ECE 2026

Introduction to Signal Processing

Lab Report 5

Name of the Student: Sunny Patel

E-mail: sunnypatel18@gatech.edu

Lab Section: L08

Recitation Instructor: Professor Coyle

Submitted to: Chirag Modi

1. **Introduction**

The purpose of this lab is to reproduce a musical illusion called Shepard’s Scale in which the tones played seem to be continually rising forever, yet seem to stay within one octave. During this lab I expect to learn how to create notes of the Shepard’s Scale by summing individual sinusoids, which are all separated by octaves. I hope to learn how to manipulate the amplitudes of these sinusoids and create the illusion via amplitude-weighted notes in a scale over and over. Using the Gaussian weighting for the complex amplitude, we hope to hear Shepard’s Scale. Background knowledge includes how to perform Gaussian weighting and how to convert a key number to a frequency.

1. **Work Done**

**Part A**

We are synthesizing a C-major scale, starting at middle–C, where each note in the scale is accompanied by eight other notes separated by octaves. Four of the octaves are above and four of the octaves are below from middle C. We used the key2note function that was written in Lab 4 to produce desired notes, which were used to create four keynotes above and four keynotes below the middle C key number (49). After setting the nine keys with eight of the keys being a multiple of 12 away from middle C (49). The following the key numbers would be 1, 13, 25, 37, 49, 61, 70, 82, 94. For the complex amplitude input in the key2note function we ignored amplitude weighting and we used random complex amplitudes. (2\*exp (2\*j)) The purpose for this step is to create one note (middle C) on all the octaves of a keyboard, which will be used to play once through a scale of seven different notes. We expect the outcome to generate a vector of values that are generated from the sum of nine sinusoids at frequencies relative to the key numbers listed above. If we used the soundsc function we should be able to hear a note that is the sum of the nine sinusoids for one second as specified in our input for key2num.

As show below in the code written, we added nine different frequency sinusoids and concatenated the eight harmonics and middle C into one vector. We did this because we can play the note using the soundsc function. In the key2num function the inputs include the random complex amplitude, the key number, the duration (1sec) and the sampling frequency of 22050 as stated in part a of Lab 5. We called the key2note function nine times with nine different frequencies. Variable A is a vector with 22,050 elements because we are sampling all the sinusoids at this sampling frequency.

If we wanted to synthesize a different keynote then we would have to do is change the middle key number from 49 to whatever new key number. This would change the values of vector A and if we used the soundsc function we would be playing a sum of nine different sinusoids all relative to that new key number. This script can be easily converted to a function if we replaced the complex amplitude, (2\*exp (2\*j)) with variable X, 49 with a variable named keynum, the one-second duration with dur, and the sampling frequency (22050) with variable fs. All these variables would be used in the input of the new function we would create. If we decided to use a lower sampling frequency to synthesize these nine sinusoids, the duration of the note played would be shorter because we have a fewer number of samples. If we used a higher sampling frequency there wouldn’t be a noticeable difference in the audio quality because of Shannon’s Sampling Theorem.

|  |
| --- |
| Code #1 – Sum of 9 Sinusoids with the middle key being 49 |
| A = [key2note(2\*exp(2\*j),**49** - 48 , 1, 22050)+ key2note(2\*exp(2\*j),**49** - 36 , 1, 22050)+ key2note(2\*exp(2\*j),**49** - 24 , 1, 22050)+ key2note(2\*exp(2\*j),**49** - 12 , 1, 22050)+ key2note(2\*exp(2\*j),**49** , 1, 22050) + key2note(2\*exp(2\*j),**49** + 12 , 1, 22050)+ key2note(2\*exp(2\*j),**49** + 24 , 1, 22050)+ key2note(2\*exp(2\*j),**49** + 36 , 1, 22050)+ key2note(2\*exp(2\*j),**49** + 48 , 1, 22050)]; |

**Part B**

In this part we are trying to play a scale, which consists of playing seven different notes where each note is the sum of the nine octave-spaced keys. In order to play the scale over and over again we need to play the scale a total of five times, which would total to 35 different keys being played. To make sure that we can hear 35 different keys we introduced silence between each key. The purpose of this step is to create a scale of seven different notes, which will be used to create the Shepard’s scale when the amplitude weighing is introduced. If we repeat the scale being played it will allow us to create the illusion that the played tone seems to be continually rising forever.

In the code written below we added increments for the key number and we also added a “for” loop that has increments from 0 to 6, which equates to seven notes in a scale. Using the repmat function we repeat the scale five times. This created a total of 35 notes being played. For the silence to be introduced between each note we created a vector of zeros that has a length of the sampling frequency divided by four. Once the loop ran we started to build a vector where every time we finished the loop it would concatenate with the previous values of the previous scale. Once that vector was fully concatenated, we inserted the zeros vector between each value from the loop vector. This helped create the pause in the sound, which in turn helped us distinguish the 35 different notes being played in the loop. We use the first concatenated vector for the loop from 0 to 6 and use the function repmat, which repeats the scale five times. Lastly we use the soundsc function to play the specific vector we want to here, if that is the vector with the pauses in between or with no pauses in between.

If we decided to use the repmat function with the third input being a different value, then the scale would be repeated the number of times the value is specified as. If we had more notes in the scale instead of seven we can change the loop definition where the increment counter would increase or decrease depending on the length of the scale. If we didn’t use the pauses with the zeros vector we wouldn’t be able to distinguish the 35 notes easily. The pauses helped us make sure that we ran the loop properly and repeated the scale five times. We used structures instead of arrays because we wanted interweave the zeros vector with the repeated scale and using structures made this process easier. If we used arrays instead of structures we would have to index every even value using the colon operator and then insert the zeros vector at every other position, hence creating an interweaving vector. For the zeros vector we only had one row, with the number of columns being the sampling frequency times ¼ because we the length of the zeros vector is directly proportional to the time of the pause between each note. If we used a higher fs we would get longer pauses and if we used a smaller fs we would get shorter pauses.

|  |
| --- |
| Code #2 – Adding the loop and playing the scale of 7 keys, 5 times. |
| scale.keys = [-48 -36 -24 -12 0 12 24 36 48];  ii = 0;  kk = 0;  tone = cell(1,12);  for ii = 0:6  for kk = 1:length(scale.keys)  nn = ii+1;  tone{1,nn} = [key2note(2\*exp(2\*j),49 - 48 , 1, 22050)+ key2note(2\*exp(2\*j),49 - 36 , 1, 22050)+ key2note(2\*exp(2\*j),49 - 24 , 1, 22050)+ key2note(2\*exp(2\*j),49 - 12 , 1, 22050)+ key2note(2\*exp(2\*j),49 , 1, 22050) + key2note(2\*exp(2\*j),49 + 12 , 1, 22050)+ key2note(2\*exp(2\*j),49 + 24 , 1, 22050)+ key2note(2\*exp(2\*j),49 + 36 , 1, 22050)+ key2note(2\*exp(2\*j),49 + 48 , 1, 22050)];  end  end  %z is the pause between each note  z = zeros(1,22050\*0.25); % create sample of 0 frequencies for a pause between each note  %make one with a pause between each note  soundPause = [tone{1,1} z tone{1,2} z tone{1,3} z tone{1,4} z tone{1,5} z tone{1,6} z tone{1,7}];  %repeat five times (pause):  soundFinalpause = repmat(soundPause,1,5);  %repeat each sound  soundsc(soundFinalpause,22050); |

**Part C**

In this part of the lab we are finally introducing amplitude weighting, which would call the Gaussian form. The lab mentions to have the center of the Gaussian form somewhere between 260 and 500 Hz. We decided to choose 440 Hz because that would correlate to the key numbers used in the previous parts of the lab. The purpose for the amplitude weighing is to create the illusion seen in Shepard’s Scale. The entire reason why the illusion occurs is because of this amplitude weighing. In part A, we disregarded the complex amplitude and used a random complex amplitude just to make sure we can hear the scale being played once and then five times. The amplitude weighing will be inserted in the key2num function as the first input, which is the complex amplitude. In the previous lab, we already wrote a Gaussian function that incorporates the amplitude weighing. If we replaced the random complex amplitude with the Gaussian function we should expect to hear the Shepard’s scale, where the illusion that the notes are being played continually forever.

In the code written below, we replaced the first input of the key2num function with the Gaussian amplitude weighing function. This function was given to us in the previous lab. In Lab 5 Part C it asks us to use two as the sigma value, which is responsible for controlling the width along the log base 2 axis. In this part it asked us to use a frequency between 260 and 500 Hz where the Gaussian form will be centered. We decided to use 440 Hz because it correlates to the key number we used in this lab (49). Also in this part we were asked to create a plot of the weighting function. The plot needs to be plotted versus the log base 2. At the end of this code I added the gauss\_weight2 function, which is responsible for plotting the log base 2. We used the amplitude of the keynote equation that was given to us and set the center frequency (fc) to 440 Hz. When I plotted the function I created two plots with one being the regular plot function and the other one using the semilogx function. I used the semilogx function because the scale for the x-axis is a logarithmic (base 10) scale.

If we changed the center frequency (fc) then the plot of the log base 2 will have a Gaussian centered at the new specified frequency. If we choose our center frequency to be 380 for example then the center of the Gaussian curve will be at 380 Hz. If we choose a different sigma value instead of using two, the length of Gaussian curve will be longer or shorter depending on the value of sigma. If sigma were a larger number then the width would be larger, and if the sigma were a smaller number then the width would be smaller. The width of the curve and the value of sigma are directly proportional. If we decided to plot versus f instead of lag base 2 f then the plot would be appear distorted because the axis is logarithmic. If we used the semilogx function plot versus f then the plot wouldn’t look distorted because the scale of the x-axis is in base 10.

|  |
| --- |
| Code #3 – Adding the Gaussian Weighting to the Complex Amplitude & Plotting |
| scale.keys = [-48 -36 -24 -12 0 12 24 36 48];  fs = 11025\*2;  ii = 0;  kk = 0;  fc = 440;  sigma = 2;  tone = cell(1,12);  for ii = 0:6  for kk = 1:length(scale.keys)  keynum = scale.keys(kk);  nn = ii+1;  tone{1,nn} = [key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 48 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 36 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 24 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 12 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + ii, 0.25, fs) + key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 12 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 24 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 36 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 48 + ii, 0.25, fs)];  end  end  %concatenate each note with the cell contents:  sound = [tone{1,1} tone{1,2} tone{1,3} tone{1,4} tone{1,5} tone{1,6} tone{1,7}];  %repeat five times (no pause):  soundFinal = repmat(sound,1,5);  %repeat each sound  soundsc(soundFinal,fs);  %for graph of gaussian  figure(1)  gauss\_weight2(fc, 2,27.5,7040); %calls gauss\_weight2, see below for function  -----------------------------------------------------------------------  function ww = gauss\_weight2(fc, sigma,f\_start,f\_end)  ff = f\_start : 1/12 : f\_end;  log\_ff = log2(ff);  ww = exp(-1.\*(((log\_ff)-log2(fc)).^2)/(2.\*(sigma).^2)); %amplitude of the keynote  subplot(1,2,1)  plot(log\_ff,ww,'b-');  hold on  subplot(1,2,2)  semilogx(ff,ww,'r-');  end |

|  |
| --- |
| Figure #1 – Plot versus log base 2 of f with sigma value of 2. |
| 1. The plot on the left is using the function plot and is plotting the log base 2 of f.  2. The plot on the right is using the function semilogx and is plotting f. This has a logarithmic base x-axis. |

**Part D**

In this part of the lab, we are finally playing Shepard’s Scale and making sure that we do indeed hear the illusion where the notes are being played continually forever. The Gaussian weight function that we added to the complex amplitude input of the key2num function, should work properly and create the illusion. The purpose of this part is to show that the illusion actually works with the code written above. We want to see if we can hear the illusion where the tones keep increasing forever. We should expect to hear the Shepard’s Scale clearly and it should be very hard to figure out where one scale ends and where the next scale starts.

When we ran the code we heard the illusion and successfully played Shepard’s Scale. We compared our sound to the one on the site of the Acoustical Society of America, and it was pretty accurate. We couldn’t differentiate where each scale ended and where each scale started, which is what we wanted to expect. To our ears it sounded as if the notes were continuing to increase forever. We emailed our teacher assistant with the audio file, which is hard evidence that out code successfully played Shepard’s Scale.

In this part it asked us to experiment with the sigma value to create a good sounding illusion. We played around with the sigma value by increasing and decreasing it and if we increases the sigma value their wasn’t a huge noticeable difference. On the other hand if we made the sigma value very small such as .01, we couldn’t hear anything. I think the reason is because the width of the Gaussian curve is too small to be heard. If we played the loop for a longer period of time, you would be able to hear the illusion for a longer period of time and it creates a better illusion. If we had an infinite loop then this illusion would continually play forever, which is what our objective of this lab was to achieve.

**Part E**

In this part of the lab we have to create a spectrogram when Shepard’s Scale is played three times. The spectrogram allows for visual confirmation that Shepard’s Scale does indeed create and illusion. The purpose of this part is for the spectrogram to display a linear amplitude scale, which makes it easier to explain the illusion. We should expect to see three partial Gaussian curves all with the same highest frequency.

In the code written below, in order to play the Shepard’s Scale three times instead of five we changed the second input of repmat to three (highlighted in the code). This would repeat the scale a total of three times. For the spectrogram we used the spectrogram function and plotted the soundFinal, which is the vector of values without any pauses. We wanted to plot this because we can quickly hear the increase and scale and it helps hear the illusion better. The second and third inputs are the window length and overlap length respectively and the 512 and 384 values are a common call to the function. The fourth input is the size of Fast Fourier Transform (FFT) and the last input is the sampling rate. In the common call to the function the fourth input is set to 512 and the fs is set in the code to 22050 Hz.

If we decided to use a higher number such as 1024 for the second input then it would be better to see the separate spectrum lines because there is a longer window length. If we used a larger sampling frequency the Gaussian curves have more samples and the spectrogram doesn’t show any lines for the frequencies higher than the ones in the scale. If we used a lower sampling frequency, there are multiple tones being played at the same time, which distorts the Shepard’s Scale and the illusion no longer remains.

|  |
| --- |
| Code #4 – Spectrogram plot code with the Shepard’s Scale played three times |
| scale.keys = [-48 -36 -24 -12 0 12 24 36 48];  scale.durations = 0.25 \* ones(1,length(scale.keys));  fs = 11025\*2;  ii = 0;  kk = 0;  fc = 440;  sigma = 2;  tone = cell(1,12);  for ii = 0:6  for kk = 1:length(scale.keys)  keynum = scale.keys(kk);  nn = ii+1;  tone{1,nn} = [key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 48 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 36 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 24 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 12 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + ii, 0.25, fs) + key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 12 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 24 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 36 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 48 + ii, 0.25, fs)];  end  end  %z is the pause between each note  z = zeros(1,fs\*0.25); % create sample of 0 frequencies for a pause between each note  %concatenate each note with the cell contents:  sound = [tone{1,1} tone{1,2} tone{1,3} tone{1,4} tone{1,5} tone{1,6} tone{1,7}];  %repeat five times (no pause):  soundFinal = repmat(sound,1,3);  %repeat each sound  soundsc(soundFinal,fs);  %plot spectrogram:  figure(1)  spectrogram(soundFinal,512,384,512,fs,'yaxis'); |

|  |
| --- |
| Figure #2 – Spectrogram of Time vs. Frequency |
| This spectrogram on the left has three partial Gaussian curves plotted. When the first Gaussian curve ends it drops down to the frequency where the first Gaussian curve begins. |

**Part F**

In this part of the lab we are now playing 12 notes in the octave instead of seven previously. The purpose of this is to see whether on not the illusion would sound better than it already did when we played seven notes. We should expect to hear a much better Shepard’s Scale because we are playing all the notes in a scale. The pitch difference of the notes along each octave will be more obvious.

In the code written below all we changed was the interval of the “for” loop with the increment counter from 0:6 to 0:11, which changes the scale from seven notes to 12 notes. This change creates a longer Shepard’s Scale, which makes the illusion sound better than just with seven notes. The audio file will be sent to the teacher assistant for hardcore evidence that the Shepard’s Scale does indeed sound better.

If we changed the increment counter more than 12 notes you would be hearing the same scale again since there are only 12 notes in a scale. This would distort the Shepard’s Scale and you wouldn’t hear the illusion properly. If you choose a smaller increment counter then the Shepard’s Scale wouldn’t be as noticeable of an illusion.

|  |
| --- |
| Code #5 – Changing the scale from seven notes to 12 notes |
| scale.keys = [-48 -36 -24 -12 0 12 24 36 48];  scale.durations = 0.25 \* ones(1,length(scale.keys));  fs = 11025\*2;  ii = 0;  kk = 0;  fc = 440;  sigma = 2;  tone = cell(1,12);  for ii = 0:11  for kk = 1:length(scale.keys)  keynum = scale.keys(kk);  nn = ii+1;  tone{1,nn} = [key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 48 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 36 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 24 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum - 12 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + ii, 0.25, fs) + key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 12 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 24 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 36 + ii, 0.25, fs)+ key2note(exp(-1.\*((log2(keynum)-log2(fc)).^2)/(2.\*(sigma).^2)),keynum + 48 + ii, 0.25, fs)];  end  end  %concatenate each note with the cell contents:  sound = [tone{1,1} tone{1,2} tone{1,3} tone{1,4} tone{1,5} tone{1,6} tone{1,7} tone{1,8} tone{1,9} tone{1,10} tone{1,11} tone{1,12}];  %repeat five times (no pause):  soundFinal = repmat(sound,1,5);  %repeat each sound  soundsc(soundFinal,fs); |

**Part G**

In this part of the lab we need to write the .wav file and send it to our teacher assistant for grading. The purpose of this step is to create an electronic distributable form of Shepard’s Scale, which in this case is in a form of an audio file. We expect that the teacher assistant can successfully hear that the audio file does indeed perform the Shepard’s Scale properly. When we used the 22050 for the sampling frequency the sound quality was distorted. I fixed this issue by changing the bit input to 32. The default is 16. I emailed the .wav file to my teacher assistant for verification.

|  |
| --- |
| Code #6 – Wavwrite function to create audio file for 7 notes and 12 notes |
| wavwrite(soundFinal,22050,32,'7notes.wav');  wavwrite(soundFinal,22050,32,'12notes.wav'); |

1. **Conclusion**

After completing all eight parts of the lab, we figured out how each part is an integral part of the lab and figuring out Shepard’s Scale. The use of Gaussian weighing the complex amplitudes can be manipulated to create the illusion of continually rising notes. If we created the Shepard’s Scale with 7 notes instead of 12 notes, the illusion doesn’t seem to be precise, but when 12 notes are used (full scale) the Shepard’s Scale sounds the best. If we just used any random complex amplitude then the illusion of Shepard’s Scale wouldn’t be achieved. This implies that Gaussian weighting has a major effect on how the sinusoids are affected and if manipulated correctly one can perform illusions such as Shepard’s Scale.

During the lab we encountered a major problem on how to concatenated the 35 notes together that when we played it would sounds like the notes are increasing continually. We thought that a simple concatenation of the vectors every time the loop was ran would work, but that didn’t seem to do the job. We then had to find out a way to properly combine these values that were being produced every time a look was ran. We figured out that structures would be the most effective way to create the inter-weaving situation allowing for the scale to be properly heard and the illusion to be successful. We also had some trouble with figuring out how to add pauses between each note being synthesized. We thought that just sampling adding zero values in between the produced value from the loop would easily do the job, but the pauses weren’t long enough since we were sampling at a rate of 22050. Having one zero value in between would only create a pause for 1/22050 of a second, which is not a major pause. We then decided to create a row of zeros that would be inserted in between each produced value. This proved to work and then we adjusted the length of the zeros vector to create the perfect length for each pause between each note being played.

For the plot of log base 2 of f we created two options for plotting the same graph. We used the regular plot function, which required us to use the log base 2 of f. We could have easily done the same plot with the function semilogx, which would have changed the scale of the x-axis to a log base and then you would have to simply plot f. We also had the choice of adjusting the sigma value to make a good sounding illusion. We decided to stick with the sigma value of 2 since we tried a higher sigma value, but the sound of the illusion didn’t have a noticeable difference. If we did use a very small sigma value then there would have been no audio file since the width of the Gaussian curve would be too small to play.

I think that I learned a lot about how the key system works on a keyboard and how to figure out each key’s respective frequency. I also expected that the Gaussian curve would have had a full Gaussian curve when the spectrogram was plotted, but just a shift every time the loop was executed, but I was proven wrong where there was only a partial Gaussian curve. I didn’t think that the Gaussian curve would end and then the frequency would drop and then continue to rise again. I think that our ears aren’t fully able to pick up on the difference where the frequencies drop and then rise, which creates the Shepard’s Scale.