Acoustics Final Report

Northeastern University MUSC2350, Spring 2020

Matt Dailis

April 20, 2020

Contents

Web version: https://mattdailis.github.io/simulating-strings/.

Preface

This is Matt Dailis's[1] work for *Acoustics and Psychoacoustics of Music* taught by Victor Zappi[2] at Northeastern University, Spring 2020.

In Section 1, I will describe my attempt at using additive synthesis to simulate the sound of a guitar. I jumped right into this without examining much prior work, so my approach was a little naive and primitive. I do explain concepts along the way, and I reference some alternative approaches at the end.

In Section 2, I will give a more concise description of edgetones and wind instruments, based largely on what we covered in class.

The appendix contains a full program listing of my octave program. It can also be found at https://github.com/mattdailis/simulating-strings.

1 Simulating Strings

My intent is to attempt to simulate a guitar using purely math. I wanted to know how big the gap is between the theory I learned in class and the true complexity of a real musical instrument. I pulled out my notes from class, fired up my laptop, and decided to give it a try!

1.1 Getting set up with Octave

For this project, I used *GNU Octave*, an open source programming language and environment for mathematical modeling, based off of MATLAB.

Octave has an `audioplayer` function, which when provided with a vector of floating point numbers between -1 and 1, treats them as a waveform and plays them back.

```
audioplayer (vector, bit_rate, bit_depth)
```

A *sample* is a discrete measurement of *sound pressure level* (SPL) averaged over a predefined duration

[1] https://mattdailis.github.io
[2] https://toomuchidle.com/

of time. The `bit_rate` variable represents the number of samples to play per second. I chose to set this to be 44100, which is a standard bit rate used for CDs.[3] The `bit_depth` variable has to do with the precision of the floating point numbers themselves.

1.2 Pure tones and equal-amplitude harmonics

My approach to simulating strings was to use *additive synthesis*[4]. This means that I attempted to simulate a vibrating string by building it up as a sum of partials. To start, I generated a waveform for the the fundamental frequency using octave's `sinewave` function *(See Listing 1)*. Given a vector size and a period, it returns a set of values between -1 and 1 in the form of a sine wave with the specified period.

```
f1 = sinewave(bitRate * 4, bitRate / 440)
```

Listing 1: Octave provides a convenient sinewave function, which asks for a vector size and a period measured in number of samples

I defined my own convenience function `puretone` which would take the bit rate, duration, and frequency and return the corresponding sine wave *(See Listing 2)*.

```
function puretone(seconds, frequency)
  sinewave(bitRate * seconds,
           bitRate/frequency);
endfunction
```

Listing 2: I defined my own ***`puretone`*** *function which allows me to think in terms of frequency instead of period*

Now I had the ability to make pure tones, but I wanted *harmonics*. A harmonic is a partial whose frequency is an *integer multiple of the fundamental*.[5] We usually only care about the first six harmonics or so, because after that they start to get to very high frequencies near the edge of human hearing. I defined a `createharmonics` function that returns a sum of six harmonics *(See Listing 3)*. Notice that the returned vector must be divided by six to make sure the whole range of values is between -1 and 1.

[3]44100 is a common sampling frequency because of the Sony CD standard: https://en.wikipedia.org/wiki/44,100_Hz

[4]https://en.wikipedia.org/wiki/Additive_synthesis

[5]https://en.wikipedia.org/wiki/Harmonic

```
createharmonics(duration, fundamental):
  f1 = puretone(duration, fundamental);
  f2 = puretone(duration, fundamental * 2);
  f3 = puretone(duration, fundamental * 3);
  f4 = puretone(duration, fundamental * 4);
  f5 = puretone(duration, fundamental * 5);
  f6 = puretone(duration, fundamental * 6);

  return (f1 + f2 + f3 + f4 + f5 + f6) / 6;
```

Listing 3: ***`createharmonics`*** *generates the first six harmonics and adds them together*

I was so excited about the fact that my equations were producing the pitches that I wanted that I created a sample song using this function.

```
A3 = createharmonics(0.5, 220);
A4 = createharmonics(0.5, 440);
A5 = createharmonics(0.5, 880);
B4 = createharmonics(0.5, 495);
C5 = createharmonics(0.5, 523.26);
D4 = createharmonics(0.5, 293.33);
D5 = createharmonics(0.5, 293.33 * 2);
E4 = createharmonics(0.5, 330);
E5 = createharmonics(0.5, 660);
F5 = createharmonics(0.5, 348.84 * 2);
GS4 = createharmonics(0.5, 415.305);

aMinor = [A4, (C5 + E5) / 2,
          E4, (C5 + E5) / 2];
eMajor = [B4, (E5 + GS4) / 2,
          E4, (D5 + GS4) / 2];
dMinor = [A4, (D5 + F5) / 2,
          D4, (D5 + F5) / 2];

song = [aMinor, eMajor, aMinor, eMajor,
        dMinor, aMinor, eMajor,
        A4, E4, A3];
playSound(song, bitRate)
```

Listing 4: A sample song using the functions created so far - it sort of sounds like music!

You can hear the result here: http://mattdailis.github.io/strings/audio/string-simulation-0.wav

After listening to the result, I could recognize this as music, but it sounded nothing like a guitar. What's missing?

First off, in a string, the relative amplitudes of the harmonics are not all the same.[6] Secondly, for a plucked instrument, the amplitudes of all of the

[6]I found this out by plucking a string on my guitar and looking at the spectrum in the n-Track Tuner mobile app

harmonics change over time, eventually diminishing to silence. Lastly, the soundboard of the instrument will act as a filter affecting the output of the instrument.[7] Let's tackle these issues one by one.

1.3 Relative amplitudes of harmonics

First off, the fundamental frequency of a plucked string will always be the most prevalent harmonic.[8] The relative amplitudes of harmonics of a plucked string depend on the pluck location.

We model a pluck as a "kink" in the string.[9] The prevalence of each harmonic depends on whether the initial kink location is at one of that harmonic's nodes or antinodes. Put another way, it depends on the *similarity* of the string shape at the moment of the pluck to the shape of the resonant mode.

Similarity, in linear algebra, is defined as the dot product between two vectors. The more "aligned" those two vectors are, the higher their dot product.

If we take the *fourier transform* of the string shape, we should get an idea for which frequencies are represented. Let's first define the shape of our string.

Let's define a kink in terms of a piecewise function.

Let k be the kink location whose value is between 0 and 1, and L be the length of the string.

$$y_1 = \frac{x}{k}L, x \leq kL$$

$$y_2 = \frac{1-\frac{x}{L}}{1-k}, x > kL$$

The following pairs of graphs show the kink function on the left, and its FFT on the right. The only axis worth looking at is the x axis of the FFTs - each number corresponds to the harmonic index.

These images were generated using *octave-online*[10] with the following call:

```
v = kink(1000, 0.1)
bar(abs(fft(v-mean(v)))(1:10)(2:end))
```

Listing 5: This line of code generated the graphs below

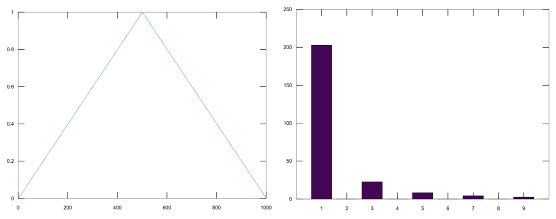

Figure 1: ***kink(0.5)*** *and its FFT*

Notice that the fundamental is always the most prominent, but the behavior of the rest of the harmonics varies. Observe *Figure 1* - the pluck location is in the center of the string, which emphasizes odd harmonics, and has no even harmonics because all even harmonics have a node in the center.

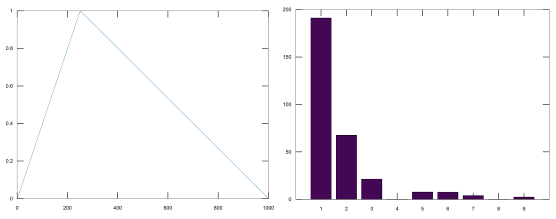

Figure 2: ***kink(0.25)*** *and its FFT*

Moving the pluck location to the quarter point of the string (*Figure 2*), we see more harmonics pop up, but the fourth and eighth (and all multiples of four) are still silent, because the kink location is at the node of the fourth harmonic.

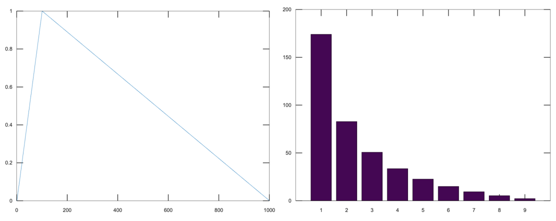

Figure 3: ***kink(0.1)*** *and its FFT*

[7]Mathematical Modelling and Acoustical Analysis of Classical Guitars and Their Soundboards

[8]Intuitively, this is because a "kink" in a string has nodes at the ends no nodes in between, which is similar to the shape of the fundamental

[9]Slides day 23

[10]Every time I tried to use octave's `plot` function on my computer, I got a segmentation fault...

In *Figure 3*, all nine of the first harmonics are present. The tenth is not pictured, but it would be zero, because it has a node at the pluck location.

This is the result of scaling the harmonics using the weights from the FFT:

http://mattdailis.github.io/strings/audio/string-simulation-1.wav

After listening to this result, I found that it sounded a little better - the fundamental was more prominent than before. It still did not sound like a physical string though.

1.4 Damping

When one plucks a string, it does not sustain the sound for very long. Immediately, it starts to lose energy to friction at the imperfect boundaries of the string, as well as friction with the fluid (air) in which it is vibrating.[11] I hoped that adding damping will at least make it sound plausible that the strings are being plucked.

Let's focus on the kinetic energy lost due to the motion of the bridge, since that is more significant than the energy lost to the air.[12] The way we take into account the bridge motion is by modeling it as an impedance mismatch, similar to how we would model a tube open on one end. This results in an exponential decay.

```
function y = damping(x, dampingTime, bitRate)
  y = 0.5 ^ (x / (dampingTime * bitRate));
endfunction
```

Listing 6: I found that a decay halflife of about 0.3 seconds sounded good to me

In this model, all of the frequencies decay at the same rate, which isn't necessarily accurate, although looking at a the spectrum of plucking my guitar string, I think this is a reasonable approximation.[13]

http://mattdailis.github.io/strings/audio/string-simulation-2.wav

1.5 Soundboard

Okay, we've now made a generic plucked string instrument, but what makes a guitar a guitar? One of the aspects that has the biggest contribution to the timbre of a stringed instrument is its *soundboard*. A soundboard is a resonance chamber that takes the input vibration from a string and transforms its frequency spectrum, behaving as an acoustic filter. In a guitar, the string transfers its vibration through the bridge and into the top of the guitar. The top of the guitar is an *idiophone*[14] that creates a pressure wave inside the body as it vibrates. It is the modes of this piece of wood plus the sound propagation inside of the body that together create this acoustic filter.[15]

To implement a filter in octave, I intended to use the `signal` library. While I did eventually manage to install it, I did not have enough time to implement this part before the project deadline. However, I read some papers about soundboard design. Luthiers install *braces*, which are strips of wood glued to the soundboard to create areas of greater stiffness, which encourages modes that have nodes in those locations.[16] That same paper included Figure 4, which shows the frequency responses of "good" versus "bad" quality guitars. They both show peaks around 110 and 220 hertz, though the good guitars have higher amplitude peaks.

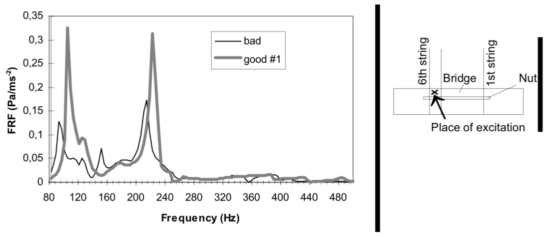

Figure 4: This diagram was taken from "Frequency Response Function Of A Guitar - A Significant Peak" By Samo Sali

[11]The physics of vibrating strings - Giordano, Gould, Tobochnik

[12]The physics of vibrating strings - Giordano, Gould, Tobochnik

[13]obtained using the n-Track Tuner mobile app

[14]at first I thought it was a membranophone, but I suppose there is no tension involved

[15]`https://newt.phys.unsw.edu.au/music/guitar/guitarchladni.html`

[16]"Frequency Response Function Of A Guitar - A Significant Peak" By Samo Sali

1.6 Subtractive synthesis

When I got this far in the project, for the first time I actually searched for "synthesizing guitar sound" on the internet.[17] I found that the most commonly used algorithm for generating guitar sounds does *not* use additive synthesis! Instead, it uses subtractive synthesis, which means it starts with all possible frequencies (i.e. white noise), and filters them down to the frequencies of a guitar.

1.6.1 The Karplus-Strong Algorithm

The Karplus-Strong algorithm[18] is a way of cheaply synthesizing guitar-like sounds using one or two sine wave oscillators. It can be summarized by four steps (see Figure 5)

1. Generate a short burst of white noise
2. Apply delay
3. Pass it through a filter
4. Loop

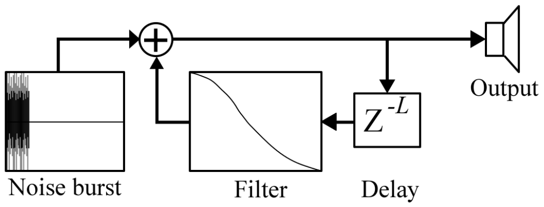

Figure 5: Karplus Strong

The most important part of this algorithm is the interaction of the delay in step 2 with the filter in step 3. The delay helps select the frequency, while the filter creates the timbre. The loop step allows the sound to change over time.

[17]I should have started with this! Although the process of discovery was also very instructive

[18]karplus-strong.pdf

2 Woodwinds: Edgetones

This section will give a shallow overview of wind instruments, with a deep dive in the middle on *edgetones*.

A *wind instrument* is similar to a stringed instrument in that it has a *sound source* and a *sound modifier*. However, instead of having a vibrating soundboard, wind instruments typically have a tube that contains a one-dimensional air column through which sound propagates as a *longitudinal wave*.[19]

2.1 Sound source

The sound source is responsible for generating a stream of vibrating air. We can categorize this generation into three phenomena: *free edge oscillation*, *reeds*, and *vibrating lips*. Here, we will only focus on *free edge oscillation*, since this is most relevant to the concept of edgetones.

2.1.1 Free edge oscillation

In *free edge oscillation*, a steady flow of air needs to hit a sharp object head-on (see Figure 6).

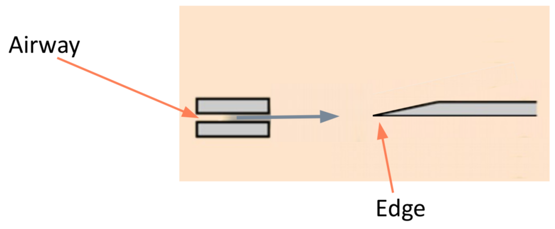

Figure 6: A narrow stream of air passes through an ***airway*** *and hits a sharp* ***edge*** *head-on*

When we talk about a "steady flow of air," we are talking about *laminar flow*. Laminar flow is when a fluid moves in smooth layers (laminae) and each layer is moving in the same direction as the whole fluid, meaning there are no cross-currents or eddies.[20] This flow is laminar inside of the airway, but at some distance from the airway it becomes *turbulent flow*.

[19]A *longitudinal wave* oscillates in the same axis as it propagates.

[20]https://en.wikipedia.org/wiki/Laminar_flow

Turbulent flow is when the motion of a fluid is chaotic and changing. The laminar flow gets a certain distance into the unconstrained air and loses its structure, and becomes turbulent.

If we place a sharp edge at approximately the distance from the airway at which the flow naturally becomes turbulent, we force the flow to pick one side of the edge. The eddies will increase in intensity on that side and cause the flow to flip to the other side (see Figure 7). This phenomenon will repeat in a periodic fashion.

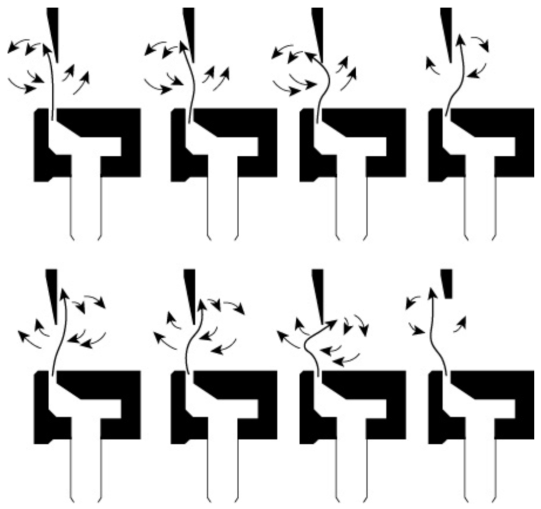

Figure 7: Edgetone illustration from textbook

We can describe the frequency of this periodic fluctuation as a ratio between the the velocity of the air flow, v, and the distance between the airway and the edge, d.

$$f \propto \frac{v}{d}$$

This equation, however, is not 100% correct - the frequency is not continuous. As velocity increases, at a certain point, frequency will have a jump discontinuity (See Figure 8). I do not fully understand why this happens, but it is commonly used by musicians to acheive higher frequencies.[21] Notice the slope of the lines - this is not very convenient for musicians, since it is hard to control your air velocity so precisely. We will see a solution to this in the **edgetones** section.

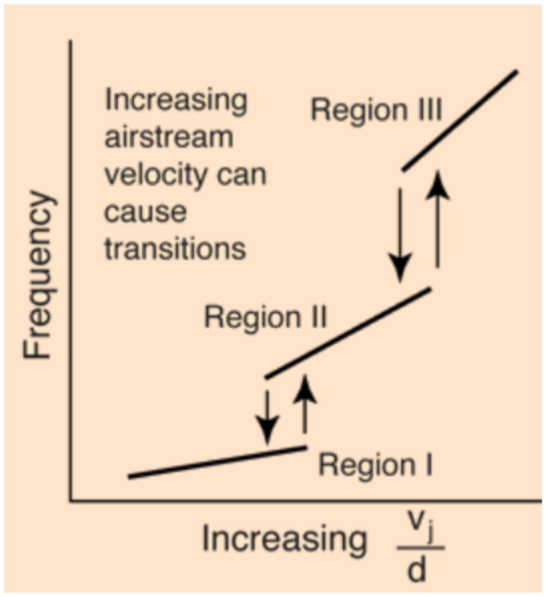

Figure 8: Frequency increases stepwise.

2.2 Sound modifier

Many wind instruments have a long tube called the *bore* that houses the air column. The air column vibrates at resonant modes that depend on the length[22], L, of the bore. c is the speed of sound in air.

$$f_n = \frac{nc}{2L}$$

2.2.1 Edgetones

In a *woodwind* instrument, whose sound source is an airway followed by an edge, the gap between the airway and edge is one of the two open ends of the tube. We learned that tubes with open ends allow resonant modes with antinodes at the ends. This means that

[21]musicians call this technique *overblowing*

[22]assuming no holes

after the initial "transient" part of the sound, the vibration of the air column will *induce* a vibration at the edge. This is called an **edgetone**.

An edgetone is a form of coupling, like sympathetic vibrations in strings. It forms a feedback loop, and most interestingly, it causes the source to vibrate at the resonant mode of the bore. This is significant, because if you remember from Figure 8 and the corresponding equation, the frequency of the source depended on air velocity and distance, which are *not* properties of the bore! The vibration of the air column has a high enough amplitude to overcome the eddies of the turbulent flow at the edge and force the flow to oscillate at a frequency dictated by the properties of the bore.

Does this mean that the air velocity and edge distance have no effect on the output frequency of the instrument? Not quite. While it is true that for small changes in velocity, the frequency remains constant (dictated by the resonant mode of the bore), the jump discontinuities will have an effect on the output frequency. The steps in frequency at the edge will help *select* which resonant mode of the bore will have the highest amplitude. Examine Figure 9. There are now *ranges* of values of $\frac{v}{d}$ that result in the same frequency.[23]

2.3 Modifying the modifier

Unlike an acoustic guitar, which typically leaves the soundboard the same and changes the sound source, woodwinds usually come with the ability to dynamically modify the acoustic properties of the bore. The way they do this is with *holes*. Holes in the bore force nodes at those locations because they fix the pressure at the hole location to be approximately equal to the atmospheric pressure outside of the bore.

3 Bibliography

- The physics of vibrating strings - Giordano, Gould, Tobochnik

[23]This is very convenient for musicians, because it reduces a continuum of frequencies to a discrete set. Frets on a guitar perform a somewhat analogous function.

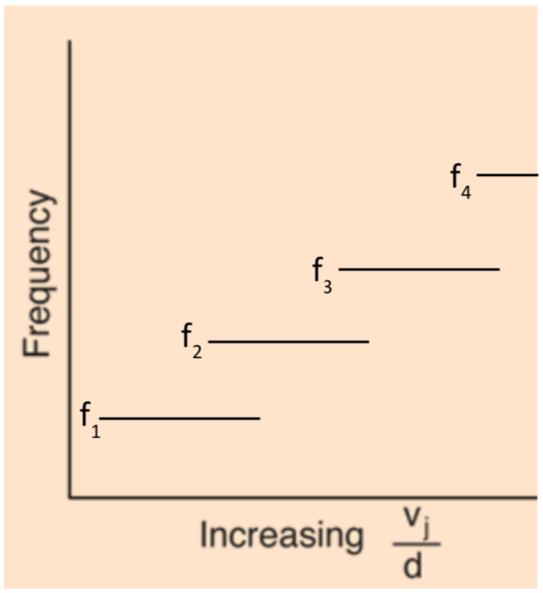

Figure 9: The edgetones flatten pitch change locally, but still react to the jump discontinuities

- Digital Synthesis of Plucked-String and Drum Timbres, Karplus and Strong
- Response Variation in a Group of Acoustic Guitars - Mark French
- Simple model for low-frequency guitar function
- Frequency Response Function Of A Guitar: A Significant Peak - Samo Sali
- Mathematical Modelling and Acoustical Analysis of Classical Guitars and Their Soundboards
- Loaded String simulation source code

4 Appendix

4.1 Program listing

The following is the source code for the octave program I wrote for the simulating strings section.

```
#pkg load signal
```

```
function y = damping(x, dampingTime, bitRate)
  y = 0.5 ^ (x / (dampingTime * bitRate));
endfunction

function y = createDamping(bitRate, duration,
     dampingTime)
  y = arrayfun(@(x) damping(x, bitRate,
      dampingTime), [1 : bitRate * duration])
      ;
endfunction

function playSound(vector, bitRate)
  player = audioplayer (vector, bitRate, 16);
  play (player);
  while(isplaying(player))
  endwhile
endfunction

function y = puretone(bitRate, seconds,
    frequency, phaseShift=0)
  y = sinewave(bitRate * seconds, bitRate/
      frequency, phaseShift);
endfunction

function y = createtone(bitRate, duration,
    frequency, dampingFactors)
  y = puretone(bitRate, duration, frequency)
      .* dampingFactors;
endfunction

function y = createharmonics(bitRate,
    duration, fundamental, weights)

  dampingFactors = createDamping(bitRate,
      duration, 0.3);

  M = [];

  ## Build up matrix where each row is
      another harmonic
  for index = 1 : length(weights)
    M = [M; weights(index) * createtone(
        bitRate, duration, fundamental *
        index, dampingFactors)];
  endfor

  ## Collapse them at the end
  S = sum(M);

  y = S / max(S);
endfunction

function v = ffttest(bitRate, X, checkFreq)
  L = length(X);
  Y = fft(X);
  P2 = abs(Y / L);
  P1 = P2(1:(L / 2) + 1);
  checkIndex = checkFreq * L / bitRate;
  [checkVal, checkIndex2] = max(P1(checkIndex
       - 5 : checkIndex + 5));
  v = checkVal;
endfunction

## location must be between 0.0 and 1.0
function y = kink(L, location)
  k = location;
  x = (0 : L);
  y1 = x / (k * L);
  y2 = (1 - (x / L)) / (1 - k);
  y = [y1(1:k*L), y2(k*L+1:L)];
endfunction

## Returns the first 10 harmonic weights for
    the given pluck location
function y = getHarmonicWeights(pluckLocation
    )
  v = kink(1000, pluckLocation);
  X = abs(fft(v - mean(v)));
  y = (X / max(X))(2:10);
endfunction

function filtertest()
  sf = 800; sf2 = sf/2;
  data=[[1;zeros(sf-1,1)],
        sinetone(25,sf,1,1),
        sinetone(50,sf,1,1),
        sinetone(100,sf,1,1)];
  [b,a]=butter ( 1, 50 / sf2 );
  filtered = filter(b,a,data);
endfunction

function main()
  bitRate = 44100;

  weights = getHarmonicWeights(0.15);
  duration = 2.0;

  A3 = createharmonics(bitRate, duration,
      220, weights);

  C3 = createharmonics(bitRate, duration,
      523.26 / 2, weights);
  E3 = createharmonics(bitRate, duration,
      330, weights);
  E2 = createharmonics(bitRate, duration, 330
       / 2, weights);
  aMinor = [A3, (C3 + E3) / 2, E2, (C3 + E3)
      / 2];

  song = [aMinor];
  playSound(song, bitRate)

  ## Uncomment the next line to save the
      audio to a file
  ## audiowrite("with_damping.wav", song,
      bitRate);
endfunction

main()
```