Eng. 100: Music Signal Processing
DSP Lecture 8
Project 3: Music signal processing team project

Curiosity:

- https://www.youtube.com/watch?v=Q3oItpVa9fs (sound and matter - Nigel Stanford)
- https://paris.cs.illinois.edu/demos (audio source separation)
- Stable Audio AI (generated jazz)

Announcements:

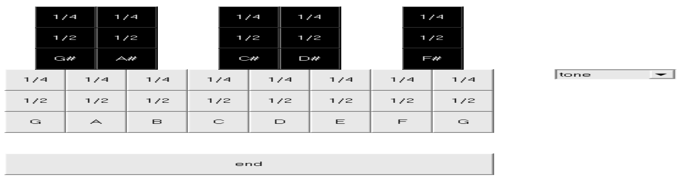
- Project 2 (report and code) due this week *(extended to Monday, but...)*
- Read Project 3 description before discussion/lab this week (Google Drive under "TC assignment details")
- Prepare P3 ideas for feedback in lab this week
- HW 4 due Friday
- Midterm course feedback
- Midterm Mar. 20; "practice" on https://w2p.eecs.umich.edu/fessler1

Learning objectives

- Understand P3 possibilities and expectations
- Learn some simple audio DSP tools: sound mixing, basic reverb
- Learn autocorrelation method for finding pitch
- Understand git purpose and basics

Outline

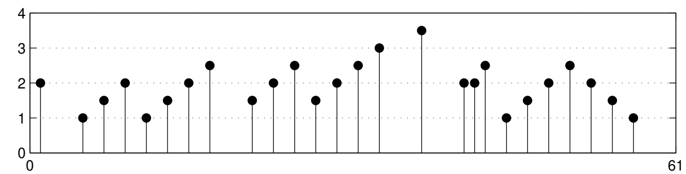
- Part 0: P3 examples
- Project 3 "classic"
  - Part 1: Illustration of a "baseline" music synthesizer



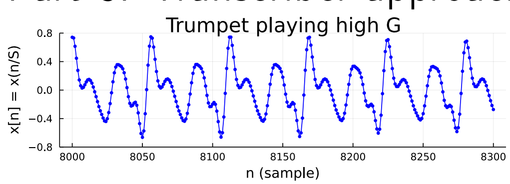
$\rightarrow$ `song.wav`

  - Part 2: Illustration of a "baseline" music transcriber

`song.wav` $\rightarrow$ $\boxed{\text{transcriber}}$ $\rightarrow$



  - Part 3: Transcriber approaches (new DSP methods)



$\rightarrow$ $\boxed{\text{pitch identification}}$ $\rightarrow$ 787.5 Hz

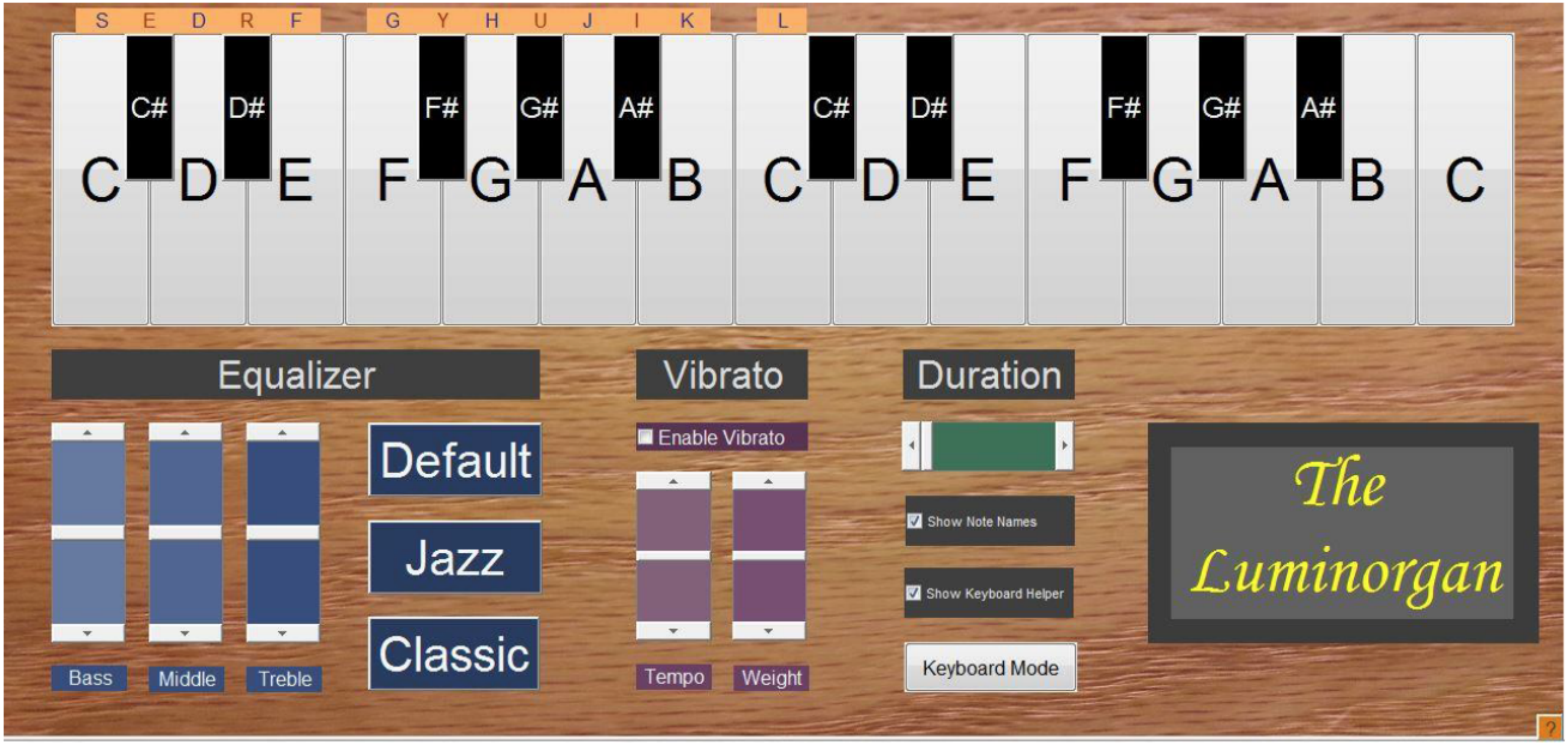
- Part 4: Git

P3 is an engineering design project, so there is considerable room for creativity and originality.
(lifelong learning...)

F13 Projects

- Spectrum modifications
  - Effects processor: flange / reverb / pitch shift
  - Effects processor: echo (delay) / flange / tone control
  - Audio equalizer (bass boost etc.)
  - Track mixer (with volume and tone controls)
  - Pitch shifter (with effects: echo / chipmunk ...)
  - Pitch and tempo shifter with phase vocoder
  - Flute autotuner
- Spectrum analyzers
  - Pitch analyzer and tuner
  - Tuner and metronome
- Synthesizers
  - Synthesizer with several instruments and tone controls
  - Synthesizer with attack/decay controls and chords
  - Synthesizer with selectable harmonics and other effects (tremolo, "ET")
  - Hammond B-3 organ synthesizer (with vibrato)
- Other
  - Audio compressor
  - Speech recognition system (spoken numbers)

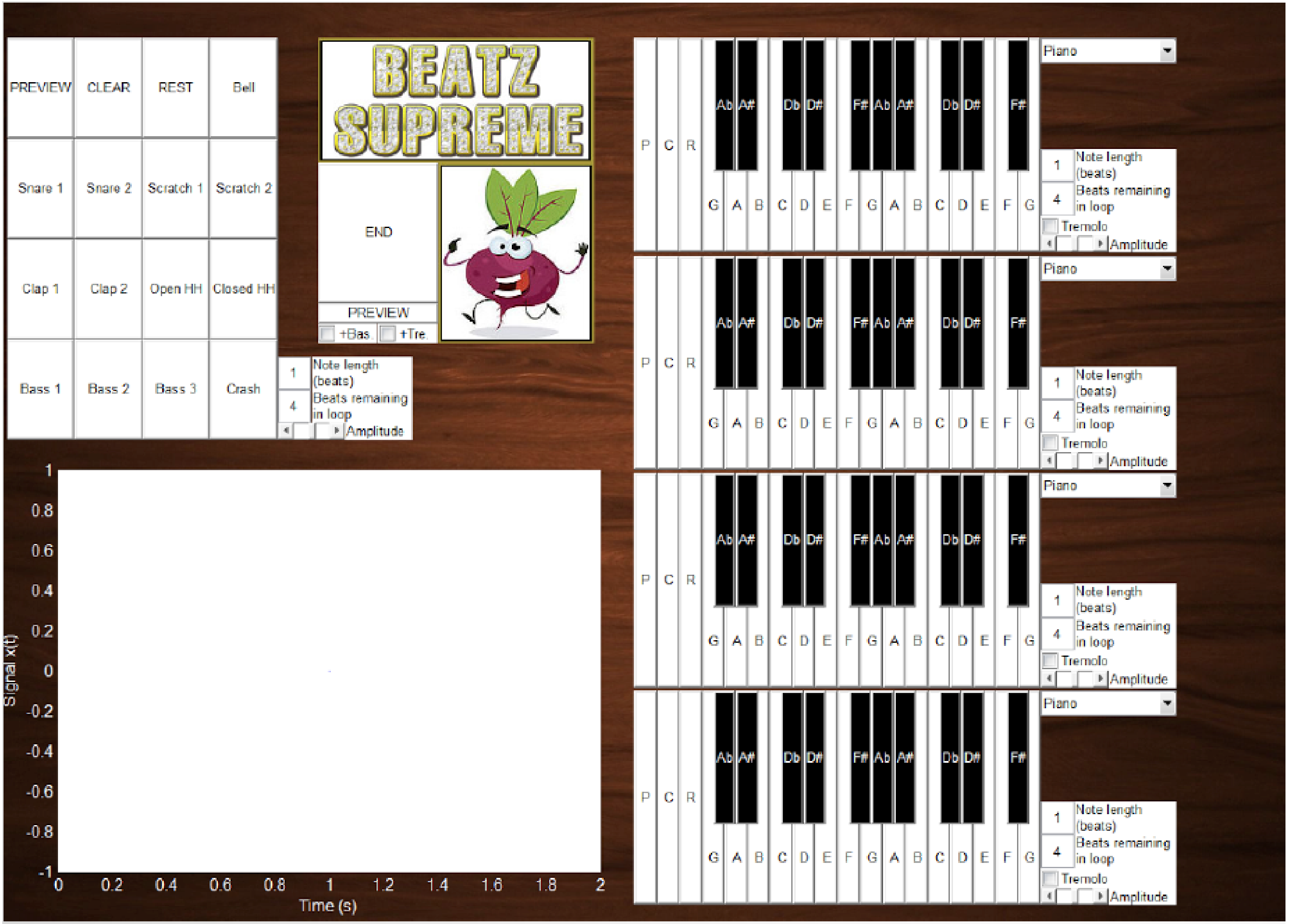
F13 example GUI



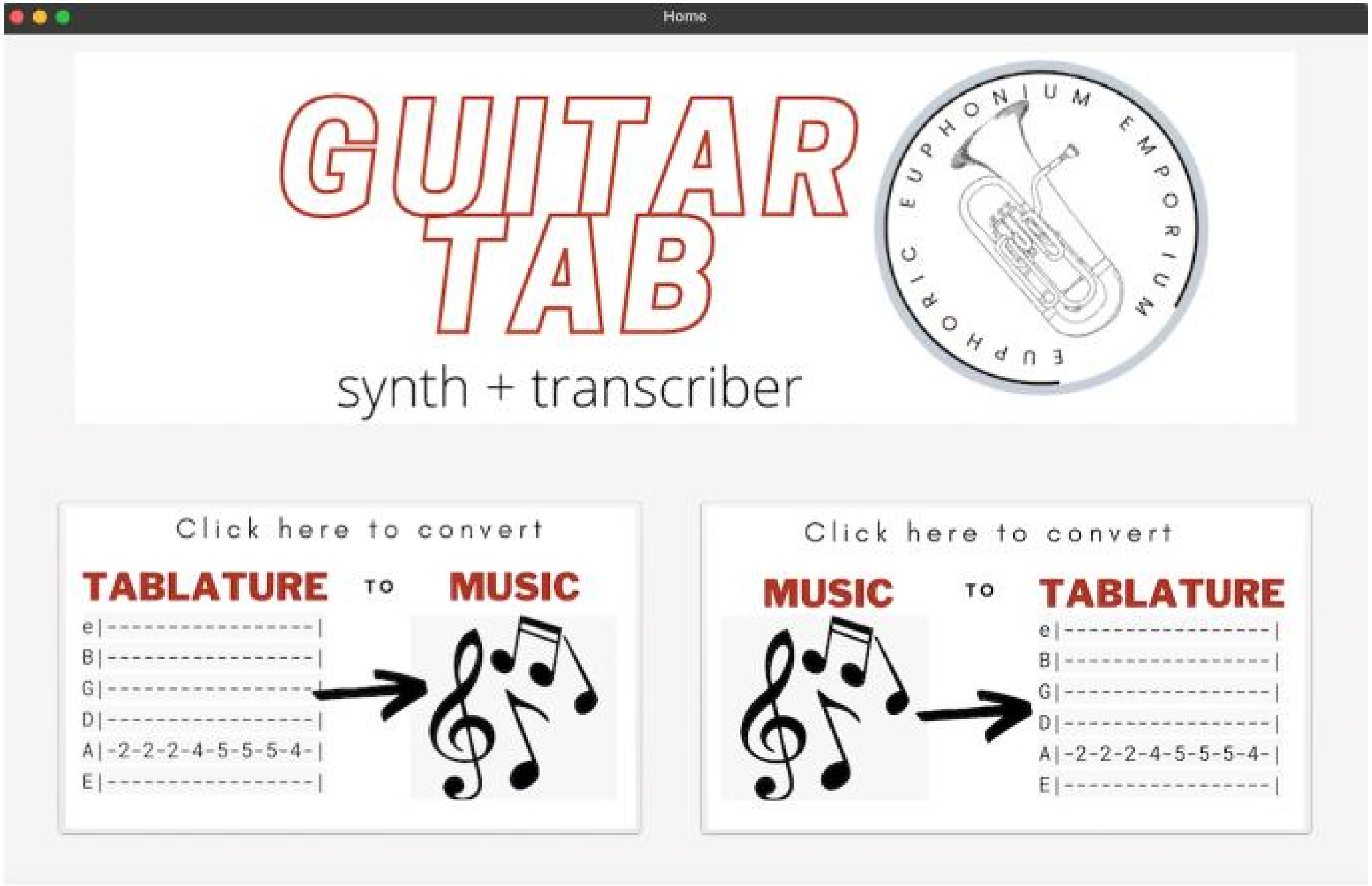
F14 Projects

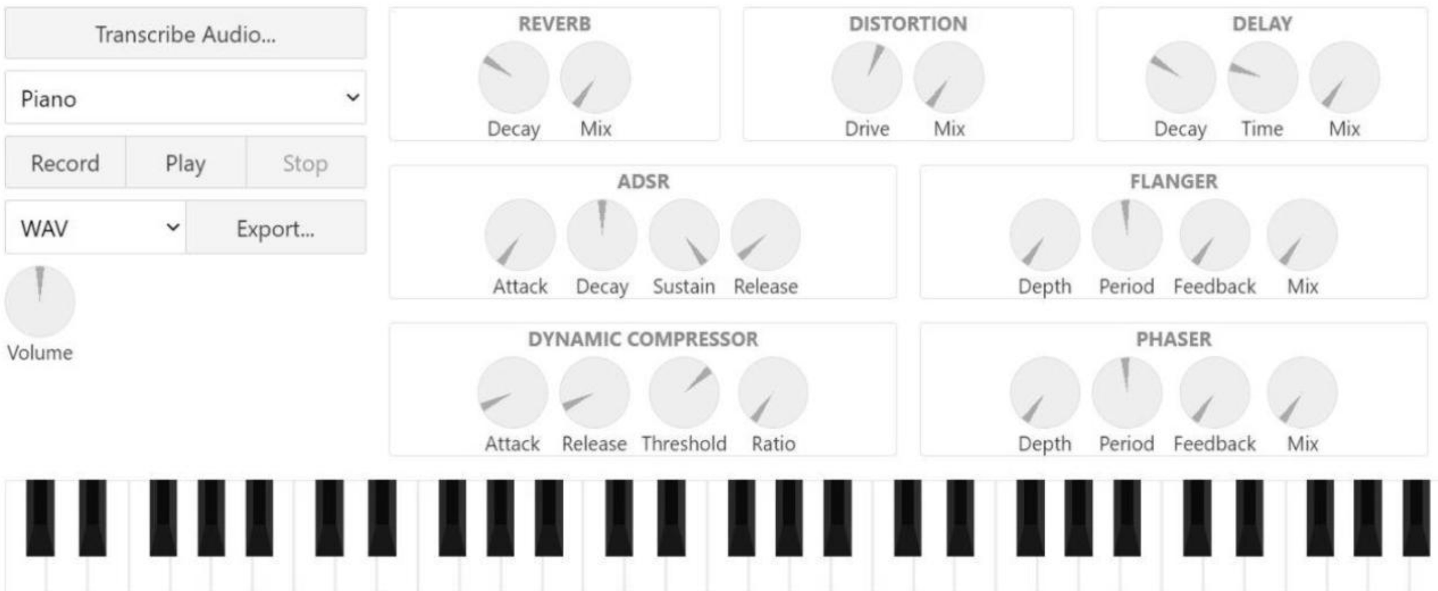
- Spectrum analyzers / modifiers
  - Pitch / tempo shifter
  - Pitch tracker / synthesizer
  - Real-time effects (flange, reverb, etc.)
- Synthesizers
  - Synth with controls
  - Synthesizer / transcriber with rests and chords
  - Synth with controls
  - Synth with 8-bit sounds and random composer / transcriber
  - Synth with keyboard input
  - Loop synth with many options
  - Polyphonic looping synth
  - Synthesizer with chords / transcriber
  - Drum machine / tempo tracker

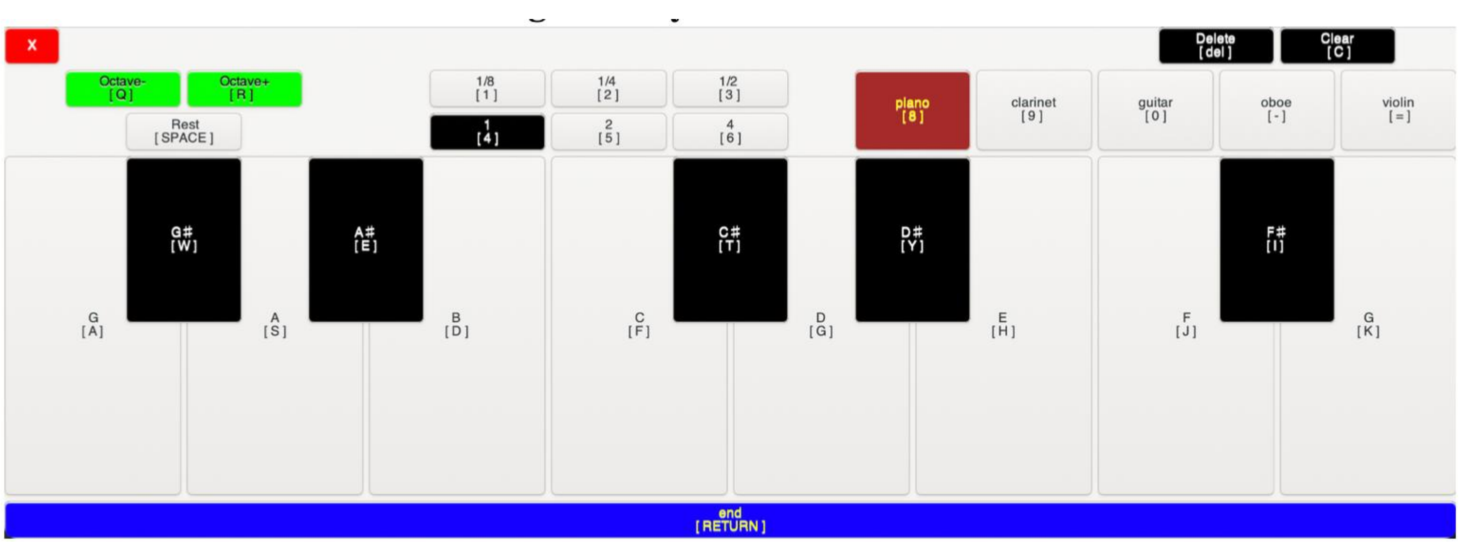
F14 example GUI

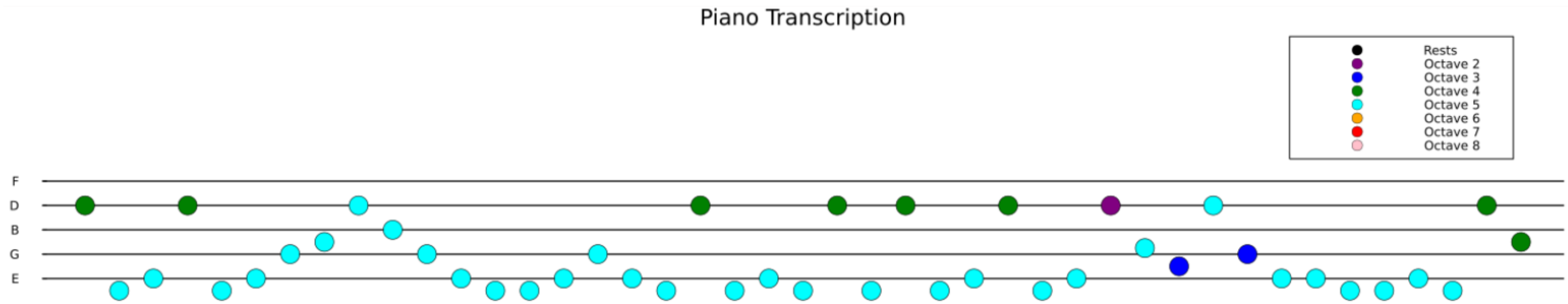


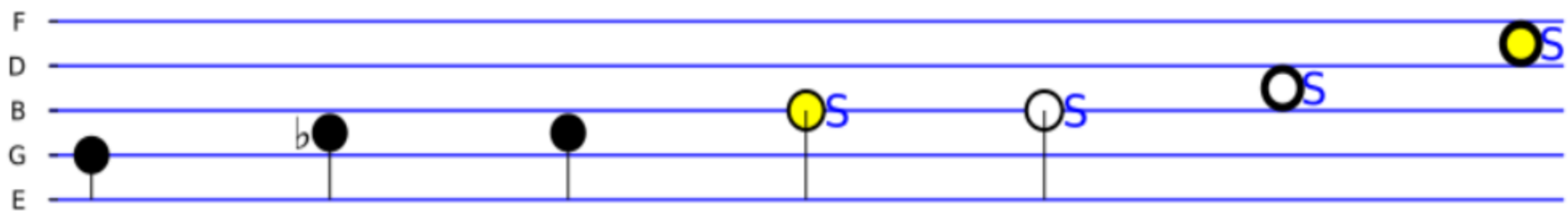
W22 project component examples

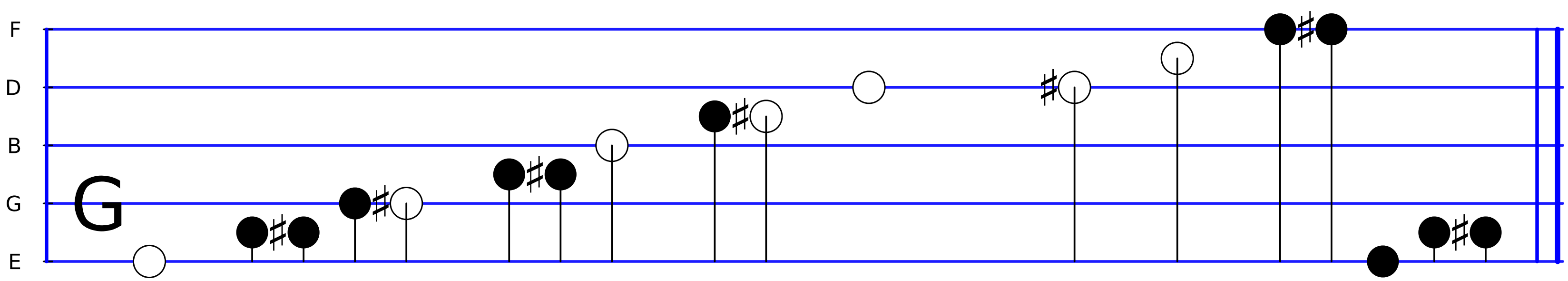



Transcribe Audio...
Piano
Record
Play
Stop
WAV
Export...
Volume
REVERB
Decay
Mix
DISTORTION
Drive
Mix
DELAY
Decay
Time
Mix
ADSR
Attack
Decay
Sustain
Release
FLANGER
Depth
Period
Feedback
Mix
DYNAMIC COMPRESSOR
Attack
Release
Threshold
Ratio
PHASER
Depth
Period
Feedback
Mix


X
Delete [del]
Clear [C]
Octave- [Q]
Octave+ [R]
Rest [SPACE]
1/8 [1]
1/4 [2]
1/2 [3]
1 [4]
2 [5]
4 [6]
piano [8]
clarinet [9]
guitar [0]
oboe [-]
violin [=]
G# [W]
A# [E]
C# [T]
D# [Y]
F# [I]
G [A]
A [S]
B [D]
C [F]
D [G]
E [H]
F [J]
G [K]
end [RETURN]


Piano Transcription
Rests
Octave 2
Octave 3
Octave 4
Octave 5
Octave 6
Octave 7
Octave 8
F
D
B
G
E


F
D
B
G
E
S
S
S
S


F
D
B
G
E
G

Project 3 "classic"
Part 1: Basic music synthesizer

Basic synthesizer overview

- Synthesize (at least) two octaves
- Instrument choices (at least 5):
  - Electric guitar
  - Trumpet
  - Clarinet
  - Tone
  - Brass section
  - Your team's own sound(s), at least one of which must use additive synthesis
- Synthesizer GUI
  - Instrument selector (*e.g.*, pull-down menu)
  - Note durations (at least 3): whole, half, quarter (think carefully about UX)
- Polyphony / mixing (optional)
  - Create individual tracks separately for different instruments.
  - Add them together in `Julia`,
    e.g., `x = 0.7 * x_guitar + 0.4 * x_trumpet`.
  - Use `soundsc` not `sound` to play the sum signal!
  - The basic transcriber is not required to handle polyphony
    (but that would be impressive and has been tackled by some past teams).

Music synthesis: Instruments

- Download `project3.wav` from Canvas (3.4 Mb).

```
(x, S) = wavread("project3.wav"); soundsc(x,S)
```

- Contents:
  - Sound snippets: Length $N = 32768$ each, $S = 44100$ Hz.
  - Electric guitar, clarinet, trumpet, tone; 13 notes each.
- Additive synthesis: Create your own instrument. Be creative!
- Brass section: Reverb (add delayed copies) of trumpet.

Music Synthesizer: Instrument selection in Gtk.jl

Many options in Gtk:

- buttons (use a lot of screen area)
- “pull-down” menus / “pop-up” menus / “combo boxes”
- ??? think carefully and creatively about UX

https://juliagraphics.github.io/Gtk.jl/latest/manual/combobox

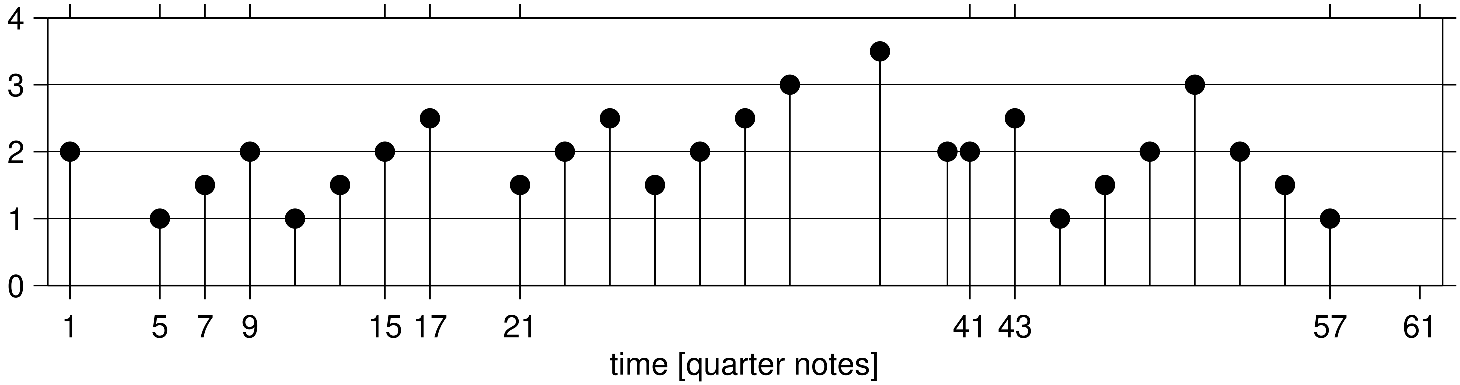
Music Synthesizer: Note duration hack

One simplifying approach uses the following note durations.

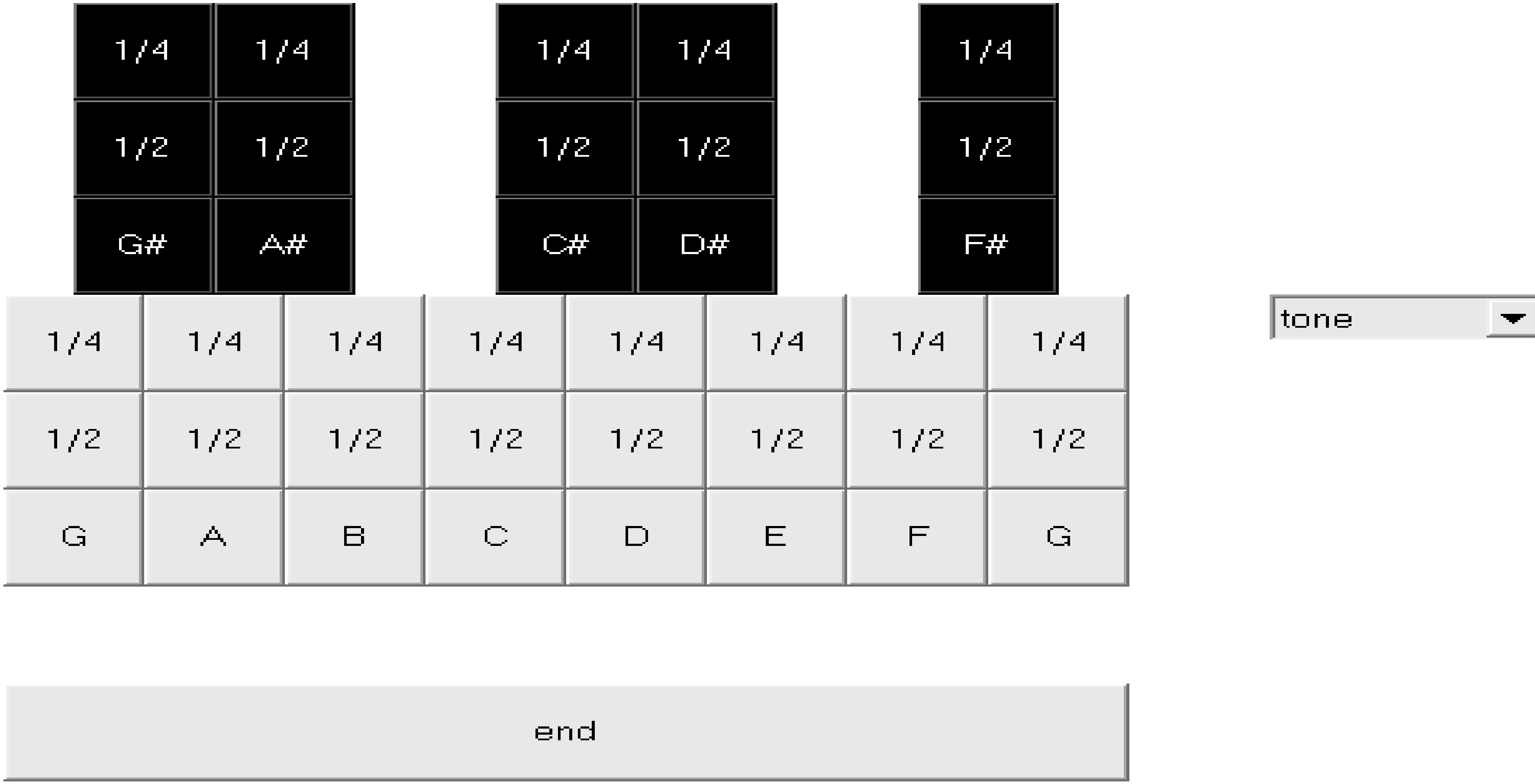
Note	Whole	Half	Quarter	1 second
Length	$32668+100$	$16284+100$	$8092+100$	$S=44100$

Simplification: the final 100 samples of each note are zeros,
to provide duration information to transcriber.
Is 100 zeros a problem musically? $\boxed{??}$

Example of (obsolete) basic music transcriber output with note duration spacing:

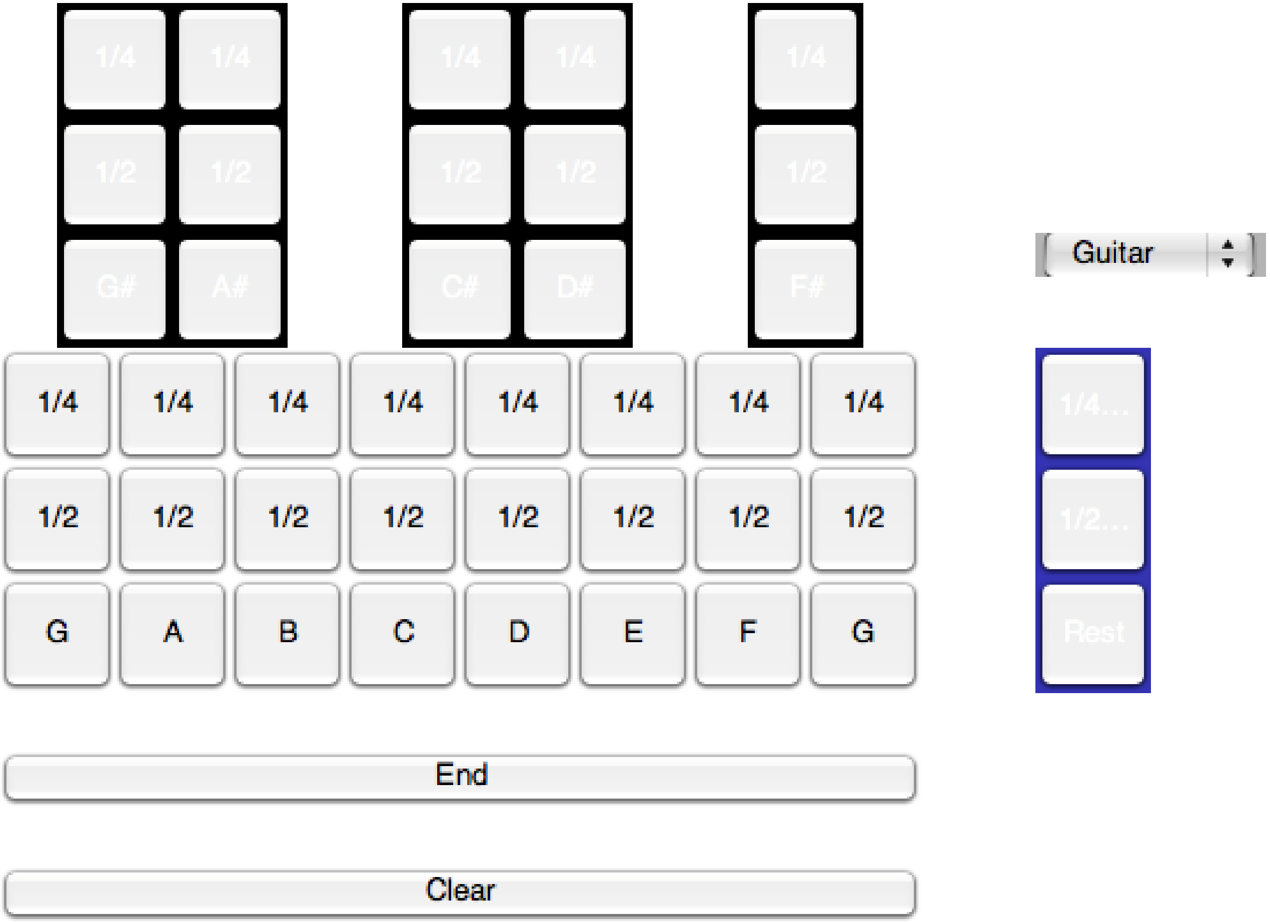


Basic music synthesizer: GUI example



Simplification: "performer" selects duration manually.
Clicking "end" button causes song to be played
and writes song to a `.wav` file. (Do not use a `.mat` file!)

Music Synthesizer: GUI example on Mac/`Matlab`



Suggestion: Start with just a few notes; add more later.
Use `for` loops and functions!

Music Synthesizer: Reverb

- To add "reverb" (reverberation, *i.e.*, echo) to a sampled signal `x` with `N=length(x)` try this:
```
y = x[1:N-2*D] + x[1+D:N-D] + x[1+2*D:N]
```
- Try delay `D` ≈ 1000. (units?)
- Use many more than 3 echoes to make it sound good.
- Random delay values can sound more realistic.
- Example. play
  Trumpet signal `x` followed by "reverberated" signal `y`
- Somewhat like a "marching band" or "trumpet section" with multiple trumpets.
  See https://github.com/nantonel/ImageMethodReverb.jl

Later lectures will discuss more advanced music synthesis techniques.

Basic synthesizer summary

Minimum features:

- At least two octaves
- Multiple instruments (at least 5)
- At least one instrument based on Fourier synthesis (explained later)
- At least 3 note durations: quarter, half, whole
- 100 samples of zeros after each note to aid transcriber is allowed

The first letter of "Basic" is B...

Part 2: Basic Music Transcriber

Basic transcriber overview

- Read `.wav` file produced by your synthesizer (or other music source).

```
(x, S) = wavread("song.wav")
```

- Generate musical staff notation as in Project 1.
  - BUT: Also must depict note duration, e.g., by separation (or symbol).
  - BUT: Must work for music sounds, not just sinusoids!
- Otherwise, same as Project 1 transcriber specs.
- Does not have to include bass clef, e.g., for guitar.
- Does not have to work for polyphonic music (stretch goal!)
- Must have some *test* process, typically like error rate versus SNR plot, as in Project 2, for multiple (at least 5) instruments.

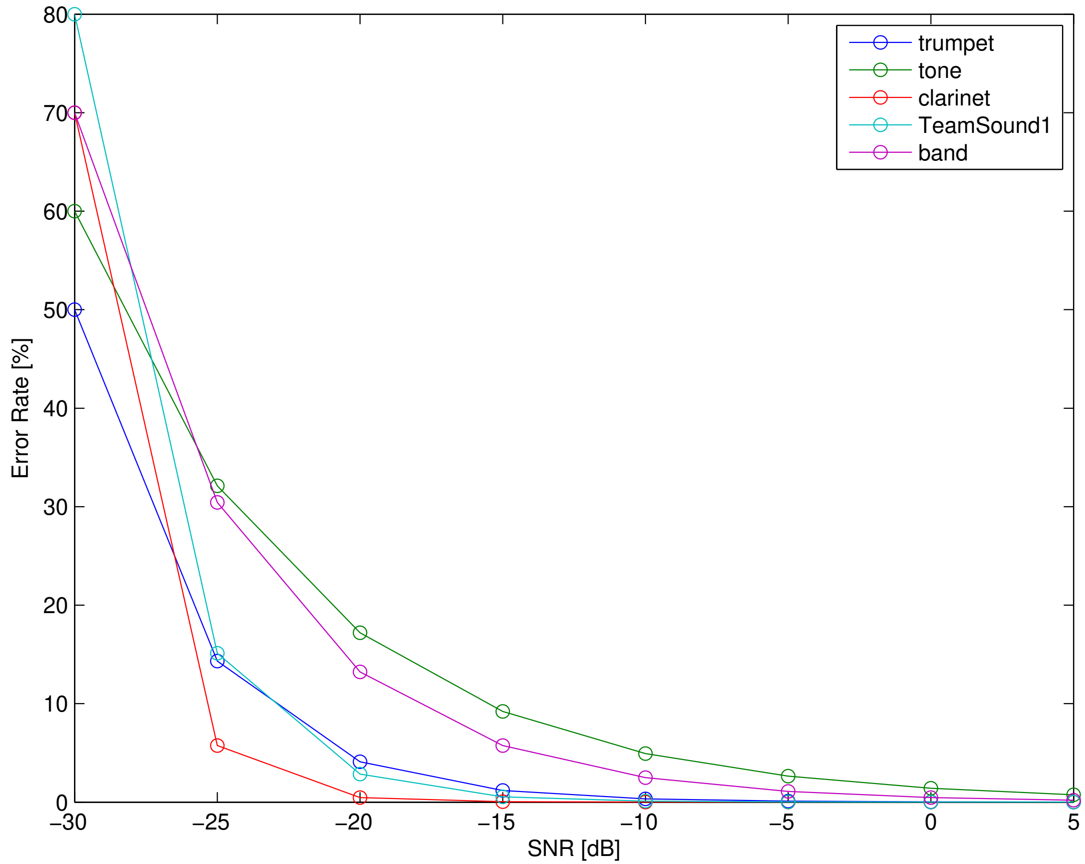
Basic music transcriber simplifications

- Output: Musical scale & notes using `stem` (Project 1) (aim higher!)
- Duration: Shown in output by separation between notes:

Note Type	Whole	Half	Quarter
Separation	3 extra spaces	1 extra space	0 extra space

- Use `reshape` (more details in later lecture)
  - columns ending in 100 zeros help indicate note lengths
  - Find indexes `I` of those columns;
  - can ignore the other columns for finding frequencies f.
  - As in Project 1: frequencies $\Longrightarrow$ MIDI value $\Longrightarrow$ table lookup to specify vertical staff position
  - Stem plot or (preferably) more advanced graphics
- Do not need display the (bass) guitar tones.
- Do need an error rate versus SNR plot, as in Project 2.

Example error rate vs SNR plot



(These are hypothetical values for illustration only.)

- Put at least 5 instruments on one plot. (Extra instruments not required here.)
- Vary the SNR enough that the error rates range from 0% to at least 50% for every instrument. Is the example plot above adequate? A. Yes, or B. No. $\boxed{??}$

Part 3: Transcriber approaches

Transcriber possibilities: Previous methods

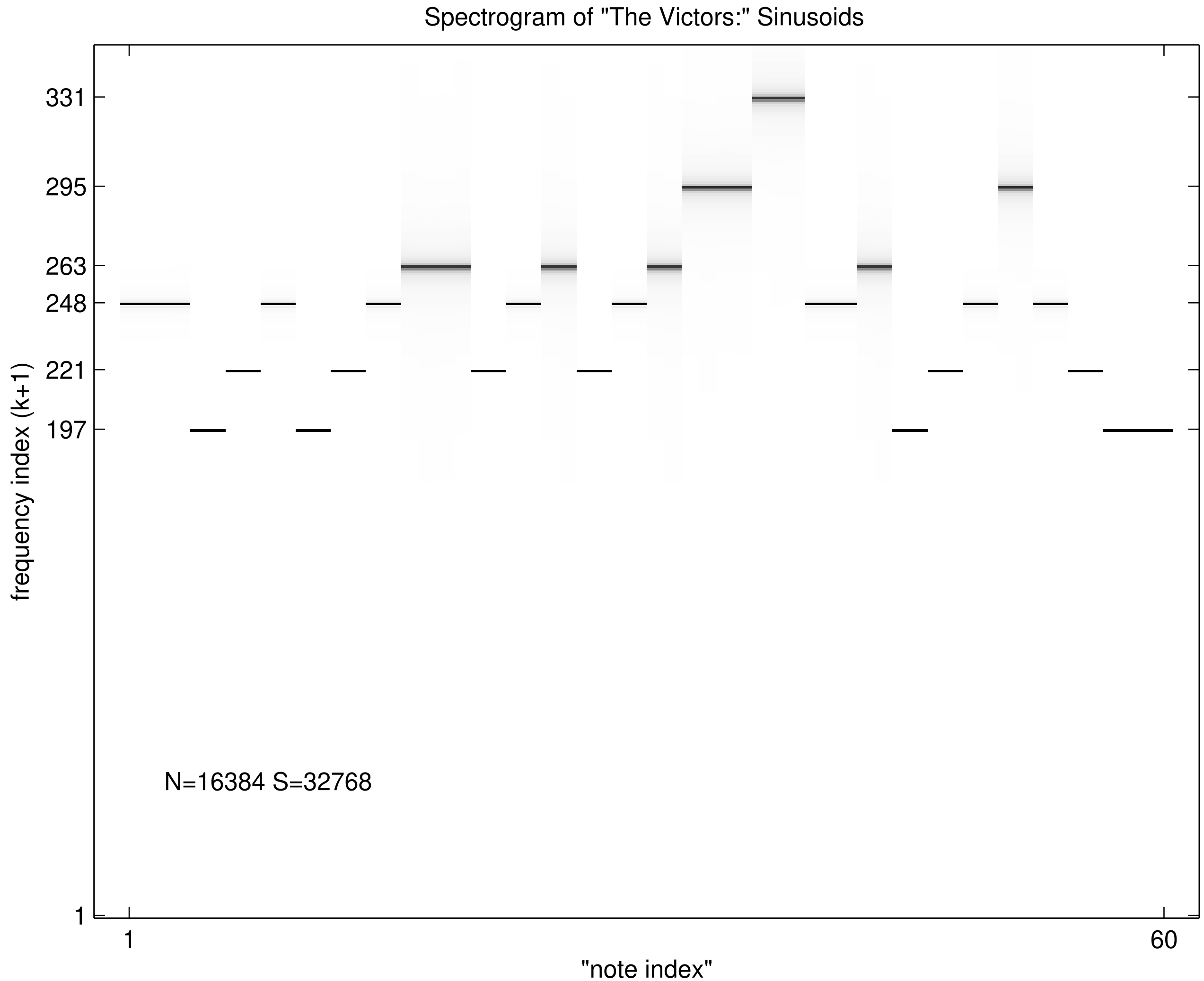
- Arccos? $\boxed{??}$
- Spectrogram: Look for peaks.
  Conceptually simple & most comprehensive and suitable for polyphony; probably requires the most advanced programming.
- Correlation using a set of candidate signals (P2).
  Probably works for your synthesizer only?
- Correlation using a set of candidate sinusoids (P2).
  Might match a harmonic rather than the fundamental?

Transcriber possibilities: New methods

- Autocorrelation of waveform segment with itself:
  $y[m] = \sum_n x[n]x[n-m]$ has sharp peak at m=period.
- Harmonic product spectrum (HPS):
  Down-sample spectra multiple ways and multiply;
  this process emphasizes 1st harmonic.
- Combinations / variations of the above methods.

All of these methods have been used previously.

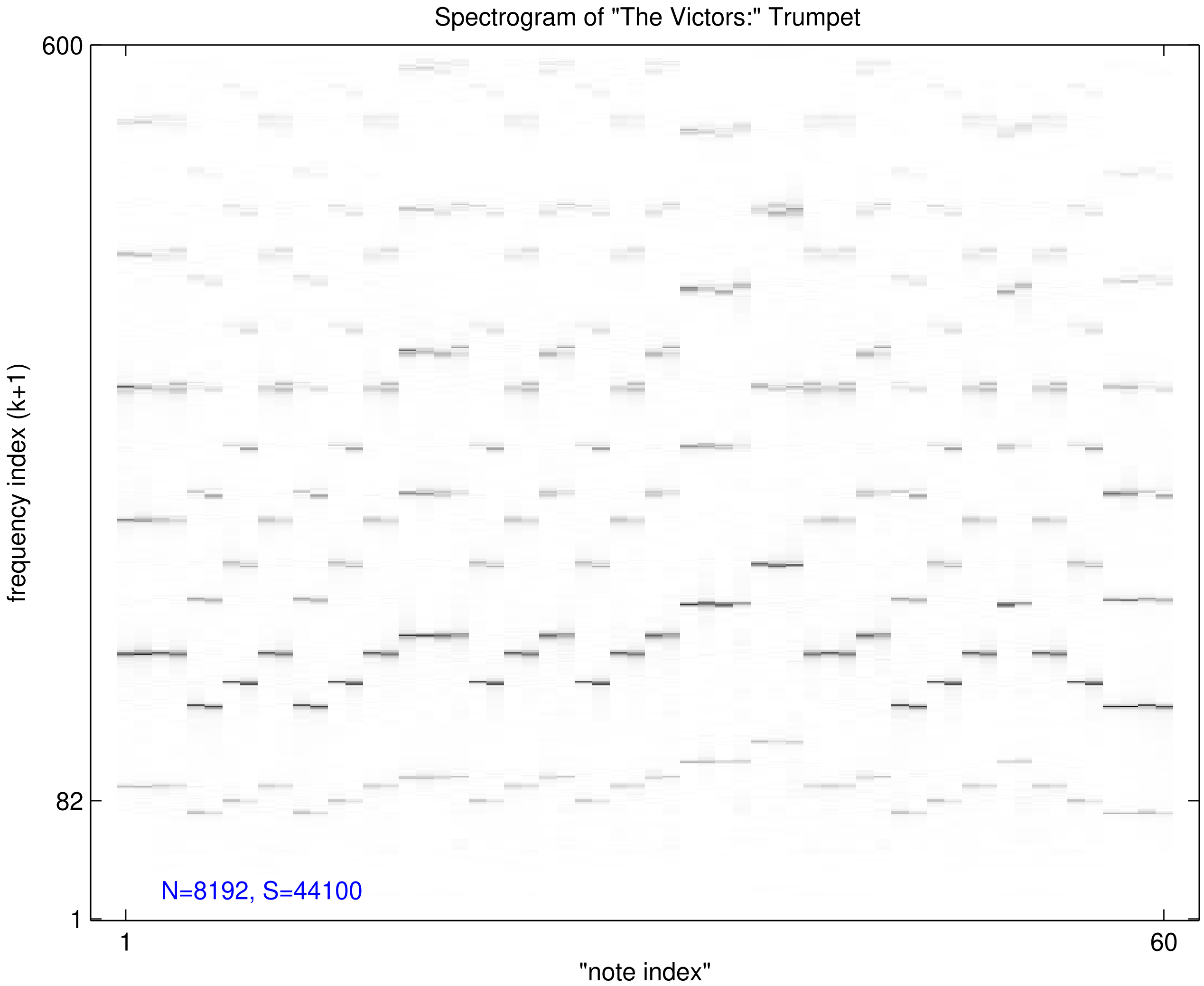
Spectrogram of "The Victors:" Sinusoids



play

Durations are evident, except for two subsequent sinusoids of the same frequency.

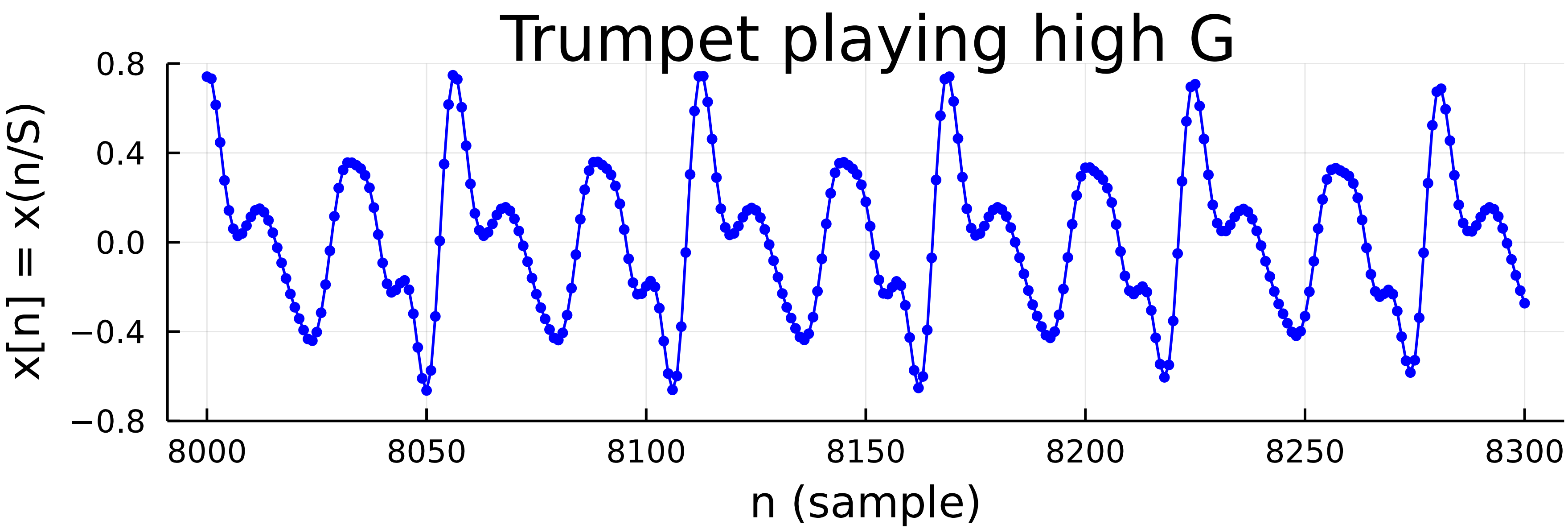
Spectrogram of "The Victors:" Trumpet



play

Finding fundamental frequency of each note here is possible but more challenging.

Example: Trumpet Waveform



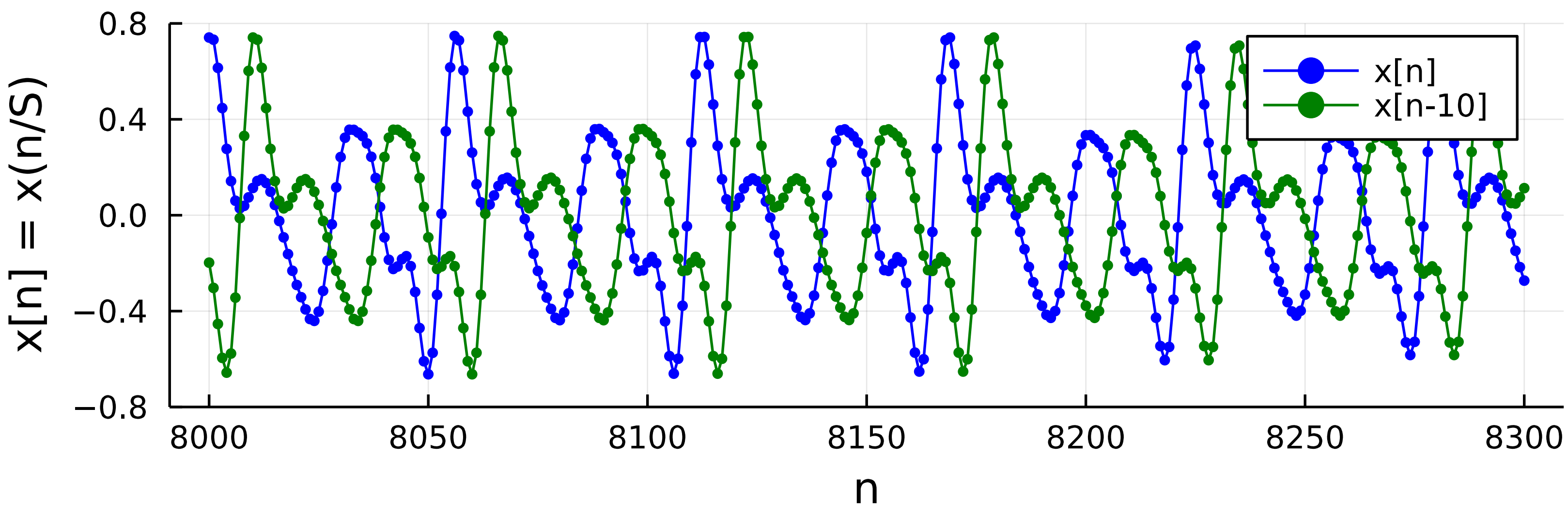
Pitch detection: Find period of periodic waveform. Given $S = 44.1$kHz.

- By eye/hand: what is frequency here? ?? [wiki]
- How can we automate? (Think about what we did by eye.)

Autocorrelation method

Recall: the correlation of one signal $x[n]$ with another signal $y[n]$ is the sum of their product: $\text{correlation} = \sum_n x[n]y[n]$.

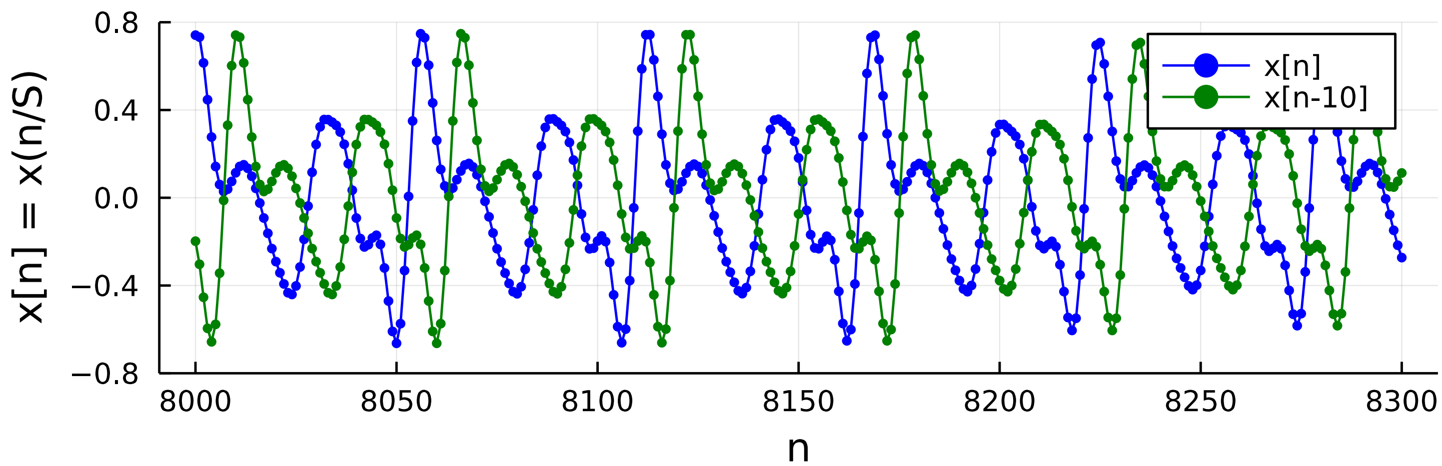
The autocorrelation of signal $x[n]$ is the correlation of $x[n]$ with a time-shifted version of itself, *i.e.*, the "other signal" is a time-shifted version of $x[n]$.

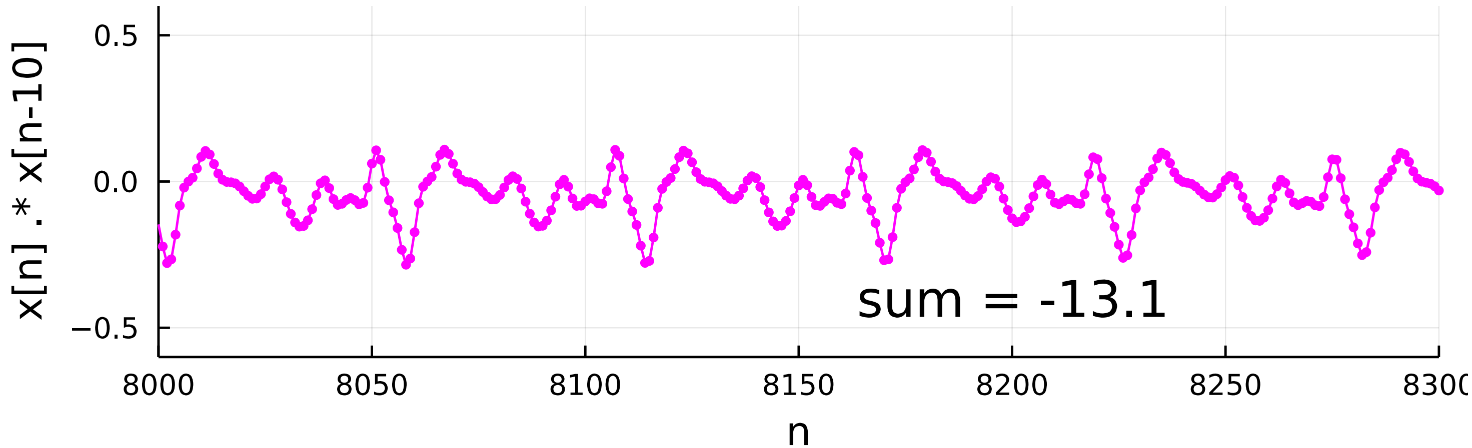


The auto-correlation value depends on the shift amount m:

$\text{autocorr}[m] = \sum_n x[n]x[n-m]$. (Units of shift m? $\boxed{??}$)

Autocorrelation illustrated

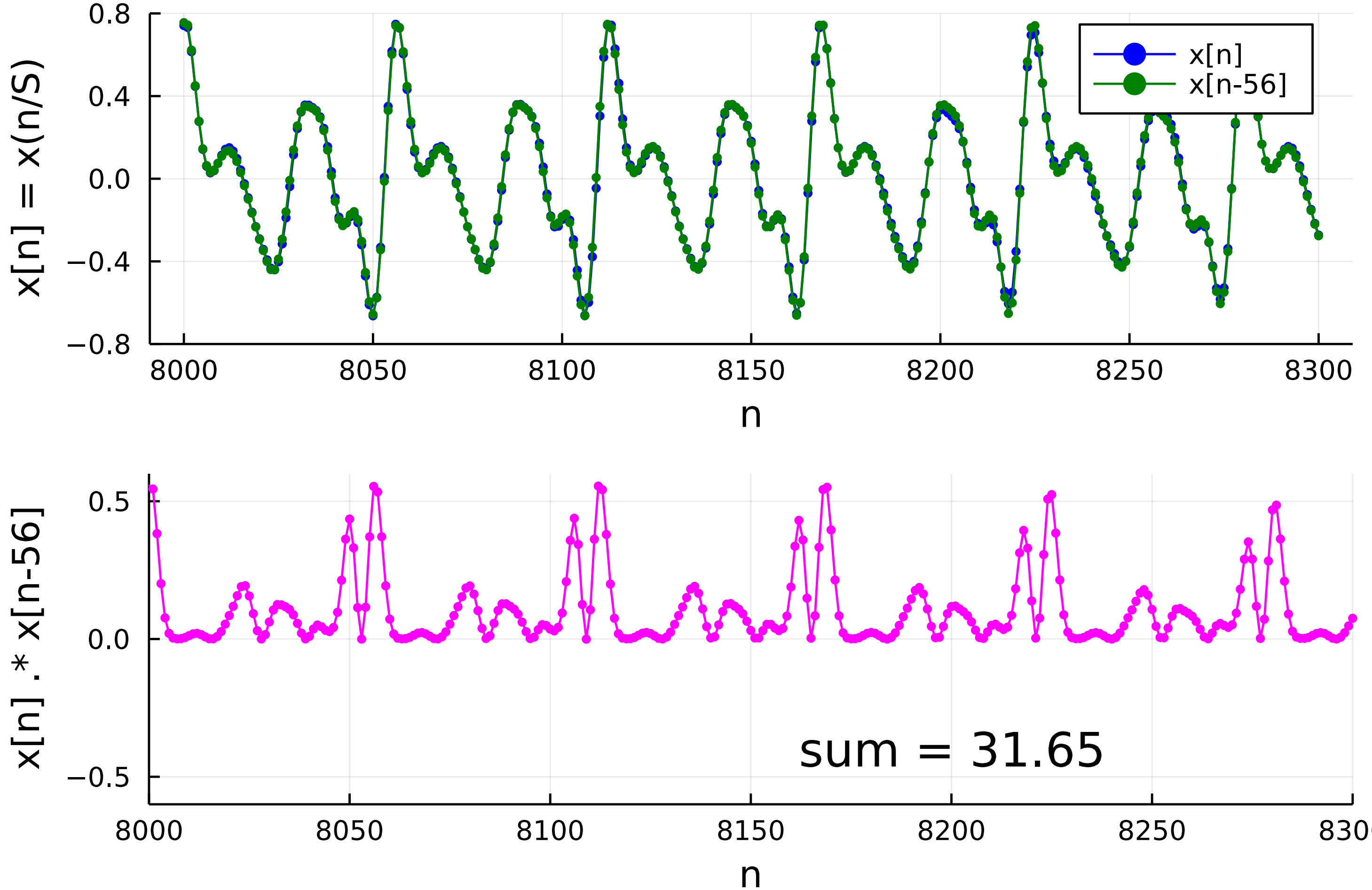




Given `x` $= [x[1]\ x[2]\ \ldots\ x[N]]$, $\quad \text{autocorr}[10] = \sum_{n=11}^{N} x[n]x[n-10]$

In Julia: `sum( x[11:end] .* x[1:end-10] )`

Autocorrelation illustrated



$\text{autocorr}[m] = \sum_n x[n]x[n-m]$ is large when shift m = a period.

For which shift is autocorrelation largest? ??

Autocorrelation properties

Cauchy-Schwarz inequality:
Maximum value of autocorrelation is $\sum_n x^2[n]$, *i.e.*,

$$\mathbf{autocorr}[m] = \sum_n x[n]x[n-m] \le \sum_n x^2[n].$$

For which m does $\mathbf{autocorr}[m]$ equal that largest value? ??

Fast way to compute (normalized) autocorrelation for all shift values m:

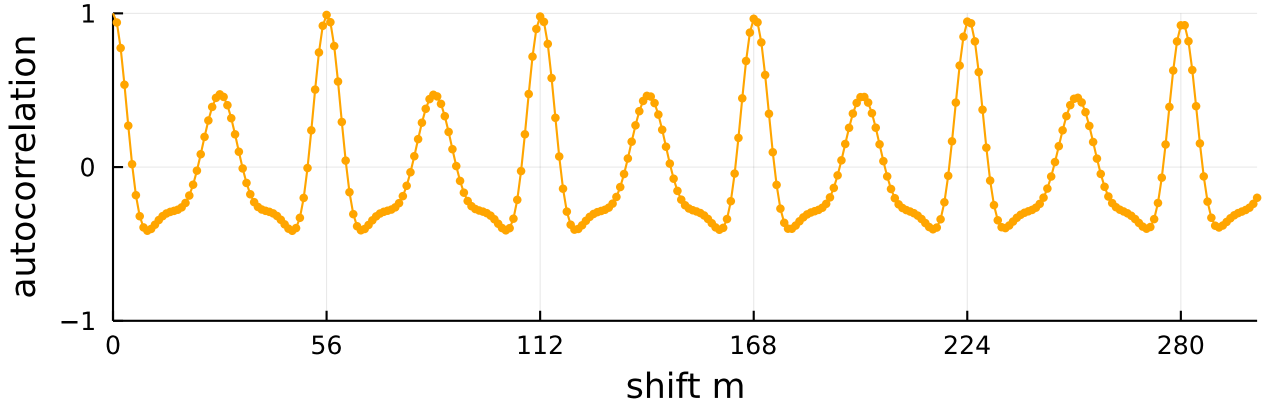
```
using FFTW: fft, ifft
autocorr = real(ifft(abs2.(fft([x; zeros(size(x))])))) / sum(abs2, x) # normalize
```

With this normalization, autocorrelation values are between -1 and 1.

(Take EECS 351 to learn why these FFT (!) commands yield autocorrelation values.
Related to Wiener–Khinchin theorem.)

Example: Trumpet autocorrelation / autocorrelogram

A plot of *all* autocorrelation values is an autocorrelogram:




```
using FFTW: fft, ifft
autocorr = real(ifft(abs2.(fft([x; zeros(length(x))])))) / sum(abs2, x) # normalize
plot(0:length(autocorr)-1, autocorr, marker=:circle, markersize=3, color=:orange)
```

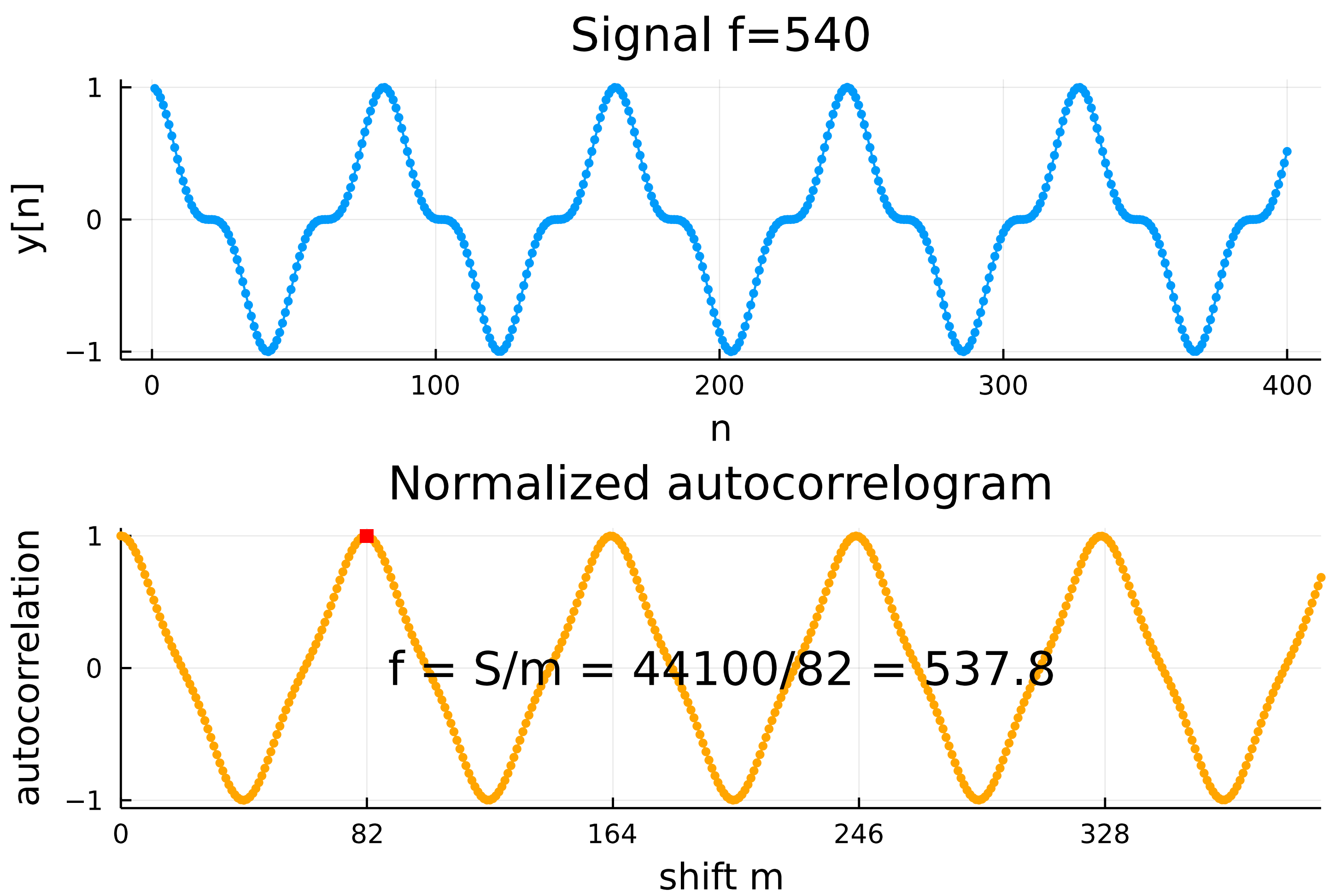
To find frequency use $f = 1/T$ where $T = m\Delta = m/S$ so $\boxed{f = S/m}$ for $m > 1$.

Highest peak always at $m = 0$ (array index `1` in `Julia`)

Second largest peak away from zero at index $= m = 56 \implies T = \dfrac{m}{S}$

$$\implies \text{Frequency} = \frac{1}{\text{period}} = \frac{S}{m} = \frac{44100 \text{ Hz}}{56} = 787.5 \text{ Hz}. \qquad \text{(high G)}$$

Vocal demo

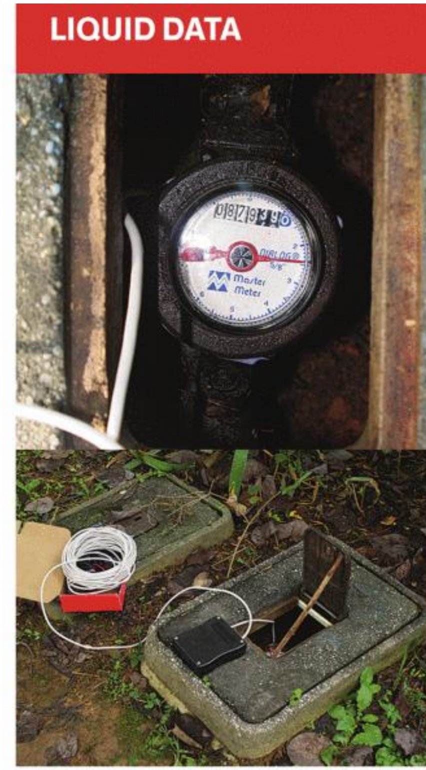


Live demo (automatic!?) with singing vowel sound (AEIOU)

```
l/fig/p3/demo-autocorr1.jl
```

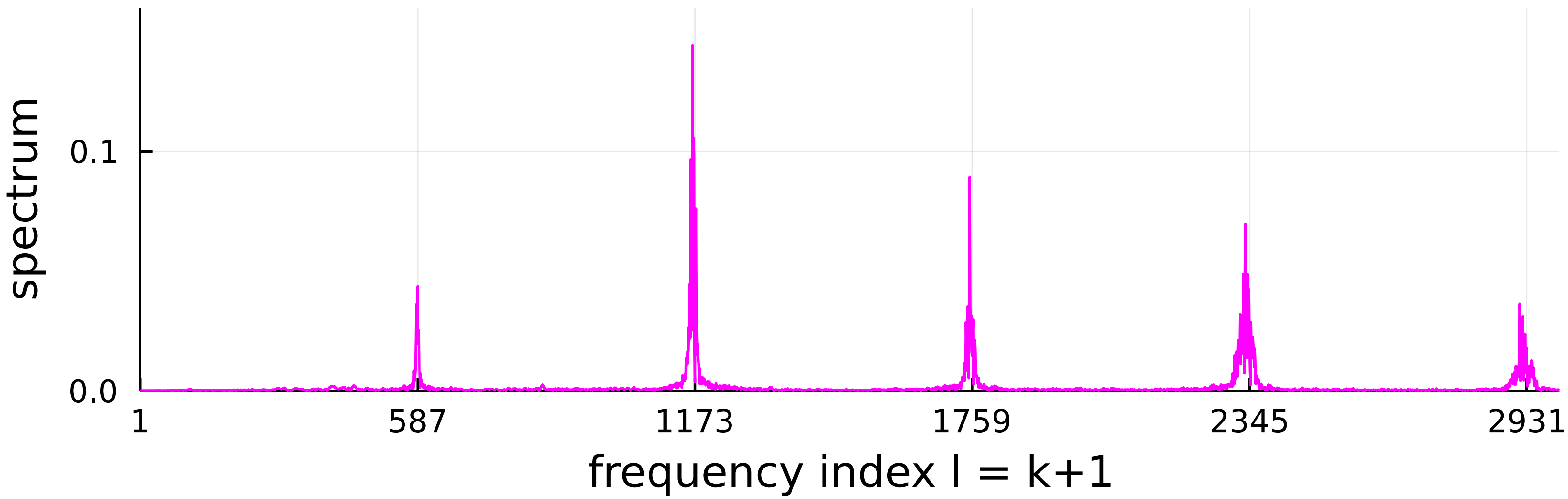
```
using Sound: record, sound
using FFTW: fft, ifft
using Plots: plot, plot!, default, gui
default(label="", markerstrokecolor=:auto, markersize=3, ytick=-1:1, widen=true)
record(0.001) # warm-up
x, S = record(5)
x = x[1:2:end] / maximum(abs, x); S ÷= 2 # reduce memory
#f0 = 540; S = 44100; x = cos.(2π*f0*(1:5S)/S).^3 # test code
Nx = length(x)
t = (1:Nx)/S
p0 = plot(t, x, xlabel="t [s]", ylabel="x[n] = x(n/S)")

Ny = 800
n = Int(2.0 * S) .+ (1:Ny)
p1 = plot!(deepcopy(p0), t[n], x[n], color=:magenta, xlims=extrema(t[n]))
y = x[n] # small segment for plot
p2 = plot(y, xlabel="n", marker=:circle, title="Signal", yaxis = ("y[n]", (-1,1)))
autocorr = real(ifft(abs2.(fft([x; zeros(length(x))])))) / sum(abs2, x)
p3 = plot(0:length(autocorr)-1, autocorr, marker=:circle, color=:orange,
 xaxis = ("shift m", (0,Ny)), yaxis = ("autocorrelation", (-1,1), [-1, 0, 0.8, 1]),
 title = "Normalized autocorrelogram")
big1 = autocorr .> 0.8 # find large values
big1[1:findfirst(==(false), big1)] .= false # ignore peak near m=0
peak2start = findfirst(==(true), big1)
peak2end = findnext(==(false), big1, peak2start) # end of 2nd peak
m = peak2start:peak2end; plot!(m, autocorr[m], color=:black, marker=:circle)
big1[peak2end:end] .= false # ignore everything to right of 2nd peak
m = argmax(big1 .* autocorr) - 1
#m = argmax(i -> autocorr[i], peak2start:peak2end) - 1 # alternative way
f = round(S/m, digits=2)
plot!([m], [autocorr[m+1]], marker=:square, color=:red, xticks=((0:6) * m),
 annotate=(Ny/2, -0.5, "f = S/m = $S/$m = $f"))
plot(p2, p3, layout=(2,1)); gui()
tone = cos.(2π*f*(1:2S)/S); sound([tone; x], S)
```

Arduino water flow meter



From [1]: "So my first challenge was to program an Arduino attached to the magnetometer to transform that noisy magnetic signal into a flow rate. I toyed with the idea of using a Fourier transform to pick out the dominant frequency corresponding to the flow rate, but instead I plumped for autocorrelation. That is, the program multiplies a short sample of the signal by a time-lagged version of itself and sums up the results. To find the dominant frequency, the Arduino code increments the lag between the two samples and looks for a peak in the summed results. That requires much less processing and seems pretty robust with respect to noise and harmonics."

Example: Trumpet spectrum



By Project 1, the fundamental frequency is

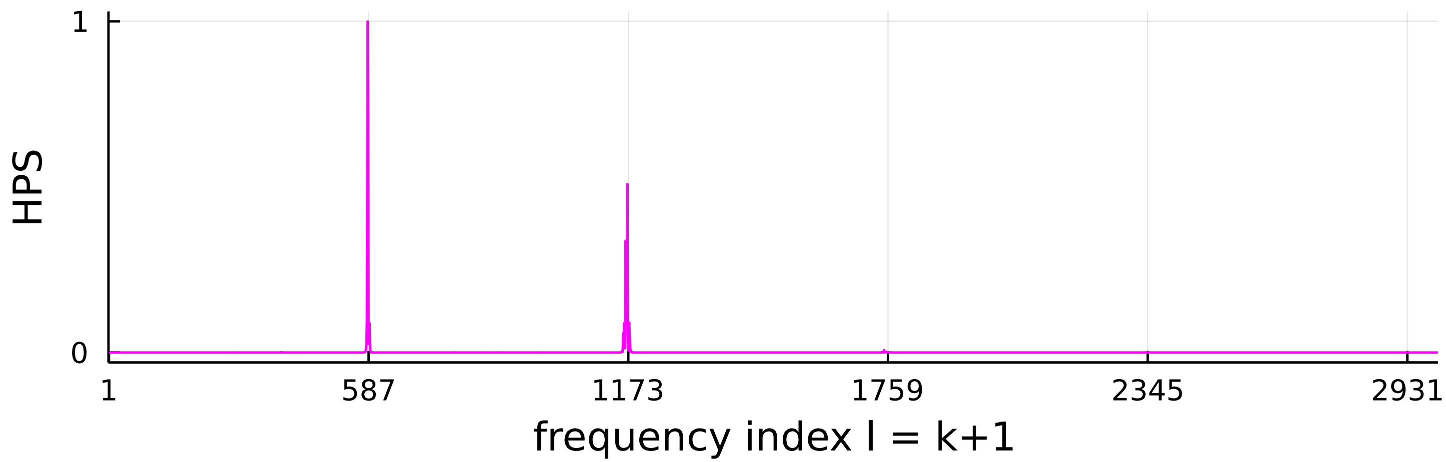
$$f = \frac{k}{N} S = \frac{587-1}{32768} 44100 \text{ Hz} = 788.7 \text{ Hz} \qquad \text{(high G)}.$$

BUT: here the 2nd peak is higher than the 1st peak.
$\Longrightarrow$ Using simple `argmax` function may locate the wrong peak!
If we do not use `argmax` how do we avoid small noise peaks?

Harmonic product spectrum (HPS)

Multiply spectrum by down-sampled versions of itself, aka decimation.

```
X = abs.(fft(x)) # spectrum
M = N ÷ 3 # here we use three copies
hps = X[1:M] .* X[1:2:2M] .* X[1:3:3M]; hps ./= maximum(hps) # normalized
```



- HPS emphasizes harmonics over other stuff, so easier to find peaks in noise.
- Now the highest peak matches the fundamental frequency.
- (HPS may not always work as well as in this example.)
- Note that we down-sampled the *spectrum*, not the *signal*!

Transcriber: Other helpful (?) ideas

- Filtering out unwanted signals (noise)
  - If we are interested in only 1 octave of pitches, then can filter out all signals not in that octave (Lab 3).
  - Can help for some approaches, but not for others.

Transcriber: frequencies

- Prof. Yagle generated the sounds in `project3.wav` using circle of fifths and multirate filtering (advanced topics).
- Frequencies differ slightly from equal temperament tuning.
- If needed, use the "tone" signal to determine tuning.
- Alternatively, round MIDI number to the nearest integer:
  $\text{MIDI} = 69 + \texttt{round}\,(12 \log_2(\text{frequency in Hertz}/440))$

Project 3 challenges

- The octave problem:
  how to distinguish G (392 Hz) from G (784 Hz).
  (Trumpet has this problem.)
- Use pattern recognition to identify instrument type from the pattern of harmonics? Not required, but will impress...
- Your team must sell/defend your choice of method(s)
  in both your final oral presentations and written reports.

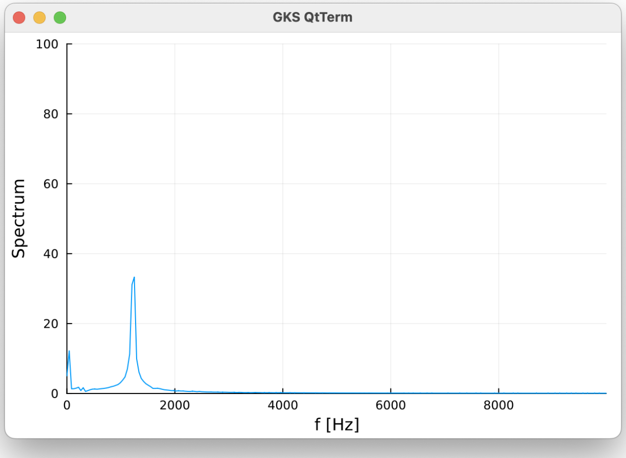
Some P3 tools / resources

- Real-time spectrum analyzer

  https://github.com/JuliaAudio/PortAudio.jl/blob/master/examples/spectrum.jl
- Some Julia audio tools: https://github.com/JuliaAudio
- Unicode music symbols: https://unicode-table.com/en/blocks/musical-symbols
- Survey paper on music transcription http://doi.org/10.1109/msp.2018.2869928
- Phase vocoder: https://en.wikipedia.org/wiki/Phase_vocoder
- Audio data sets (mostly from past students):
  - https://medleydb.weebly.com
  - https://freesfx.co.uk
  - https://philharmonia.co.uk/resources/sound-samples
  - https://imslp.org/wiki/Main_Page
  - https://mixkit.co

Real-time spectrum demo

```
# fig/p3/spectrum.jl Plot a real-time spectrum.
# cf https://github.com/JuliaAudio/PortAudio.jl/blob/master/examples/spectrum.jl

using Plots: plot, gui
using FFTW: fft
using PortAudio: PortAudioStream
using SampledSignals: domain, Hz, (..)

const N = 1024 # buffer size
const stream = PortAudioStream(1, 0)
const buf = read(stream, N)
const fmin = 0Hz
const fmax = 10000Hz
const fs = Float32[float(f) for f in domain(fft(buf)[fmin..fmax])]

while true
    read!(stream, buf)
    plot(fs, abs.(fft(buf)[fmin..fmax]), label="",
     xaxis = ("f [Hz]", (fs[1], fs[end])),
     yaxis = ("Spectrum", (0, 100))); gui()
end
```



P3 conclusion

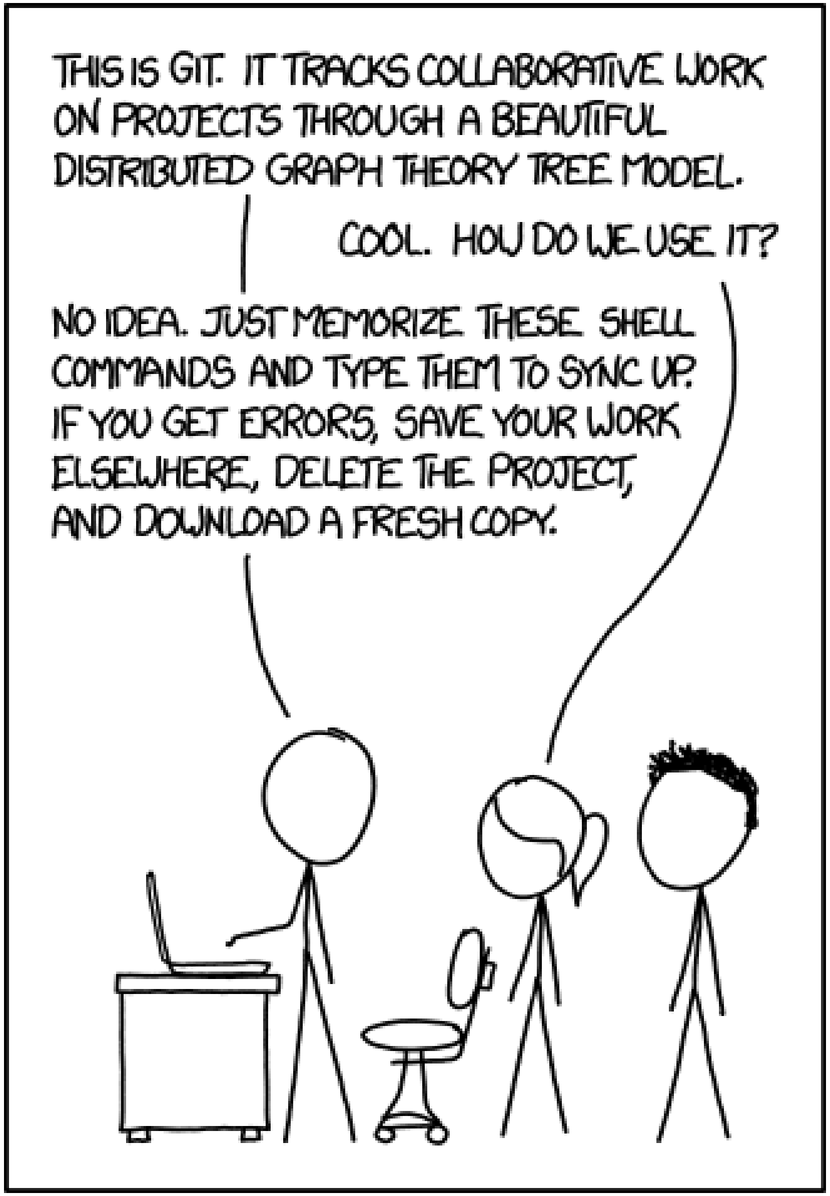
- I am not telling you how to do this project!
  Music transcription is not a solved problem;
  different approaches have pros/cons.
- Apply what you have learned in the course.
- Research on music synthesis/transcription.
  pitch detection algorithms
- There are many interesting P3 possibilities beyond the basic synthesizer and basic transcriber described here. Baseline level of sophistication...
- Your presentations of the results (tech comm)
  are as important as results themselves.
  This is very realistic for real-world engineering.

Part 4: git

Git is the way to collaborate for code!

- What is git?
  - "a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers who are collaboratively developing source code during software development." 2024-03-05
  - Like "google docs" for collaborative code editing
  - Originally authored by Linus Torvalds in 2005
- Why use git?
  - Archives all (committed) code changes (*cf.* lab notebook)
  - Track changes
  - Cloud backup via github.com, unlike VS Live Share
    Use in tandem: LS for concurrent work on a file; git to sync/backup
  - Code review
  - >90% of developers report it as their primary version control system

Powerful tools do have a learning curve



https://xkcd.com/1597

Recommendations

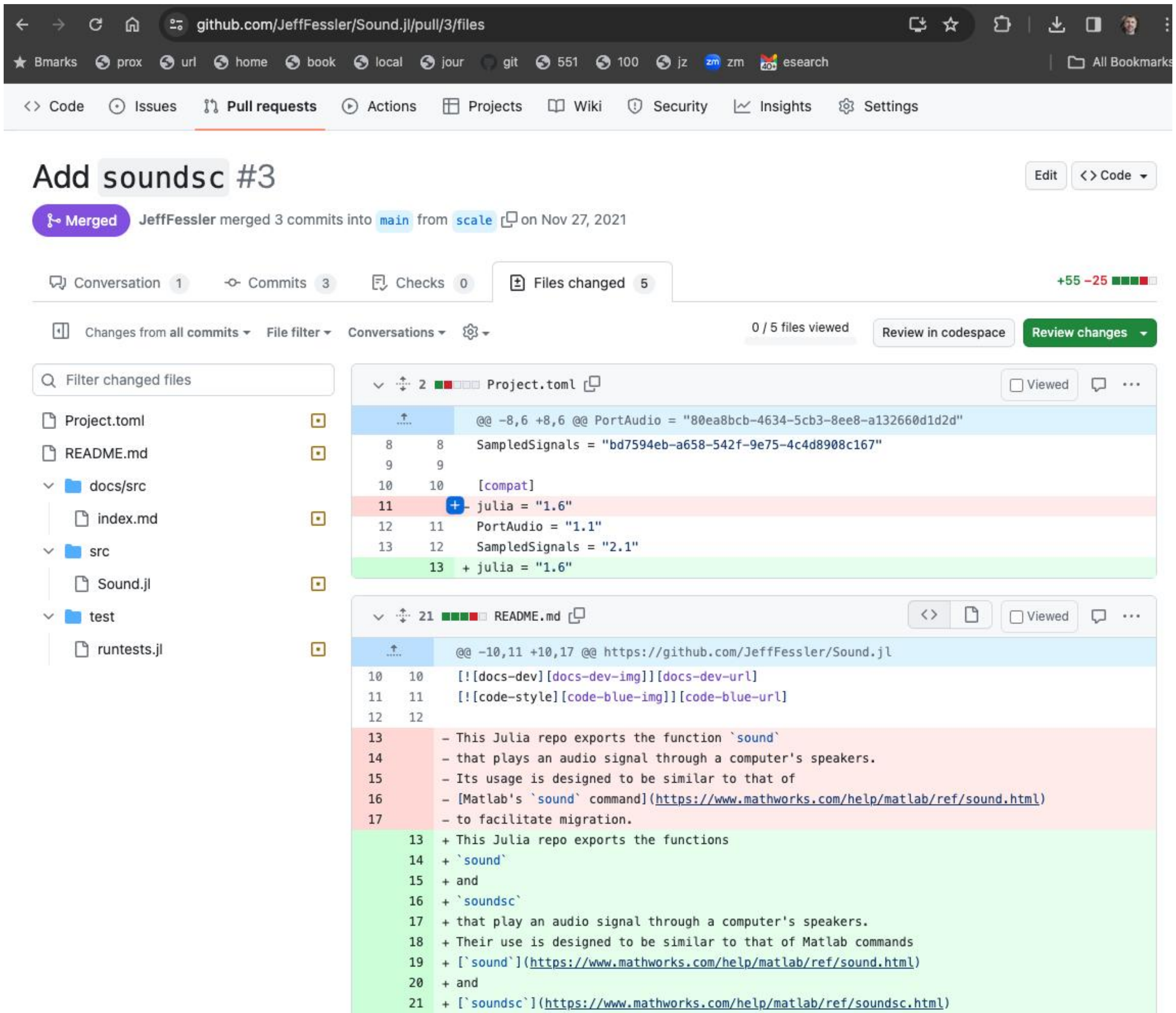
- All team members create (free) personal account at github.com
  - Follow GitHub instructions
  - Think professionally (future employers)
  - Configure 2FA security
  - Study GitHub's Hello World tutorial
- One team member:
  - Create *private* repo (code repository) for P3 at github.com
  - Add group members
  - Add `JeffFessler` and your lab instructor
  - Peer instruction for using git please!
- An effective approach for teamwork is the github-flow process.

Interacting with git and GitHub

- VS Code includes native Git support with nice tutorials
- GitHub Desktop (free) app
- Shell commands
  - `git clone`
  - `git branch`
  - `git checkout`
  - `git add`
  - `git commit`
  - `git push`
  - `git pull`

Illustration

https://github.com/JeffFessler/Sound.jl/pull/3



References

[1] D. Schneider. Water stats on tap. *IEEE Spectrum*, 52(12):22–23, December 2015.