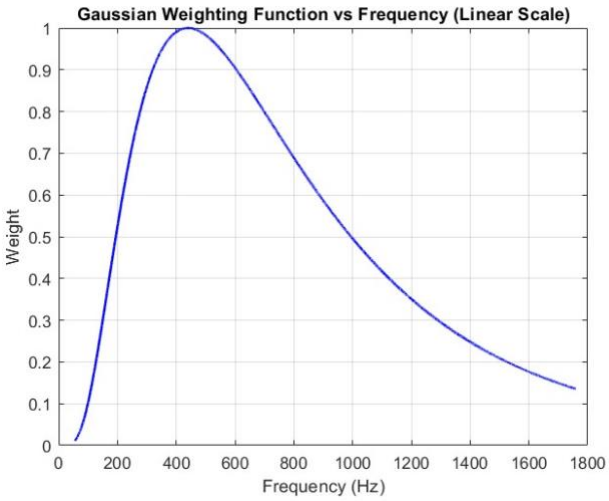
```
fc = 440;
wd = (1760 - 55)/6;
sig = 1;
frequencies = 55:1/12:1760;
weights = FrequencyWeighting(fc,sig,frequencies);

%Plot using the semilogx command (log scale)
figure;
semilogx(frequencies, weights, 'b-', 'LineWidth', 1.5); % Plot with logarithmic x-
axis
xlabel('Frequency (Hz)'); % Label for x-axis
ylabel('Weight'); % Label for y-axis
title('Gaussian Weighting Function Centered at 440 Hz with One Octave Width'); %
Title of the plot
grid on; % Turn on the grid for better visualization
hold on;
xline(fc, 'r--', 'LineWidth', 1.5, 'Label', 'Center Frequency (440 Hz)',
'LabelHorizontalAlignment', 'right'); % Vertical line at center frequency
hold off;
```



(d) To plot the Gaussian as a function of frequency, we can use the plot command, and the resulting plot is shown below. *6 points.*

```
% Plot using the plot command (linear scale)
figure;
plot(frequencies, weights, 'b-', 'LineWidth', 1.5); % Plot with linear x-axis
xlabel('Frequency (Hz)'); % Label for x-axis
ylabel('Weight'); % Label for y-axis
title('Gaussian Weighting Function vs Frequency (Linear Scale)');
grid on;
```

The Gaussian appears distorted because the plot command uses a linear scale for frequency. Since the Gaussian function is defined on the log scale (log2(f)), the bell shape is compressed towards the center frequency when plotted on a linear frequency axis, causing distortion. When using semilogx, the x-axis is logarithmic (base 2), which aligns with the Gaussian function's definition in terms of log2(f). This restores the expected bell shape because the x-axis correctly represents the frequency spacing as intended by the Gaussian distribution in log-frequency space.



3.2 Synthesize Octaves with Gaussian Weighting

We can synthesize and play a signal that is sampled at 8000 Hz and composed of C_4, C_5, C_6, C_3, and C_2 with the following Matlab code.

(a) *9 points*

```
%3.2 Synthesize Octaves with Gaussian Weighting
% Parameters
fs = 8000;        % Sampling frequency in Hz
duration = 2;     % Duration of the signal in seconds
t = 0:1/fs:duration-(1/fs);  % Time vector
fc = 440;         % Center frequency of the Gaussian at 440 Hz
sigma = 1;        % Width of one octave in log scale

% Key numbers for the notes: C2, C3, C4, C5, C6
% MIDI key numbers: C2 (16), C3 (28), C4 (40), C5 (52), C6 (64)
%keynums = [36, 48, 60, 72, 84];
keynums = [16, 28, 40, 52, 64];

% Frequencies corresponding to the key numbers for C2, C3, C4, C5, and C6
% Note that 49 is the keynum of A4 (the reference)
frequencies = 440 * 2.^((keynums - 49)/12);

% Calculate Gaussian weights for these frequencies
weights = FrequencyWeighting(fc,sigma,frequencies);

% Initialize the final signal
final_signal = zeros(size(t));
```

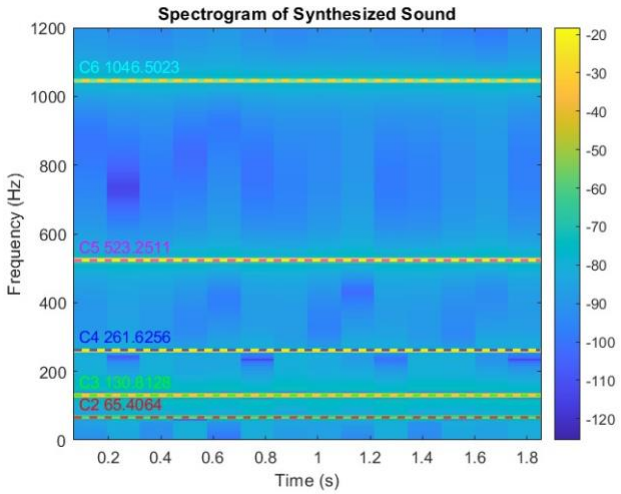
```
% Generate each note using key2note function and add with Gaussian weights
for i = 1:length(keynums)
    note = key2note(weights(i),keynums(i), duration, fs);  % Synthesizes the correct
sinusoidal signal for the key number
    final_signal = final_signal + note; % Weight the note by the Gaussian and sum
end

% Normalize the final signal to avoid clipping
final_signal = final_signal / max(abs(final_signal));

% Play the synthesized sound
sound(final_signal, fs);

% Optional: Save the sound to a file
audiowrite('Gaussian_Weighted_Octave_Sound.wav', final_signal, fs);
```

(b) *9 points*

We can plot the spectrogram of the synthesized signal with the following Matlab code.

```
% Plot the spectrogram
window_length = 2048; % Long window length for good frequency resolution
overlap = window_length / 2; % 50% overlap
nfft = window_length; % Number of FFT points

figure;
[S, F, T, P] =spectrogram(final_signal, window_length, overlap, nfft, fs, 'yaxis');
title('Spectrogram of Synthesized Sound');
imagesc(T, F, 10*log10(P)); % Use imagesc for better colormap handling
axis xy; % Correct the orientation of the y-axis
title('Spectrogram of Synthesized Sound');
xlabel('Time (s)');
ylabel('Frequency (Hz)');
ylim([0 1200]); % Limit frequency range to show relevant peaks
colorbar;
colorbar;

% Add lines for each frequency to identify peaks
hold on;
yline(frequencies(5), 'c--', 'LineWidth', 1.5, 'Label', ['C6 '
num2str(frequencies(5))], 'LabelHorizontalAlignment', 'left');
yline(frequencies(4), 'm--', 'LineWidth', 1.5, 'Label', ['C5 '
num2str(frequencies(4))], 'LabelHorizontalAlignment', 'left');
yline(frequencies(3), 'b--', 'LineWidth', 1.5, 'Label', ['C4 '
num2str(frequencies(3))], 'LabelHorizontalAlignment', 'left');
yline(frequencies(2), 'g--', 'LineWidth', 1.5, 'Label', ['C3 '
num2str(frequencies(2))], 'LabelHorizontalAlignment', 'left');
yline(frequencies(1), 'r--', 'LineWidth', 1.5, 'Label', ['C2 '
num2str(frequencies(1))], 'LabelHorizontalAlignment', 'left');
hold off;
```

In the spectrogram on the right, the dashed lines highlight the peak frequencies which have different colors corresponding to different amplitudes in the spectrogram scale.



4.0 A Musical Illusion

(a) We synthesize a C-major scale with the Matlab code below using a sampling rate of 22050 Hz and a sum of nine sinusoids. The code for generating frequencies is saved in GenerateFrequency.m. *6 points.*

```
function output = GenerateFrequency(freq)
output = freq* 2.^(-4:4);
end
```

```
% Parameters
fs = 22050;         % Sampling frequency in Hz
duration = 1;       % Duration of each note in seconds
t = 0:1/fs:duration-1/fs;  % Time vector for each note

% MIDI key numbers for C-major scale starting at middle-C (C4)
% C4 (40), D4 (42), E4 (44), F4 (45), G4 (47), A4 (49), B4 (51), C5 (52)
keynums = [40, 42, 44, 45, 47, 49, 51, 52];

final_signal = [];
for key=keynums

freq = 440 * 2.^((key - 49)/12);
octaves = GenerateFrequency(freq);

note_signal = zeros(size(freq));
for f = octaves
    % ignore all the weithing amplitudes as stated in part a)
 note_signal = note_signal + cos(2*pi*f*t);
end

note_signal = note_signal/max(abs(note_signal));

final_signal = [final_signal note_signal];
end
final_signal = final_signal / max(abs(final_signal));

sound(final_signal,fs)
% Optional: Save the sound to a file
audiowrite('C_Major_Scale_With_Octaves.wav', final_signal, fs);
```

(b) We can modify the code to play the scale 5 times and add silence between notes. *6 points.*

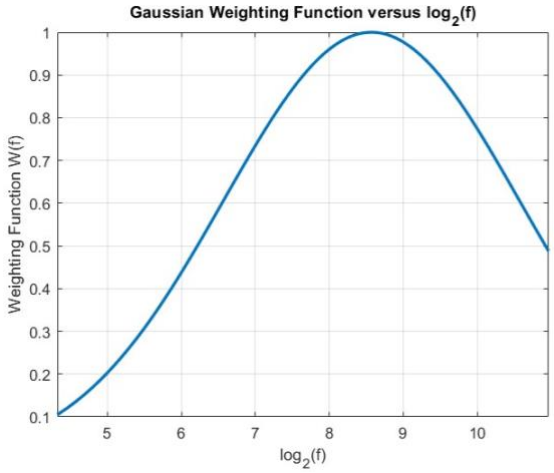
```
% Parameters
fs = 22050;         % Sampling frequency in Hz
```

```
note_duration = 0.5;   % Duration of each note in seconds
silence_duration = 0.2; % Duration of silence between notes in seconds
t_note = 0:1/fs:note_duration-1/fs;  % Time vector for each note
t_silence = zeros(1, round(silence_duration * fs));  % Silence vector

% MIDI key numbers for C-major scale starting at middle-C (C4)
% C4 (40), D4 (42), E4 (44), F4 (45), G4 (47), A4 (49), B4 (51), C5 (52)
keynums = [40, 42, 44, 45, 47, 49, 51, 52];

% Function to generate frequencies for each key, including octaves
generate_frequencies = @(freq) freq * 2.^(-4:4); % Generate frequencies for 4 octaves
below and above

% Initialize the final signal to empty
final_signal = [];

% Repeat the C-major scale five times
for repetition = 1:5
    % Loop through each key number in the C-major scale
    for key = keynums
        % Calculate the frequency for the current key
        freq = 440 * 2^((key - 49) / 12); % Calculate frequency from MIDI key number
(49 is A4, 440 Hz)

        % Generate the frequencies including 4 octaves below and above
        octaves = GenerateFrequency(freq);

        % Initialize note signal with zeros
        note_signal = zeros(1, length(t_note));

        % Sum sinusoids for each frequency
        for f = octaves
            note_signal = note_signal + cos(2 * pi * f * t_note);
        end

        % Normalize note signal to prevent excessive amplitude
        note_signal = note_signal / max(abs(note_signal));

        % Append the note signal and silence to the final signal
        final_signal = [final_signal, note_signal, t_silence];
    end
end

% Normalize the final signal to avoid clipping
final_signal = final_signal / max(abs(final_signal));

% Play the synthesized C-major scale
sound(final_signal, fs);

% Optional: Save the sound to a file
audiowrite('C_Major_Scale_With_Octaves_Repeated.wav', final_signal, fs);
```

(c) We use amplitude weighting with a Gaussian and plot with $\sigma = 2$. *6 points.*

```
% Parameters for the Gaussian weighting
fc = 380;       % Center frequency between 260 and 500 Hz
```

```
sigma = 2;      % Standard deviation in log2(f) scale

% Frequency range for plotting
f = linspace(20, 2000, 1000);  % Frequencies from 20 Hz to 2000 Hz

% Calculate Gaussian weighting function as a function
of log2(f)

W = FrequencyWeighting(fc,sigma,f);

% Plotting the weighting function versus log2(f)
figure;
plot(log2(f), W, 'LineWidth', 2);
xlabel('log_2(f)');
ylabel('Weighting Function W(f)');
title('Gaussian Weighting Function versus log_2(f)');
grid on;
% Set x-axis limits based on the frequency range
xlim([log2(20), log2(2000)]);
```



(d) We use the Gaussian weighting to create the musical illusion of the Shepard scale with the code below. *6 points*

```
% Parameters
fs = 22050;               % Sampling frequency in Hz
note_duration = 0.5;      % Duration of each note in seconds
silence_duration = 0.2;   % Duration of silence between notes in seconds
t_note = 0:1/fs:note_duration-1/fs;  % Time vector for each note
t_silence = zeros(1, round(silence_duration * fs));  % Silence vector

% MIDI key numbers for C-major scale starting at middle-C (C4)
% C4 (40), D4 (42), E4 (44), F4 (45), G4 (47), A4 (49), B4 (51), C5 (52)
keynums = [40, 42, 44, 45, 47, 49, 51, 52];

% Gaussian weight function parameters
fc = 380;       % Center frequency between 260 and 500 Hz
sigma = 2;      % Standard deviation in log2(f) scale


% Initialize the final signal to empty
final_signal = [];

% Repeat the C-major scale five times
for repetition = 1:5
    % Loop through each key number in the C-major scale
    for key = keynums
        % Calculate the frequency for the current key
        freq = 440 * 2^((key - 49) / 12); % Calculate frequency from MIDI key number
(49 is A4, 440 Hz)

        % Generate the frequencies including 4 octaves below and above
        octaves = GenerateFrequency(freq);

        % Initialize note signal with zeros
        note_signal = zeros(1, length(t_note));
```

```
        % Sum sinusoids for each frequency, weighted by the Gaussian function
        for f = octaves
            % Calculate the Gaussian weight for the current frequency
            weight = FrequencyWeighting(fc,sigma,f);

            % Add the weighted sinusoid to the note signal
            note_signal = note_signal + weight * cos(2 * pi * f * t_note);
        end

        % Normalize note signal to prevent excessive amplitude
        note_signal = note_signal / max(abs(note_signal));

        % Append the note signal and silence to the final signal
        final_signal = [final_signal, note_signal, t_silence];
    end
end

% Normalize the final signal to avoid clipping
final_signal = final_signal / max(abs(final_signal));

% Play the synthesized C-major scale with the illusion effect
sound(final_signal, fs);

% Optional: Save the sound to a file
audiowrite('C_Major_Scale_Illusion_With_Gaussian_Weighting.wav', final_signal, fs);
```

(e) We can plot the spectrogram of the synthesized signal with the following code. *6 points.*

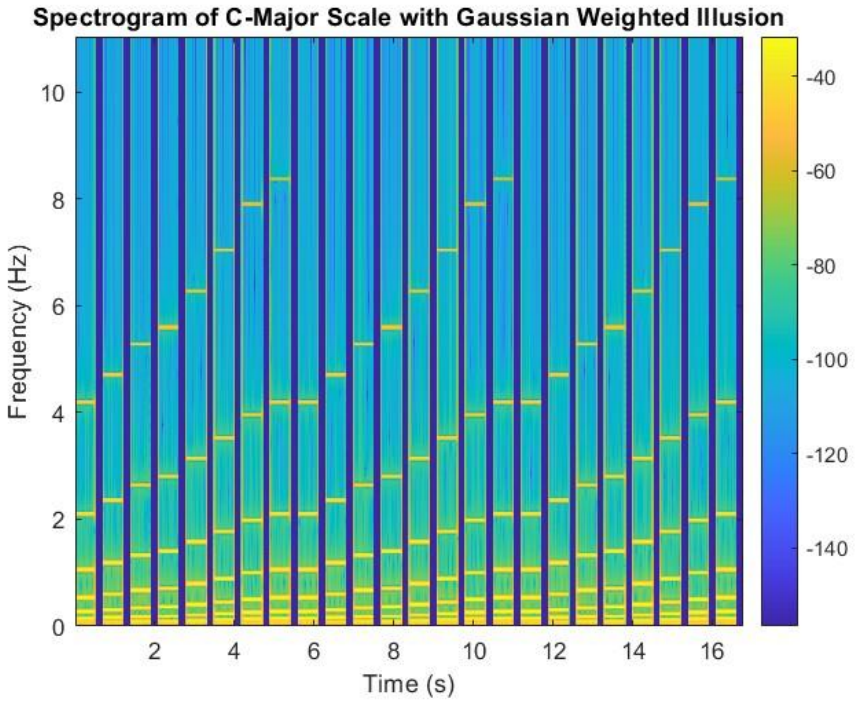
```
% Parameters
fs = 22050;               % Sampling frequency in Hz
note_duration = 0.5;      % Duration of each note in seconds
silence_duration = 0.2;   % Duration of silence between notes in seconds
t_note = 0:1/fs:note_duration-1/fs;  % Time vector for each note
t_silence = zeros(1, round(silence_duration * fs));  % Silence vector

% MIDI key numbers for C-major scale starting at middle-C (C4)
% C4 (40), D4 (42), E4 (44), F4 (45), G4 (47), A4 (49), B4 (51), C5 (52)
keynums = [40, 42, 44, 45, 47, 49, 51, 52];

% Gaussian weight function parameters
fc = 380;       % Center frequency between 260 and 500 Hz
sigma = 2;      % Standard deviation in log2(f) scale

% Initialize the final signal to empty
final_signal = [];

% Repeat the C-major scale three times for spectrogram analysis
for repetition = 1:3
    % Loop through each key number in the C-major scale
    for key = keynums
        % Calculate the frequency for the current key
        freq = 440 * 2^((key - 49) / 12); % Calculate frequency from MIDI key number
(49 is A4, 440 Hz)
```

```
        % Generate the frequencies including 4 octaves below and above
        octaves = GenerateFrequency(freq);

        % Initialize note signal with zeros
        note_signal = zeros(1, length(t_note));

        % Sum sinusoids for each frequency, weighted by the Gaussian function
        for f = octaves
            % Calculate the Gaussian weight for the current frequency
            weight = FrequencyWeighting(fc,sigma,f);

            % Add the weighted sinusoid to the note signal
            note_signal = note_signal + weight * cos(2 * pi * f * t_note);
        end

        % Normalize note signal to prevent excessive amplitude
        note_signal = note_signal / max(abs(note_signal));

        % Append the note signal and silence to the final signal
        final_signal = [final_signal, note_signal, t_silence];
    end
end

% Normalize the final signal to avoid clipping
final_signal = final_signal / max(abs(final_signal));

% Play the synthesized C-major scale with the illusion effect
sound(final_signal, fs);

% Optional: Save the sound to a file
audiowrite('C_Major_Scale_Illusion_With_Gaussian_Weighting.wav', final_signal, fs);

% Generate the spectrogram using MATLAB's built-in function
figure;
spectrogram(final_signal, 1024, 512, 1024, fs, 'yaxis');
title('Spectrogram of C-Major Scale with Gaussian Weighted Illusion');
xlabel('Time (s)');
ylabel('Frequency (Hz)');
colorbar;
```

Illusion of Continuous Rising or Descending: The spectrogram (next page) may show a "rising" or “falling” illusion due to how the Gaussian weights emphasize certain frequencies across the octaves, making it seem like the sound continuously ascends or descends without a clear starting or ending point.

Here are the visual features in the spectrogram to look for:

- *Bands of Energy*: Multiple octave-related bands that are emphasized and de-emphasized cyclically.
- *Smooth Transitions*: Lack of clear separations between the end of one note and the beginning of the next, contributing to the auditory illusion.

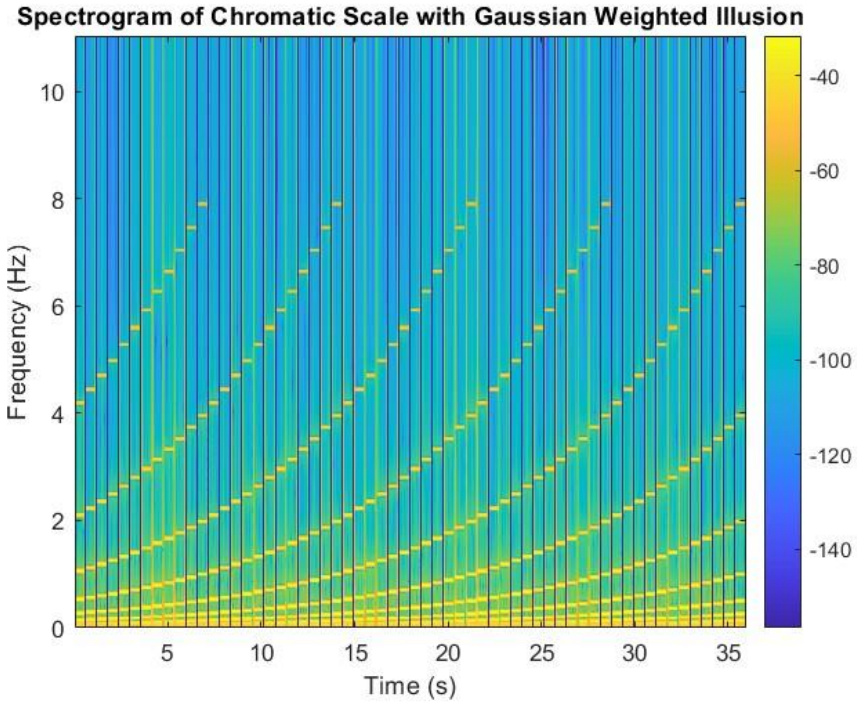


By analyzing the spectrogram, you can visualize how the Gaussian weighting contributes to the auditory illusion, making the scale sound like it's endlessly ascending or descending, even though it repeats the same notes cyclically.

(f) We play every note in the octave and notice the difference in the illusion sound with the following Matlab code. *12 points.*

```
% Parameters
fs = 22050;                % Sampling frequency in Hz
note_duration = 0.5;       % Duration of each note in seconds
silence_duration = 0.1;    % Duration of silence between notes in seconds
t_note = 0:1/fs:note_duration-1/fs;  % Time vector for each note
t_silence = zeros(1, round(silence_duration * fs));  % Silence vector

% MIDI key numbers for all twelve semitones within an octave, starting at C4
% C4 (40) to B4 (51)
keynums = 40:51;  % Chromatic scale from C4 to B4

% Gaussian weight function parameters
fc = 380;       % Center frequency between 260 and 500 Hz
sigma = 2;      % Standard deviation in log2(f) scale

% Initialize the final signal to empty
final_signal = [];

% Repeat the chromatic scale sequence multiple times
repetitions = 5;  % Number of times to repeat the sequence

for repetition = 1:repetitions
```

```
    % Loop through each key number in the chromatic scale
    for key = keynums
        % Calculate the frequency for the current key
        freq = 440 * 2^((key - 49) / 12); % Calculate frequency from MIDI key number
(49 is A4, 440 Hz)

        % Generate the frequencies including 4 octaves below and above
        octaves = GenerateFrequency(freq);

        % Initialize note signal with zeros
        note_signal = zeros(1, length(t_note));

        % Sum sinusoids for each frequency, weighted by the Gaussian function
        for f = octaves
            % Calculate the Gaussian weight for the current frequency
            weight = FrequencyWeighting(fc,sigma,f);

            % Add the weighted sinusoid to the note signal
            note_signal = note_signal + weight * cos(2 * pi * f * t_note);
        end

        % Normalize note signal to prevent excessive amplitude
        note_signal = note_signal / max(abs(note_signal));

        % Append the note signal and silence to the final signal
        final_signal = [final_signal, note_signal, t_silence];
    end
end

% Normalize the final signal to avoid clipping
final_signal = final_signal / max(abs(final_signal));

% Play the synthesized chromatic scale with the illusion effect
sound(final_signal, fs);

% Optional: Save the sound to a file
audiowrite('Chromatic_Scale_Illusion_With_Gaussian_Weighting.wav', final_signal, fs);

% Generate the spectrogram for analysis
figure;
spectrogram(final_signal, 1024, 512, 1024, fs, 'yaxis');
title('Spectrogram of Chromatic Scale with Gaussian Weighted Illusion');
xlabel('Time (s)');
ylabel('Frequency (Hz)');
colorbar;
```



1. *Chromatic Scale*: The MIDI key numbers 60 to 71 correspond to all twelve semitones from C4 to B4, making up the full chromatic scale within one octave.
2. *Gaussian Weighting*: The Gaussian weighting is applied to emphasize frequencies around 380 Hz, with weights decreasing for frequencies further from the center on a logarithmic scale.
3. *Improved Illusion*: Including all 12 notes in an octave creates a smoother, more continuous pitch progression, which often enhances the illusion of an endlessly rising or falling scale. The listener perceives a continuous transition without the distinct intervals of a major scale.
4. *Repetition*: Repeating the chromatic scale multiple times strengthens the illusion and helps to maintain the perceptual effect.
5. *Spectrogram Analysis*: The spectrogram will show overlapping bands of frequencies, with energy smoothly transitioning across the notes, illustrating the continuous nature of the illusion.

(g) Zip file of all prepared Matlab code files. 6 points

5.0 Conclusion *(7 points)*

With Gaussian weighting and synthesizing sinusoids to create signals, we can create and play the Shepard scale. The illusion comes from using frequencies separated by octaves and weighting the amplitudes. Choosing the parameters of amplitude and variance is important when creating the Gaussian, and the Gaussian used in the mini project depends on the log of the frequency.

References

[1] James McClellan, Ronald Schafer, and Mark Yoder, "Lab P-6: Synthesis of Sinusoidal Signals—A Music Illusion", *Signal Processing First*, Companion Web Site.

[2] Wikipedia, "Piano key frequencies", accessed Dec. 1, 2024.