Eng. 100: Music Signal Processing W24

February 15, 2024 22:18

Project 2: Touch-tone synthesizer and analyzer

1 Abstract

Now that you have acquired some tools for analyzing frequencies and using `Julia`, you will apply them to a small engineering project: touch-tone phone tones. The goals of this lab are: (1) To analyze touch-tone phone signals and determine their spectral content; (2) to use `Julia` to create a GUI *synthesizer* that functions as a touch-tone keypad and that generates the proper tones when pressed with the mouse; (3) to write a `Julia` *transcriber* program that accepts as input a touch-tone phone signal, determines the phone number, and prints it out on the screen; and (4) to analyze the effect of noise on this transcriber. In addition to applying the tools you have acquired, this project also helps prepare you for the final project.

2 Background

Touch-tone phones create a multi-frequency tone when a button is pressed. That tone is sent over a phone line (or wirelessly) as a signal. The goal of this project is to "reverse engineer" the touch-tone system and build your own `Julia`-based touch-tone synthesizer and transcriber from scratch. The only things you are allowed to use are: (1) the techniques you have learned so far in Engin 100; and (2) the 12 signals corresponding to each button in the keypad of your touch-tone or cell phone, as given in the file `project2.wav` on Canvas.

The correlation method used in the transcriber in Section 3.3 below uses matrix-vector and matrix-matrix multiplication. Wikipedia has a thorough description of matrix multiplication. Many students will need to review the definition to understand the correlation method.

3 Project 2: What you have to do

The results of this project will be three jl-files, one implementing a touch-tone synthesizer, and one implementing a touch-tone transcriber, and one that evaluates that transcriber's robustness to noise. You also must demonstrate to your lab instructor that they work.

You will use the first program to write a signal to a file `touch.wav` using the command `wavwrite` in the WAV package, as well as to produce a sound using `soundsc` from the `Sound` package. Then your transcriber will use `wavread` (see Lab 2) to read in and then decode the signal stored in `touch.wav`. You will also use the spectral analysis techniques that you have learned to analyze the touch-tone signals in the first place, like you did with the musical tones in Lab 2. Finally, you will investigate the effect of noise in the touch-tone signal on your transcriber.

We encourage your team to create a *private* repository on `https://github.com` for your code, and to use git software for collaborative code editing. An effective approach is the github-flow process, and a useful tool is the (free) GitHub Desktop app. None of this is *required* for Project 2, but these are industry standard tools and starting to learn them here will help your team be more efficient for Project 3. A private github repo also ensures your code is backed up. Here is a github tutorial for getting started.

3.1 Touch-tone signal analysis

Use the techniques you have learned to analyze the 12 signals generated by the 12 keys on a touch-tone phone keypad. Download `project2.wav` from the Canvas; this file contains, in succession, the signals produced by pressing keys "1,2,3,4,5,6,7,8,9,*,0,#" (in that order) for half a second each (total duration=6 seconds). Use

the `wavread` function to read the signal and sampling rate from this file. All you will be told here is that you have learned the tools necessary to do this. Go to it!

3.2 Touch-tone synthesizer

Write a `Julia` program (and save it as an jl-file) that:

- Creates an on-screen keyboard using a sequence of `Gtk` functions that resembles the 12-key keypad on a touch-tone phone or a cell phone (similar to what you did in Project 1);
- Produces the appropriate sound, lasting half a second, when pressed by clicking the mouse on it;
- Writes the signal to a file `touch.wav` using `wavwrite` for subsequent decoding by your transcriber.

3.3 Touch-tone transcriber

Write another `Julia` program (and save it as an jl-file) that does the following.

- Read a touch-tone signal (and sampling rate) produced using the program above and that was stored in `touch.wav` using `wavread`.
- Determine which key was pressed for each signal.
- Print out on the screen (at the `Julia` REPL) the phone number that the signal represents (without the "-" in 123-4567).
- Your transcriber need *not* be able to handle the "*" or "#" keys (these are not part of a phone number).

Hints

- You *could* use `abs.(fft())` to look for peaks in the spectrum of each digit signal, but it is *much* faster to look *only* for those frequencies (`F1` , . . . , `FM`) that you expect to see in the signal.
- Given a vector of sampled signal `x` where `N=length(x)` and `S` = sampling frequency, and `freqs` a vector of frequencies of interest, use the matrix-product implementation described in lecture:

```
c = cos.(2π/S * freqs * (0:N-1)') * x # check the size!
s = sin.(2π/S * freqs * (0:N-1)') * x
corr = c.^2 + s.^2
I = argmax(corr)
```

- The last statement determines the *location* (*i.e.*, index) `I` of the maximum of `corr`.
- Note the use of the transpose operation `'` above. You should study what is the size of the array of signals generated by the command `z = cos.(2pi/S * freqs * (0:N-1)')`
- Here is an equivalent alternative approach that uses `dot` product and a `Julia` comprehension loop:

```
using LinearAlgebra: dot
c = [dot(cos.(2π/S * f * (0:N-1)), x) for f in freqs] # check the size!
s = [dot(sin.(2π/S * f * (0:N-1)), x) for f in freqs]
corr = c.^2 + s.^2
I = argmax(corr)
```

  `Julia` comprehensions were described in Section 5.3 of the `Julia` tutorial. If needed for this project, review that section or the `Julia` manual section on **comprehensions**.
- See the `Julia` tips section below for more about the `argmax` command.
- You can `reshape` a signal `x` consisting of multiple tones into a 2D array of size number of samples × number of tones to make a matrix where each column is one tone, and then perform correlation one column at a time using a `for` loop or by using `eachcol` (see below) with a `Julia` comprehension.
- If you have an array `B` where you want to replace all the values that are 11 with the value 0, use `B[B .== 11] .= 0`
  There are many ways to do this part so not all of you may want to use this hint.

3.4 Transcriber test with mystery signal

Download the test signal `project2test.wav` from Canvas and use your transcriber to determine what phone number corresponds to that signal. Include that number in your report.

3.5 Transcriber robustness to noise

Now investigate the effect of noise on your transcriber, as follows.

- Add noise to the signal produced by your touch-tone synthesizer for a single key press. Use the signal for the "1" button from `project2.wav`, *i.e.*, the first 4096 samples of that signal.
  Use `randn` (not `rand`) to generate pseudo-random noise.
  Use `soundsc` to listen to the signal before and after you add noise to it for the lowest and the highest noise levels.
- Compute the Signal-to-Noise Ratio (SNR):

$$\text{SNR} = 10 \log_{10} \frac{\sum_{n=1}^{N} |\text{signal}[n]|^2}{\sum_{n=1}^{N} |\text{noise}[n]|^2}.$$

  This is the noise level figure-of-merit.
- For each of 10 different noise levels (multiply `5*randn` by successively larger numbers), estimate the error rate by counting the number of incorrectly-decoded digits out of 100. Plot the error rate (as a percentage) versus SNR. Hint. The error rates should vary from quite large values (likely over 90%) to small values (likely under 5%). Each group may get somewhat different plots because of `randn`.

3.6 Actual phone tones (optional)

- Press your own phone keypad keys and record the audio using a microphone and the `record` function in the Sound package, as demonstrated in class with guitar sounds.
- Apply your transcriber to your recorded touch-tone phone signal.
- Does your transcriber work on this signal?
- Try to use your synthesizer to dial a number on your cell phone when held near the speaker.

3.7 `Julia` tips

To see whether a variable has a certain value or not, use an **if** statement like this:

```
a = rand(1,1)
if a < 0.5
    println("a is smaller than 0.5")
else
    println("a ≥ 0.5")
end
```

Or use the **ternary operator** with `?` and `:` like this:

```
a < 0.5 ? println("a is smaller than 0.5") : println("a ≥ 0.5")
```

To check if a variable differs from some value, then use the "not equal to" symbol `!=` like this:

```
a = 2
if a != 1
    println("1 is not equal to 2, obviously")
end
```

The `argmax` and `findmax` functions are useful when working with correlations.

Try the following: `a = [10, 20, 30, 15]`

`(big1, index1) = findmax(a)`

The result is `big1 = 30` and `index1 = 3` because the largest value in `a` is its 3rd element.

If you only need the index, then use `argmax(a)`, which returns `3` here.

The `argmax` and `findmax` functions can also work with arrays.

Try the following: `b = [10 20; 30 40; 50 15]`

`(big2, index2) = findmax(b, dims=1)`

`index2 = argmax(b, dims=1)`

In this case `big2 = [50 40]` because the largest value in the first column of $b = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 15 \end{bmatrix}$ is 50 and the largest value in the second column is 40. The second output has type `CartesianIndex` which is useful for multidimensional arrays but is a bit more complicated than we need here, so instead we simplify by combining the useful `map` and `eachcol` functions:

`index2 = map(argmax, eachcol(b))`

This code loops over each column of `b` and collects the index of that column's maximum value. It returns `index2 = [3, 2]` because the largest value in the first column is in the 3rd row and the largest value in the second column is in the 2nd row.

Here is `Julia` code that uses a comprehension with a double `for` loop to perform correlation of each column of an array `X` with sinusoids of various frequencies:

```
[dot(cos.(2π/S * f * (0:N-1)), x) for f in freqs, x in eachcol(X)]
```

RQ Proj2.1. What value is displayed by the following `Julia` commands?

`c = [3, 1, 4, 1, 5, 9, 2, 6, -10]; index = argmax(c); @show(index)`

3.8 Project 2 deliverables

3.8.1 Project 2 report

Write the results of your lab as a short technical report, and upload a pdf file to Canvas. This project will be graded on *both* the technical communications components and on the DSP / code components.

Include the following parts, along with the other components described in the TC report specifications.

- A short summary of how you determined the touch-tone frequencies.
- A diagram of the frequencies associated with each touch-tone key.
  Think carefully about what type of diagram best describes the frequencies!
- The error rate versus SNR plot.
  Think carefully about appropriate labels, including units, for this plot.
- The phone number of the test signal.

3.8.2 Project 2 code

Upload to Canvas (for grading) a zip file of your three jl-files, named
`p2_`*teamname*`1.jl` and `p2_`*teamname*`2.jl` and `p2_`*teamname*`3.jl`
for your synthesizer, transcriber, and error rate versus SNR analysis, respectively. Replace *teamname* with your assigned team name.

Clear comments in the code will improve the odds of earning partial credit. It is especially useful to comment on the *size* of arrays involved in the transcriber. To earn full credit, you must submit code for:

- a working synthesizer
- a working transcriber, including reporting the phone number of the test signal
- a working error rate script.

The code part will be graded by your lab instructor(s) and Prof. Fessler.

If you created a private git repo for this project, please add Prof. Fessler and your lab instructor(s) to its access list so that we can use it to see your work.

Do not post any of your project code publicly!