Eng. 100: Music Signal Processing W24

©J. Fessler
December 17, 2023 11:22

Lab 2: Frequencies of musical tones

1 Abstract

This lab begins the study of music signal processing by determining the frequencies of musical notes. The goals of this lab are: (1) To be able to use a simple signal processing algorithm to compute frequencies of sampled sinusoids; (2) To use this algorithm to compute the frequencies of musical tones in a pure tonal version of "The Victors"; (3) To use simple data visualization methods to determine the relations between these frequencies. The result will be the 12-tone chromatic scale used for most Western music. You will need to know these frequencies to build simple music synthesizers and transcribers in Projects 1 and 3.

2 Background

We now make a first attempt to analyze musical signals. It makes sense to start with the simplest musical signals: pure sinusoidal tones. Even an musically untrained ear can sense that these are very simple signals.

We will analyze a basic tonal version of "The Victors." This version contains sinusoids of different frequencies, so we need a way to compute the frequencies of sinusoids from their samples. Then we need a way of interpreting these frequencies: Why them? What are the relations between them? How can we compute them from the tonal signals?

3 Frequency computation

This lab uses extensively the following trigonometric angle sum and difference identities:

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b), \tag{1}$$
$$\cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b)\,. \tag{2}$$

Adding these identities gives the following equality, called a product-to-sum identity:

$$2\cos(a)\cos(b) = \cos(a+b) + \cos(a-b)\,. \tag{3}$$

This simple formula leads to two different methods for determining the frequency of a sinusoidal signal.

3.1 Tuning fork method

Musicians can use a tuning fork to tune instruments. Understanding how this method works mathematically is best illustrated using an example. Substituting $a = 2\pi 443t$ and $b = 2\pi 3t$ into (3) yields:

$$\underbrace{\cos(2\pi 440t)}_{\boxed{\text{play 440}}} + \underbrace{\cos(2\pi 446t)}_{\boxed{\text{play 446}}} = \underbrace{2\cos(2\pi 3t)\cos(2\pi 443t)}_{\boxed{\text{play}}}.$$

Generalizing the preceding example, one can see that the sum of two sinusoids with the same amplitude and with approximately equal frequencies (that differ by ν Hz) is equal to a sinusoid having frequency midway between these frequencies but with "amplitude" that *varies sinusoidally with frequency* $\nu/2$. You heard an example like this in Lab 1.

Hence one way of measuring the frequency of a sinusoid is as follows:

- Listen to the sum of the sinusoid and another sinusoid at a known and similar frequency;
- This sum will sound like this: "loud-soft-loud-soft," etc. In the example above, even though the formula for the "amplitude" is $2\cos(2\pi 3t)$, which is a 3 Hz variation mathematically, our ears hear the sound get louder and softer 6 times per second (6 Hz) because positive and negative amplitudes sound the same to our ears. This amplitude variation is called "beating" and in this example the "beat" frequency is 6 Hz.
- Vary the known frequency until this "beat" disappears. Then the two sinusoids have the same frequency.

Musicians will recognize this method is similar to how pianos are tuned using a tuning fork[1]. Guitarists use this technique extensively (at least when their electronic tuner has dead batteries). You do not even have to know the two frequencies, as long as the tuning fork vibrates at the correct frequency.

3.2 Signal processing method using `arccos`

We prefer an automatic procedure that operates directly on samples of the sinusoid and uses a computer to determine its frequency, like in a smart phone application for instrument tuning. We can do this as follows. First recall that a sinusoidal signal is:

$$x(t) = A\cos(2\pi f t + \theta)\,.$$

Let $x[n]$ denote samples of the sinusoid at sampling rate S. Recall from Lab 1 that the samples of a sinusoidal signal are:

$$x[n] = x(t)\Big|_{t=n/S} = A\cos(2\pi f t + \theta)\Big|_{t=n/S} = A\cos(2\pi(f/S)n + \theta)\,. \tag{4}$$

Substituting $a = 2\pi(f/S)n + \theta$ and $b = 2\pi f/S$ into (3) and multiplying by A gives:

$$\underbrace{A\cos(2\pi f/S(n+1)+\theta)}_{x[n+1]} + \underbrace{A\cos(2\pi f/S(n-1)+\theta)}_{x[n-1]} = 2\,\underbrace{A\cos((2\pi f/Sn+\theta)}_{x[n]}\cos(2\pi f/S)\,.$$

We rewrite this equality using (4) as follows:

$$x[n+1] + x[n-1] = 2\cos(2\pi f/S)\,x[n]. \tag{5}$$

Now we rearrange (5) to solve for the frequency f:

$$\boxed{f = \frac{S}{2\pi}\arccos\left(\frac{x[n+1]+x[n-1]}{2x[n]}\right).} \tag{6}$$

(We must choose a time sample n for which $x[n]$ is nonzero.) This is a simple formula for computing the frequency f of a pure sinusoidal signal from (almost) any three of its consecutive samples $(x[n-1], x[n], x[n+1])$. We can also use the arccos formula (6) for a sequence of sinusoidal signals having different frequencies (like in music) because it requires only a couple of time samples. This is called pitch tracking. Keep in mind that this arccos method works only for pure sinusoidal signals. Real music is more complicated than (4). Stay tuned (pun intended) for more advanced methods later.

RQ Lab2.1. A sinusoidal signal is sampled at rate $S = 16000\ \frac{\text{Sample}}{\text{Second}}$ and four of its consecutive samples are $(x[11], x[12], x[13], x[14]) = (-1.951, 0, 1.951, 3.827)$. Determine the frequency of the sinusoidal signal. (Round to nearest integer.)

[1]"You Can Tune a Piano, But You Can't Tuna Fish" was an early album by REO Speedwagon, back in the vinyl LP era.

4 Data visualization

Before proceeding with the lab instructions we also must discuss data visualization. Recall the steeply dropping curve that depicts radioactive decay of a radioisotope. Also recall the steeply dropping curve that depicts a cable supporting a suspension bridge, from a tower to the midpoint of a bridge. They look similar, but are they? This is a question about modeling and many science and engineering problems require mathematical models for quantities of interest. Common models include linear, affine, polynomial, power, and exponential. We focus on the power and exponential models here.

4.1 Power model: $y = ax^p$

Suppose we believe two sets of data represented by variables x and y are related by the model

$$y = b\,x^p \tag{7}$$

for two constants b and p (where p need not be an integer). How can we test this hypothesis and determine b and p? Taking logarithms[2] gives

$$y = bx^p \implies \log(y) = \underbrace{p}_{\text{slope}} \log(x) + \underbrace{\log(b)}_{\text{intercept}}. \tag{8}$$

So a plot of $\log(y)$ versus $\log(x)$ is a straight line with slope $= p$ and y-intercept $= \log(b)$. Making such a plot is a quick way of testing this hypothesis and for determining p and b. This kind of plot is called a log-log plot because we plot the log of both the x and the y values.

To make such a plot in `Julia`, one option is to take the log explicitly:

```
plot(log10.(x), log10.(y), marker=:circle)
```

Alternatively we can use plot axis attributes to specify that one or both of the axes should use a log scale:

```
plot(x, y, marker=:circle, xscale=:log10, yscale=:log10)
```

4.2 Exponential model: $y = ba^x$

On the other hand, if we think that x and y are related by the model

$$y = b\,a^x \tag{9}$$

for some constants a and b, then we can plot $\log(y)$ versus x because

$$y = ba^x \implies \log(y) = \underbrace{\log(a)}_{\text{slope}}\, x + \underbrace{\log(b)}_{\text{intercept}}, \tag{10}$$

which is a straight line with slope $= \log(a)$ and y-intercept $= \log(b)$. This is called a semi-log plot because we apply the logarithm only to the vertical axis.

In `Julia`, such a semi-log plot is as simple as this:

```
plot(x, y, marker=:circle, yscale=:log10)
```

Back before computers, people made such plots by plotting data directly on log-log or semi-log graph paper that has non-uniform scaling that "takes the logarithms" for you. This paper avoided the need to use a calculator to

[2]In this class, and in `Julia`, "log" always means natural log (even though that might be "ln" on your calculator). In this class if we need a base-10 logarithm we will write $\log_{10}$ and in `Julia` we use the function `log10`.

compute logarithms, and also had the advantage that the data values themselves, not their logarithms, are on the axes. Using the `:log10` plot axis attribute preserves this advantage.

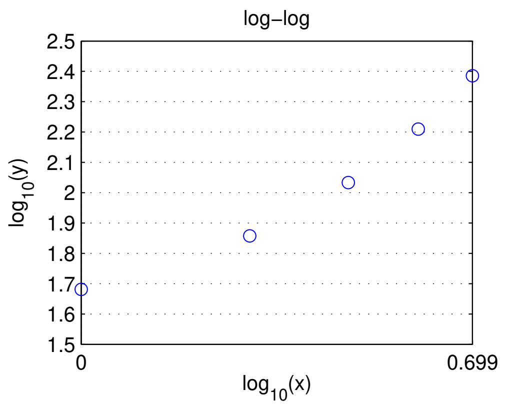
Plotting the two curves mentioned above as semi-log and log-log plots would reveal that radioactive decay follows the exponential model $y = ba^x$ where x is time and $-1/\log_2(a)$ is the half-life, whereas the suspension bridge cable curve is a parabola $y = bx^2$ which is a power model. (This parabola should not be confused with a freely-hanging cable curve, called a catenary.)

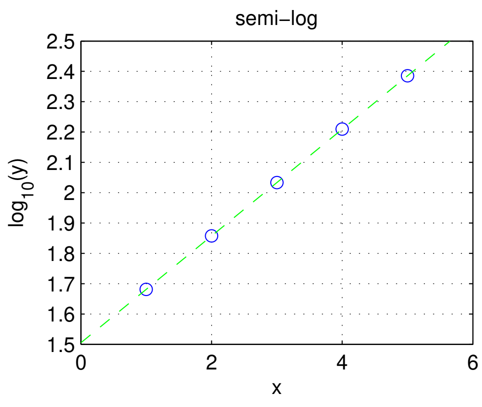
Note that both (7) and (9) involve exponentiation, but in the power model (7) we have the independent variable x raised to some *power*, whereas in the exponential model (9) the independent variable x is in the *exponent*.

4.3 Example 1: Semi-log plots

We observe the x, y pair data in the table below left. We wish to determine a model $y = g(x)$ relating x and y. The log-log plot of the data is curved (below middle), whereas a semi-log plot of the data gives a straight line (below right). This tells us that an exponential model is appropriate.
The line's slope is $(2.4 - 1.7)/(5 - 1) = 0.175 = \log_{10}(a) \Longrightarrow a = 10^{0.175} = 1.5$.
Its y-intercept (where $x = 0$) is $1.505 = \log_{10}(b) \Longrightarrow b = 10^{1.505} = 32$.
So a reasonable model that fits the given data perfectly is the "exponential" model: $y = g(x) = 32\,(1.5)^x$.

x	$y = g(x)$
1	48
2	72
3	108
4	162
5	243

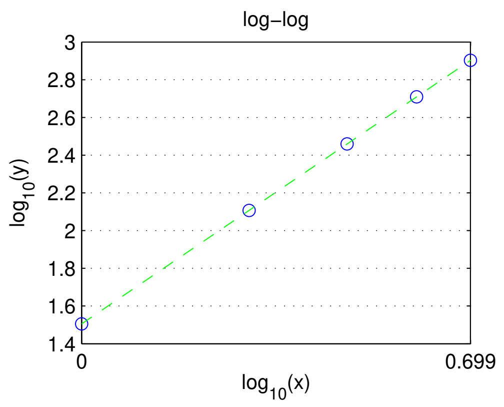


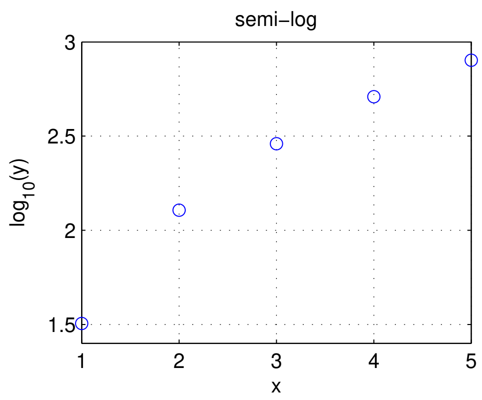


4.4 Example 2: Log-log plots

We observe the data in the table below left. We wish to determine a model for the function $y = g(x)$.
The semi-log plot of the data is curved (below right), but a log-log plot gives a straight line (below middle).
Its slope is $(2.9 - 1.5)/(0.7 - 0) = 2 = p$ (power). Its y-intercept is $1.505 = \log_{10}(b) \Longrightarrow b = 10^{1.505} = 32$.
So a reasonable model that fits the given data perfectly is the "power" model: $y = g(x) = 32\,x^2$.

x	$y = g(x)$
1	32
2	128
3	288
4	512
5	800



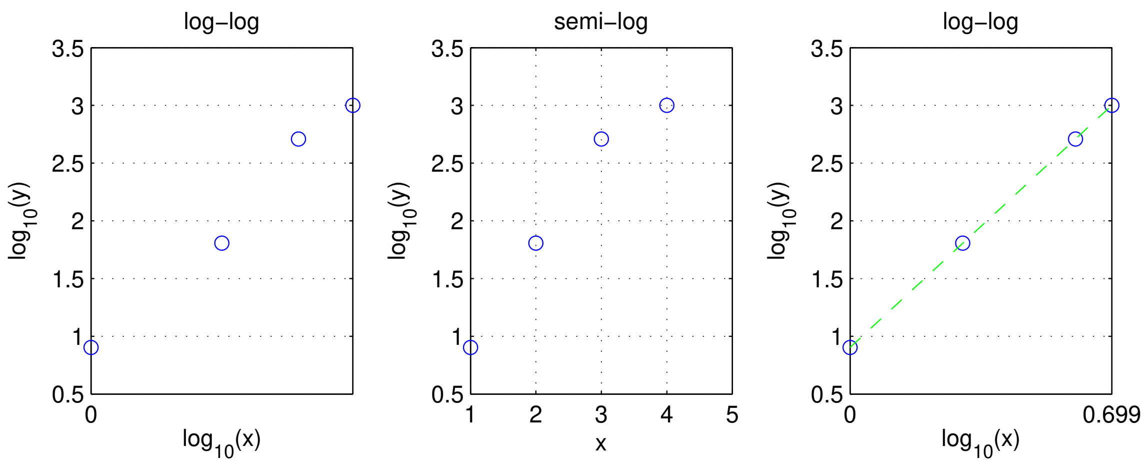


RQ Lab2.2. A particular cubic function has the formula $y = 5x^3$. If we have several x, y pairs and we plot $\log(y)$ versus $\log(x)$ we will get a line. What is the slope of that line?

4.5 Example 3: Determination of missing data

Now suppose that we know $y = g(x) = [8, 64, 512, 1000]$ for *some subset* of $x = \{1, 2, 3, 4, 5\}$. (You will see why this situation is relevant soon.) Can we still find a simple model for $y = g(x)$?

First try plotting y versus `x = 1:4` on both a log-log and semi-log scale (left and middle plots below).



The semi-log plot gives a curve, whereas the log-log plot gives a straight line with a break, suggesting that $x = 3$ is missing. To verify, plot y again versus `x = [1, 2, 4, 5]` on a log-log scale. This yields the plot on the right above, a straight line. So a reasonable model is $y = g(x) = bx^p$. The slope is $(3.0 - 0.9)/0.7 = 3 = p$, and the intercept is about $0.9 = \log_{10}(b) \implies b = 10^{0.9} \approx 8$, so a reasonable model is $y = g(x) = 8x^3$. To verify that we computed everything correctly, substitute $x = 1, 2, 4, 5$ into the model $y = g(x) = 8x^3$ and you will get the values $8, 64, 512, 1000$ that match y where we started.

Next we apply all of these tools to music signals.

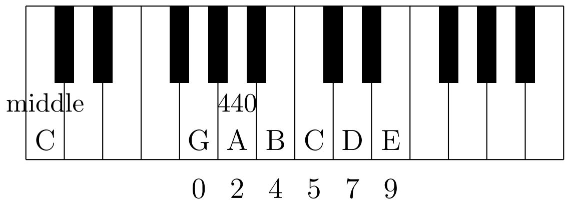
4.6 Julia tips

- In Julia, `log` means natural log, *i.e.*, log base e, often denoted "ln" on calculators.
  For base-10 log, use the function `log10()` in Julia.
- In Julia, `exp(x)` computes e^x.
  You must use broadcast to apply functions like `log` and `log10` and `exp` to every element of a vector.
  To see this, type `log10.([100, 1000, 0.1])`
  Compare to what happens if you try `log10([100, 1000, 0.1])`
- In Julia, `10^x` computes 10^x if `x` is a scalar.
  If `x` is a vector $[a\ b\ \dots\ z]$ then `10 .^ x` computes the vector $[10^a\ 10^b\ \dots\ 10^z]$
  To see this, type `10.0 .^ [-1, 2, 1]`
  Try typing the following to see what happens: `10 ^ [-1, 2, 1]`
  (That one extra dot `.` makes a big difference!)

RQ Lab2.3. Look at the equation (in Cartesian coordinates) for a catenary on Wikipedia. For that function, which kind of plot will make a straight line?

4.7 Missing frequencies in “The Victors”

The following diagram may help explain part 5.2d below.



Missing: 1 3 6 8 10 11 12 13 ... -1 -2 -3 ...

5 Lab 2: What you must do

Make a google doc report that includes all plots and discussions and computed values like `f0` for the sections below.

5.0 Frequency measurements for a single sinusoid [8]

As usual, start with the required packages. Also modify a couple plotting defaults.

```
using Plots, MAT, Sound; default(markerstrokecolor=:auto, label="")
```

We also need the `mean` function from the Statistics package (that comes included with `Julia`).

```
using Statistics: mean
```

(a) [1] Make a sinusoidal signal and plot it as follows:

```
S = 8192; t = (0:999)/S; x = 3 * cos.(2*pi*64*t);
plot(t, x, marker=:circle, xlabel="t [sec]", ylabel="x(t)")
```

(b) [1] Use the arccos formula (6) to determine the frequency (in Hz) of this sinusoidal signal as follows:

```
n = 7; f0 = (S/2/pi) * acos((x[n+1] + x[n-1]) ./ (2*x[n]))
```

(c) [2] Explain the result of part (b) in light of part (a).

(d) [1] Repeat (b) with `n = 33` instead of `n = 7`.

(e) [2] Explain what went wrong in (d). Hint: examine `x[32:34]`

(f) [1] Repeat (b) with the range `n = 7:11` instead of the scalar `n = 7`.
Hint. Here you must use broadcast with `.` in three places. One of those three places is that you must replace `x[n+1]` with `x[n .+ 1]` because now `n` is an array and adding a scalar to each element of an array requires broadcast. Type `n .+ 1` to see how broadcast works with a range type.

5.1 Frequency measurements for a "song" [15]

(a) Download "`victors_tone.mat`" from the `data` folder on Canvas.
Type `file = "victors_tone.mat"`
You may need to prefix it with a path depending on where you put the file, something like this:
`file = "/Users/fessler/Downloads/victors_tone.mat"`
Type `x = vec(matread(file)["x"]); S = 8192; sound(x, S)` Recognize this?

(b) [1] Type `n = 3000 .+ (-200:200); p1 = plot(n, x[n], marker=:circle)`

(c) [1] Is it a single pure sinusoid? (Look carefully.)

(d) [1] Type `n = 2:length(x)-1`

```
f1 = (S/2pi) * acos.((x[n .+ 1] + x[n .- 1]) ./ 2x[n])
```

This attempts to apply the frequency estimation formula (6) for $S = 8192 \frac{\text{Sample}}{\text{Second}}$ to every set of three neighboring samples of the signal $x[n]$. The second line causes an error message. Speculate why it fails.

(e) [1] Type `c2 = (x[n .+ 1] + x[n .- 1]) ./ 2x[n]; p2 = plot(n, c2, xlabel="n", ylabel="c1[n]")`

(f) [2] Now explain why the attempt to compute `f1` was unsuccessful.

(g) [1] Speculate how this behavior could actually be helpful for partitioning this type of signal into separate tones.

(h) [1] Type `c3 = reshape([0; c2; 0], 3000, :)[2:end-1,:]`

```
f3 = (S/2pi) * acos.(c3); p3 = plot(vec(f3), ylabel="f [Hz]")
```

(i) [1] The step `c2[2:end-1,:]` eliminates the first and last numbers in each segment that are outliers because of the transition between tones. Does the frequency plot look reasonable?

(j) [1] To help understand the `reshape` and `vec` operations above, type `size(c3)` , `size(f3)` , `size(vec(f3))`
Try making the plot without `vec` using `plot(f3)` and discuss why `vec` is needed here. (Do not include this plot in the report.)

(k) [1] Examine the mean (average) frequency for each of the 26 tones in this song:
```
f4 = vec(mean(f3, dims=1)); p4 = scatter(f4)
```
You might recognize the tune from this plot, even if do you not read music.
To understand the `dims=1` argument, think again about `size(f3)`

(l) Type `using WAV`
```
wavwrite(x, "test.wav"; Fs=S, compression=WAVE_FORMAT_PCM, nbits=16);
(z, _, _, _) = wavread("test.wav")
```
These statements write the signal samples $x[n]$ in `x` to the `.wav` file, and then read the values from that file into the array `z` . A `.wav` file also stores the sampling rate S.
This version uses 16-bit compression; optional: compare the size of the original `Matlab` `.mat` file to the `.wav` file (in bytes).
Type `sound(z, S)` to listen to the signal loaded from the `.wav` file. (It should be similarly familiar.)

(m) [1] Type `c5 = (z[n .+ 1] + z[n .- 1]) ./ 2z[n];`
```
p5 = plot(n, c5, ylabel="c3[n]", title="part 1m")
```
Will the arccos method work directly here?

(n) [1] Type
```
c6 = reshape([0; c5; 0], 3000, :)[2:end-1,:]; f6 = (S/2pi) * acos.(c6);
p6 = plot(vec(f6), ylabel="f [Hz]", title="part 1n")
```

(o) [2] Now we are trying to determine the frequency, using the same formula as before, but from the signal $z[n]$ that is the version of the signal that was stored in the `.wav` file. What changed?
Type `extrema(x-z)` to see how similar `x` and `z` are.

Be sure to include all 6 subplots above in your report, preferably as a single figure labeled appropriately.

5.2 Data visualization [20]

Now we are going to use the note frequencies found in the previous part to determine the relationship between between those frequencies and piano keys. To do this we must somehow number the keys (in ascending order). For now we will take the lowest note in this version of the Victors and label it number 0, and then increment by one for each key to the right.

(a) [1] Type `f = sort(unique(round.(f4, digits=2)))` to see the 6 unique frequencies that you found in the previous part, in ascending order. Hint: the first value will be around 392 Hz.
We round the values to 2 digits for ease of display and because the remaining digits are unlikely to be meaningful.

(b) [1] Type `p1 = scatter(f, ylabel="f [Hz]", title="5.2b")`

(c) [1] Type `p2 = scatter(log10.(f), ylabel="log10(f)", title="5.2c")`

(d) Neither of these plots look linear because this song does not use all possible notes so there are "missing frequencies." The person who played this song knows which notes were used, so here we give you that data. (Although you could figure it out from Section 4.5.)
Type `key = [0, 2, 4, 5, 7, 9]`

(e) [1] Type `p3 = scatter(key, f, marker=:square, title="5.2e")`

(f) [1] Type `p4 = scatter(key, log10.(f), marker=:diamond, title="5.2f")`

(g) [1] Discuss whether it would also be reasonable to make a log-log plot of `f` vs `key`.
If it is, make the plot. If not, discuss why not.

(h) [6] One (and only one!) of the preceding two or three plots has points that lie along a line. Determine which one, and then use the ideas from Section 4 to determine a mathematical formula that relates key k to frequency f. Your formula should looks something like $f = 3k^2$ but of course not that particular formula.

(i) [2] Your formula will be valid for all keys and notes, not just the 6 notes used in this particular song. Give numerical values for 12 of the frequencies (one octave's worth), whether they appear or not.
Hint, if your formula were $f = 3k^2$ then in `Julia` you could see an octave's (12 notes) worth by typing:
`k=0:11; f = 3 * k.^2`

(j) [3] Use `p5 = scatter(0:11, ???)` to make a scatter plot of the 12 frequencies you found. (Hint: the values will not be on a straight line.) Label axes!

(k) [3] Use `p6 = scatter(0:11, log10.(???))` to make a semi-log plot of the 12 frequencies you found. (Hint: the values should be on a straight line.) Label axes!

5.3 Lab report

Make a brief report that includes your 3 pages of plots and your concise answers to the questions above, including your list of frequencies (both measured and inferred). The audience will be your lab instructor, not your discussion instructor. Save your report as a pdf file and upload it to Gradescope. Although we recommend using Google Docs, you may use any editor as long as you export it in pdf format for uploading to Gradescope. To keep on schedule, you should finish this before the start of next week's lab. If you were unable to finish it during this week's lab, then seek help in office hours or via Piazza (if needed) before the due date.

5.4 Next week

Now that you know what pure musical tones are, and their frequencies, you will write a `Julia` program for a simple musical tone synthesizer with a `Julia` graphical user interface (*i.e.*, an on-screen keyboard) that generates musical tones. Then you will write a `Julia` program for a simple musical note analyzer that, when given a pure tonal musical signal, produces a `Julia` stem plot resembling musical staff notation.