Eng. 100: Music Signal Processing
DSP Lecture 7
Project 2: Touch-tone synthesizer / transcriber

Curiosity:

- https://www.youtube.com/watch?v=t1eRdL3dwSk (percussion transcription)
- http://supermegaultragroovy.com/products/capo/mac (guitar transcription: chords / tablature)

Announcements:

- Read Project 2 before lab this week! (Last set of reading questions.)
- HW3 / Lab 3 due this week
- HW4 on Canvas (Julia practice), due "next week" (after break)
- Midterm course evaluations soon

Teamwork!

If you need to miss discussion/lab:

- Email your teammates and discussion / lab instructors.
- Arrange to zoom with them during lab / discussion time if possible.
- If not, at least find out what you missed / what they need you to do.

Your team needs your contributions, in E100 and beyond.
(Your grade depends on it too.)

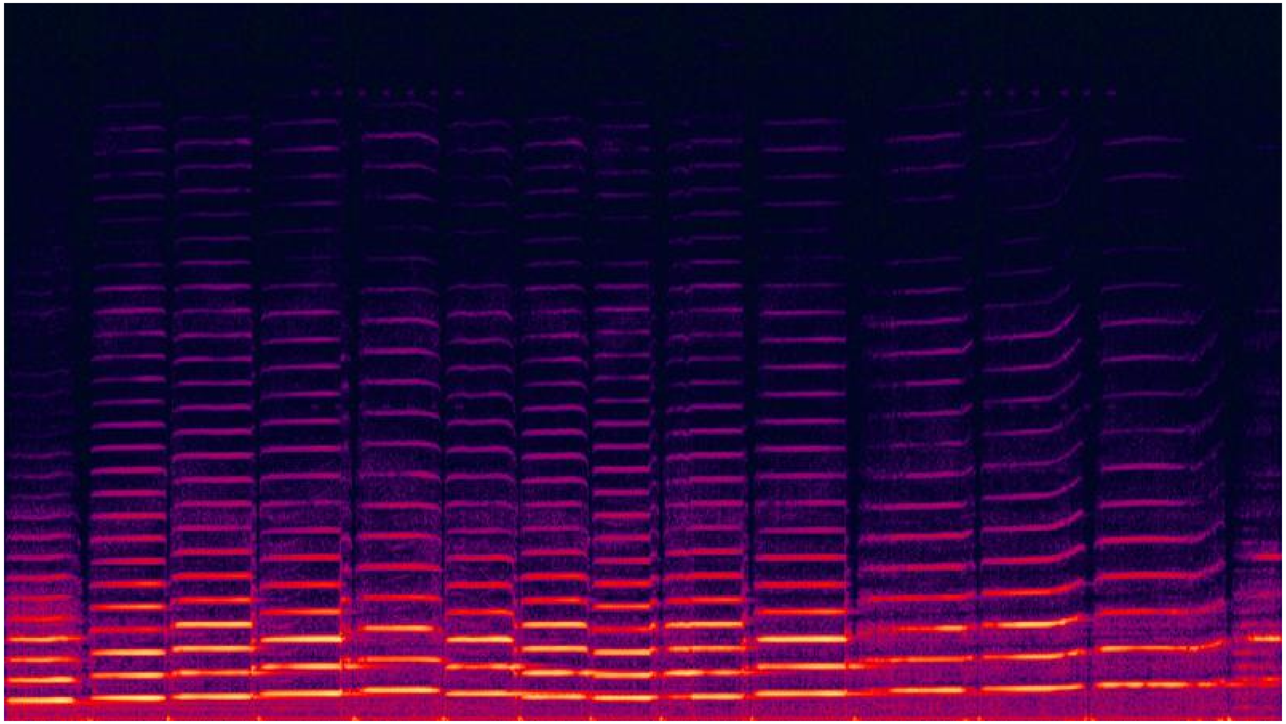
Outline

- Part 0. Spectrogram examples
- Project 2: Touch-tone phone signals
  - Part 1. Analyze spectra of touch-tone phone signals
  - Part 2. Design/build (in `Julia`) a touch-tone keypad (tone synthesizer)
  - Part 3. Design/build touch-tone transcriber (signals to phone number)
  - Part 4. Test transcriber performance for noisy signals

Process: analyze / design / build / test : a preview of Project 3

Part 0. Spectra and Spectrogram examples

Violin spectrogram

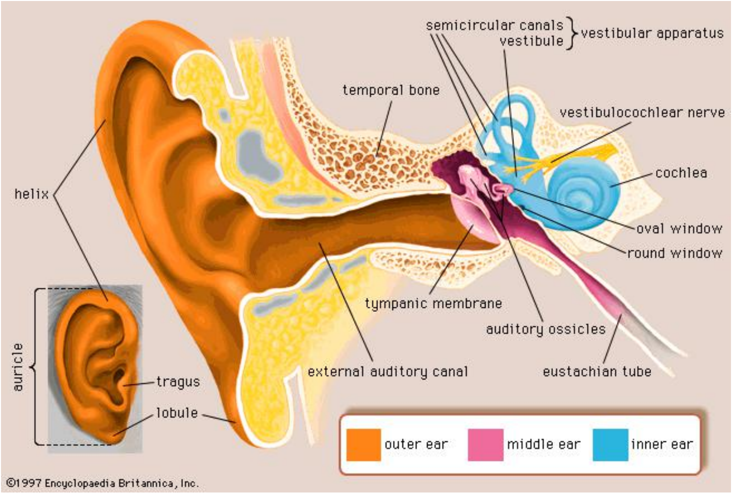


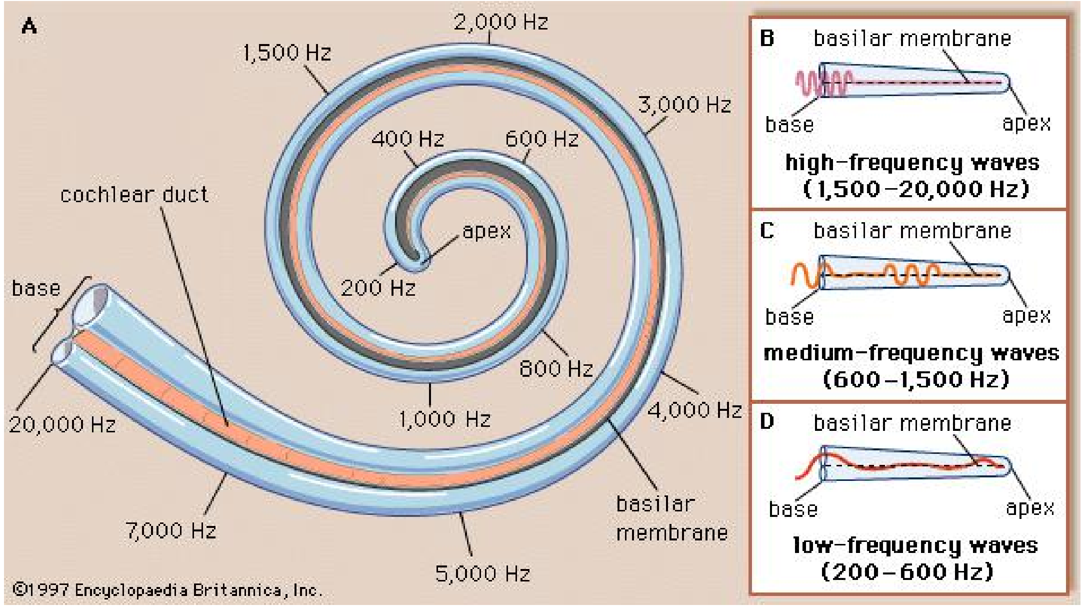
play [wiki]

Recall

- horizontal axis: time segment
- vertical axis: frequency (Hz)
- color intensity: amplitude

Nature's spectrum analyzer (Audio)





Different sound frequencies are sensed at different positions along the cochlea!

The hair cell parts called stereocilia are essentially nature's A/D converters.

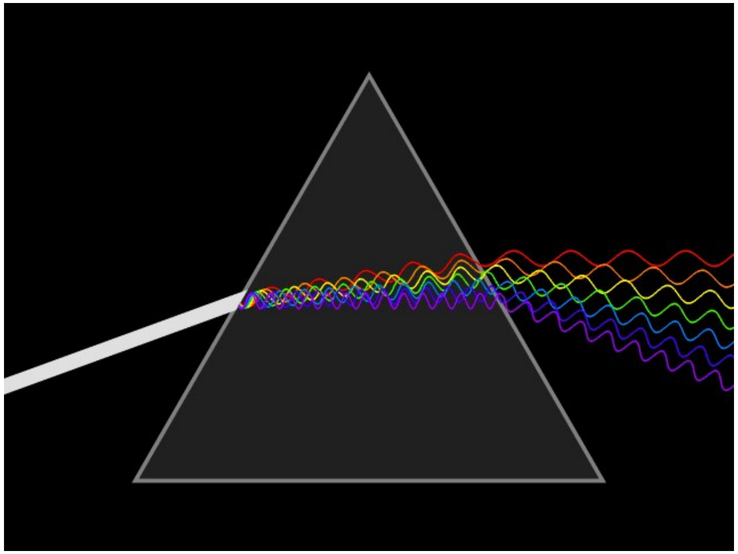
`http://www.ifd.mavt.ethz.ch/research/group_lk/projects/cochlear_mechanics`

(from Encyclopaedia Britannica Inc.)

Engineers have designed direction-finding microphones using similar principles [1]

Nature's spectrum analyzer (Light)

A glass prism separates light into individual wavelengths (electromagnetic waves with different frequencies)



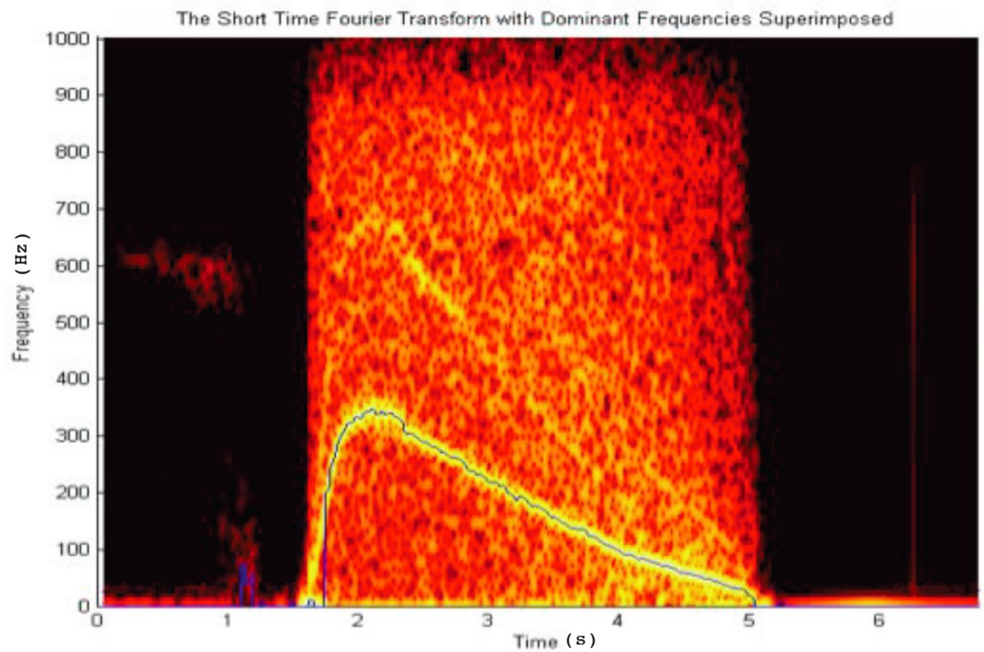
[wiki]

FFT separates sounds into individual (audio) frequencies

- White light contains "all" visible wavelengths (frequencies)
- Q: What audio signals contain "all" audio frequencies?
  A: ?? - later in this lecture

Spectrogram application: Spirometer

Low-cost spirometer designed by WUSTL students (in biomedical engineering and mechanical engineering) using microphone and spectrogram to determine air flow.



Note: "Short-Time Fourier Transform" = spectrogram Why "short time?" ??

http://mems.wustl.edu/aboutthedepartment/pages/news-story.aspx?news=335

Overview of Project 2 / Outline

Four parts:

- Part 1. Analyze spectra of touch-tone phone signals.
  - Reverse engineer tone frequencies
  - What method will you use? ??
  - Determine pattern of frequencies for touch-tone keys
- Part 2. Synthesize (in `Julia`) a touch-tone keypad.
  - Straightforward GUI using Gtk.jl: similar to Project 1 keyboard.
- Part 3. Transcribe touch-tone signals to phone number.
  - Can look for specific frequencies; do not need `fft`.
  - New method for detecting frequency components: correlator
- Part 4. Investigate your transcriber performance for signals that are degraded by white noise (*cf.*, design, build, test)

This project is a "prelude" for a (typical?) Project 3 that involves both music synthesis and music transcription.

Git is the way to collaborate for code!

Recommendation for P2 and P3:

- One team member: create a *private* repository for your team's code on `https://github.com` (or gitlab or such)
- Peer instruction for using git please!
- Use git software for collaborative code editing.
- An effective approach is the github-flow process.
- A useful tool is the (free) GitHub Desktop app.
- VS Code includes native Git support
- Github tutorial:

  `https://docs.github.com/en/get-started/quickstart/hello-world`

Think of a git repo like "google docs" for collaborative code editing.

Part 1: Analyze touch-tone signals ("reverse engineering")

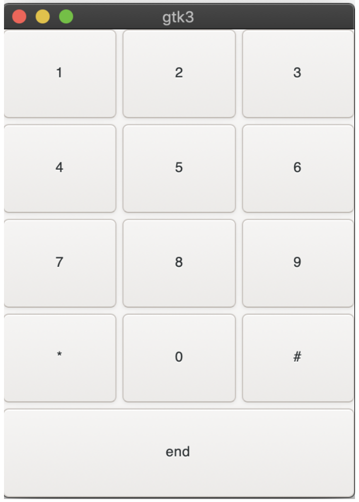
Analyze touch-tone signal spectra

- Tones from 12 keys on phone keypad in file `project2.wav`
  - Each is sampled at 8192 $\frac{\text{Sample}}{\text{Second}}$ for 0.5 second duration.
  - Use `(x, S) = wavread("project2.wav"); soundsc(x, S)`
- Analyze spectrum of *each* tone using `abs.(fft(...))`
  - Fact: each touch-tone signal is a sum of one or more sinusoids.
  - Determine # of sinusoids and frequencies (in Hz) for each tone.
  - How will you determine the # of sinusoids? ??
- Relate frequencies to touch-tone keypad
  - Look for patterns.
    (The frequency assignments were not random;
    e.g., perhaps all odd buttons use a certain frequency?)
  - Tabulate which button produces which frequencies.
- (Notice how brief this description is now.)

Part 2: Touch-tone synthesizer

Touch-tone synthesizer: keypad GUI

- Create touch-tone keypad GUI using `Gtk` (*cf.* Project 1)
- User "dials" (!) by clicking on a sequence of GUI buttons
- Simplification: 0.5 seconds for each tone (at $S = 8192$ Hz)



- Simplification: user clicks on "end" button, causing entire signal to be played using `soundsc` and stored to file `touch.wav` via:
  `wavwrite(tones, "touch.wav"; Fs=S)`
  The named keyword argument `Fs` is crucial here!
- Synthesizer check:
  `(x, S) = wavread("touch.wav"); soundsc(x)`
  should sound like a touch-tone phone "dialing" play

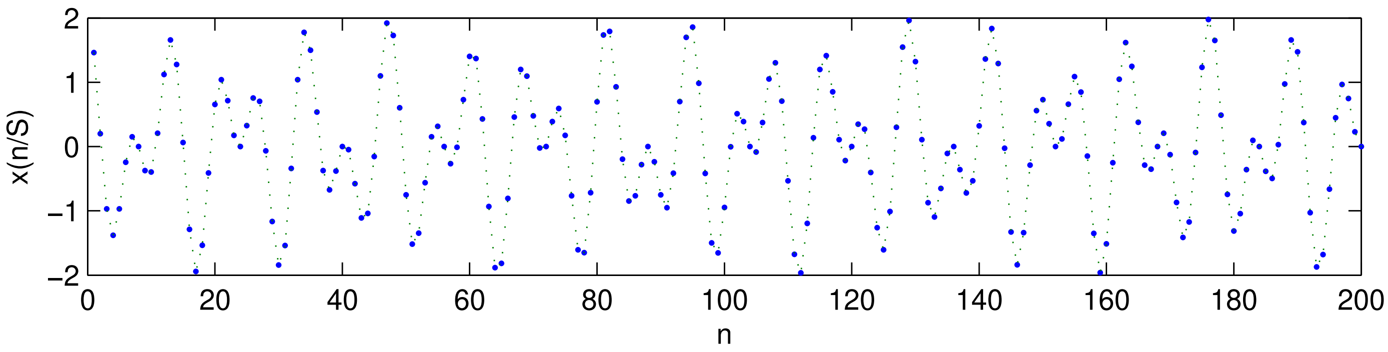
Part 3: Touch-tone transcriber

Transcribe touch-tone signals

- Input data: `(x, S) = wavread("touch.wav")`
- Output: string of numbers: 7631434
  - Do not need to include `*` or # (not in phone numbers)
  - No hyphens (dashes) needed
-

Q0.1 How to determine # of buttons pressed?
A: `length(x)` B: `length(x)` / 4096 C: `length(x)` / 8192 D: `length(x)` / 16392 E: None of these ??

- How to determine which button was pressed for each tone?



Q0.2 Could arccos method work here?

A: True B: False ??

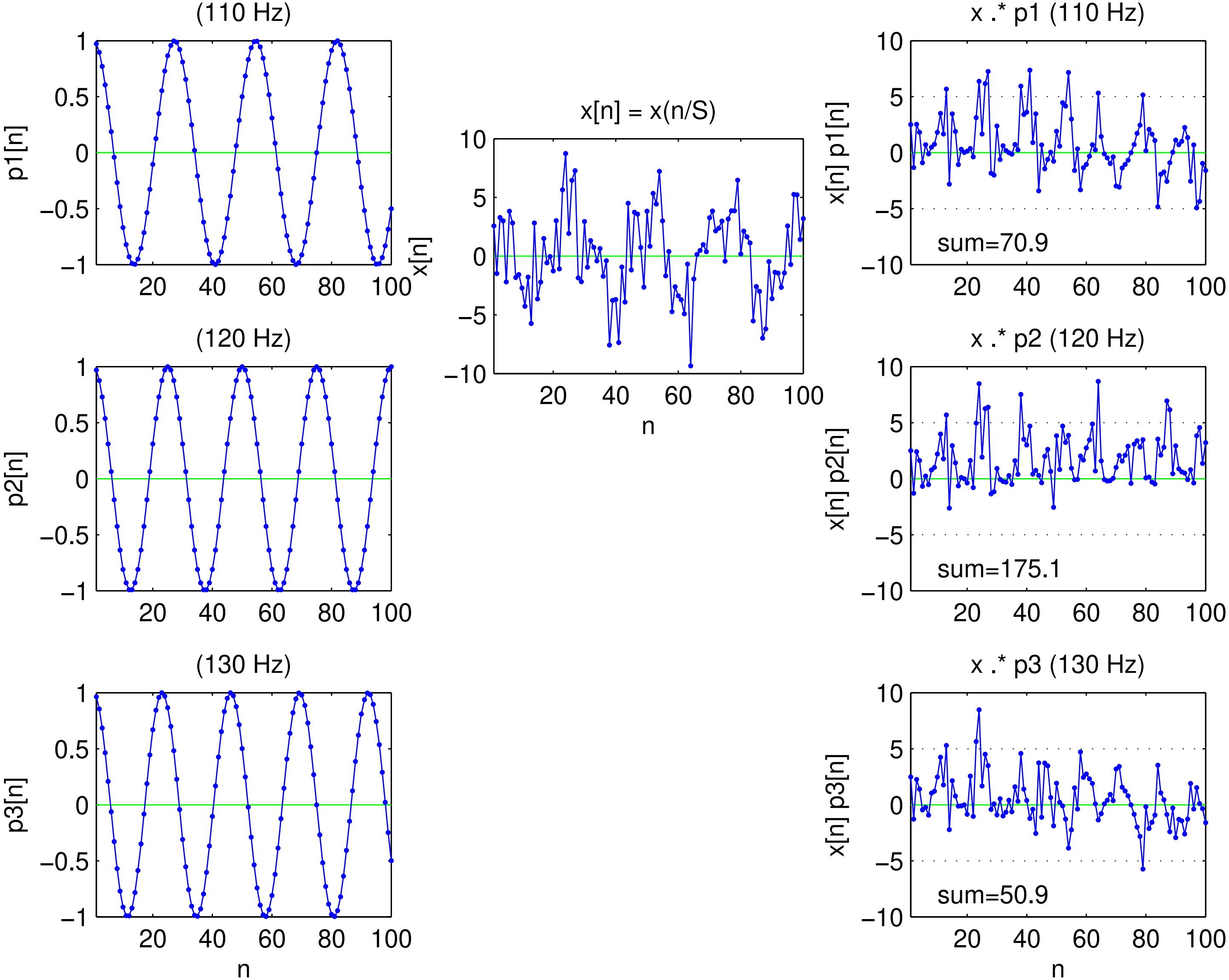
Q0.3 Could `abs.(fft(...))` method work here?

A: True B: False ??

Pattern recognition using correlation (new signal processing method)

- For this application the possible signal patterns are few:
  - There are only 12 buttons, each with a unique signal.
  - Each signal is sum of a very small number of sinusoids.
  - We know all the possible frequencies.
- Do not use `abs.(fft())`. It is unnecessarily expensive!
- New approach: a type of pattern recognition
  - Find specific frequencies using correlation operation, also known: matched filter, dot product, inner product.
  - We try to find the sinusoids that "best match" the signal. [wiki]
  - Correlation is quite robust to noise / interference.
  - Correlation is even faster / cheaper than FFT when # of possible frequencies is much smaller than # of signal samples.
  - Correlation is the foundation of deep learning (aka AI / machine learning) using convolutional neural network (CNN) methods

Correlation example: $\sum_{n=1}^{N} x[n]p[n]$



Correlation implementation: Basic

- Correlation method: multiply and sum.
  - Multiply input signal `x` by (each) candidate pattern signal.
  - Sum the resulting product signal: gives correlation value.
  - Choose candidate pattern with largest correlation.
- Example implementation in `Julia`:
  - `freqs = [110, 120, 130]`
    (The candidate frequencies in this example.)
  - Matrix-vector multiplication version:
    `corr = cos.(2pi*freqs*(1:N)'/S) * x` sizes!
    Alternative version using `Julia` comprehension loop:
    ```
    using LinearAlgebra: dot
    corr = [dot(cos.(2pi*f*(1:N)/S), x) for f in freqs]
    ```
    Either returns vector of 3 correlation values for the candidate frequencies.
  - `index = argmax(corr)`
    Returns `index` of largest correlation value in `corr` array.
  - `freqs[index]`
    frequency of the sinusoid that best matches signal `x`

Correlation implementations in `Julia`

Given two (column) vectors of same length:

$$\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \qquad \mathrm{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Mathematical formula:

$$\text{Correlation}(\mathrm{x},\mathrm{y}) = \sum_{n=1}^{N} x_n y_n \ .$$

Fortran/BASIC/C/C++ style implementation in `Julia`:

```
function correlate(x, y)
    corr = 0
    for n in 1:length(x)
        corr += x[n] * y[n]
    end
    return corr
end
```

Faster `Julia` "vectorized" style implementation (*cf.* math):

```
corr = sum(x .* y)
```

Using `Julia`'s `dot` function (most descriptive way):

```
using LinearAlgebra: dot
corr = dot(x, y)
```

Even less typing `Julia` implementation (for column vectors):

```
corr = y' * x
```

Here the `y'` is the transpose of the vector `y`.
This last form is convenient for correlating many signals with `x`.

Again, these all compute a dot product of two N-dimensional vectors.

Correlation implementation: Improved

The preceding implementation works fine for a $\cos$ signal, but not if it is $\sin$. We want it to work for both. In fact we want it to work for a sinusoid of *any* phase.

Improved implementation to be used in Project 2 (your frequencies will be different; from Part 1):

```
N = length(x); S = 8192
freqs = [110, 120, 130]
c = cos.(2π * freqs * (1:N)'/S) * x
s = sin.(2π * freqs * (1:N)'/S) * x
corr = s.^2 + c.^2
index = argmax(corr) # "argument that maximizes"
fbest = freqs[index]
```

- Basic idea: find which candidate frequencies best match `x` either as a $\cos$ or as a $\sin$ wave.
- After finding frequencies, decode which button was pressed for each 0.5 second segment of synthesizer output.

Illustration

```
S = 8192; N = 80
x = cos.(2π * 119 * (1:N)/S .- π/4) # 119 Hz with phase shift
freqs = [110, 120, 130] # candidate frequency list
c = cos.(2π * freqs * (1:N)'/S) * x # correlated with cos
s = sin.(2π * freqs * (1:N)'/S) * x # correlated with sin
corr = s.^2 + c.^2 # combine sin and cos correlations
@show argmax(c) # returns "1" :(
@show argmax(s) # returns "3" :(
@show index = argmax(corr) # returns "2" :)
fbest = freqs[index]
```

Combining correlations with both cos and sin works the most robustly for finding specific sinusoidal components.

Q0.4 What is the final value computed by this code?

??

Summary of Transcriber Specifications

- Length of each phone digit known: 0.5 sec.
- Sampling rate known: $S = 8192 \, \frac{\text{Sample}}{\text{Second}}$.
- Touch-tone signal written to file `touch.wav`.
- Use correlation method to find best match for each segment
- Do not use `abs.(fft())`: too much computation!
- Do not use numerous `if` statements: inefficient
- Output: String of phone digits without hyphen.

Part 4: Investigating transcriber accuracy in the presence of noise

Transcriber accuracy in noise: Overview

- Phone signals, wired or wireless, have noise present.
  (Music signals too.)
- So far we have mostly ignored noise.
- Noise: What exactly is noise?
- Performance: How well does your transcriber work when noise is present (as in real world)?
- Figure of merit: Numerical measurement of performance of a system (detector, estimator).

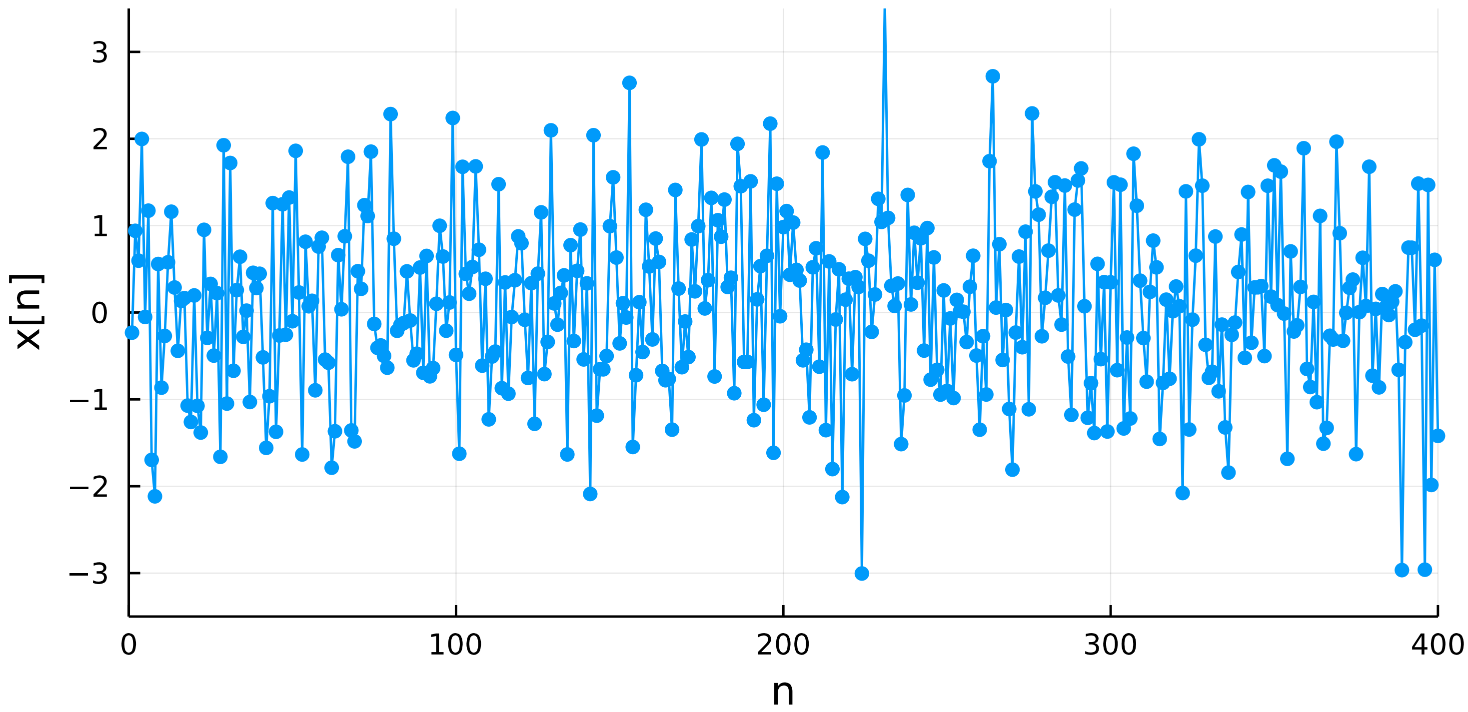
This is a first taste of the "design, build, test" engineering process.

Zero-Mean Additive White Gaussian Noise

$$\underbrace{y(t)}_{\text{measured signal}} = \underbrace{x(t)}_{\text{ideal signal}} + \underbrace{\varepsilon(t)}_{\text{additive noise}}$$

- Additive White Gaussian Noise (AWGN) is a good model for many actual sources of noise.
- Why called white? ??
- At each time t: $\varepsilon(t)$ has a Gaussian distribution (bell curve).
- At any two times t_0 and t_1, no matter how close:
  - $\varepsilon(t_0)$ and $\varepsilon(t_1)$ are completely uncorrelated:
  - knowing $\varepsilon(t_0)$ will not help one predict $\varepsilon(t_1)$
  - In words, $\varepsilon(t)$ is "completely random"
- Zero-mean implies the DC value (or average value) is zero.

White Gaussian Noise Example



To hear it:

```
z = randn(8000); soundsc(z, 8192);
```

play

Why does it sound like the wind or the ocean surf? ??

Touch-tone transcriber performance measure

- Noise level rises $\Longrightarrow$ transcriber gets more digits wrong.
- We want to investigate and quantify this error rate.
- Key factor: signal level relative to noise level.
- Signal-to-noise ratio (SNR) in decibels (dB):
  Mathematically:

$$\text{SNR} = 10\log_{10}\left(\frac{\sum_{n=1}^{N} x_n^2}{\sum_{n=1}^{N} \varepsilon_n^2}\right) = 10\log_{10}\left(\frac{x_1^2 + \cdots + x_N^2}{\varepsilon_1^2 + \cdots + \varepsilon_N^2}\right)$$

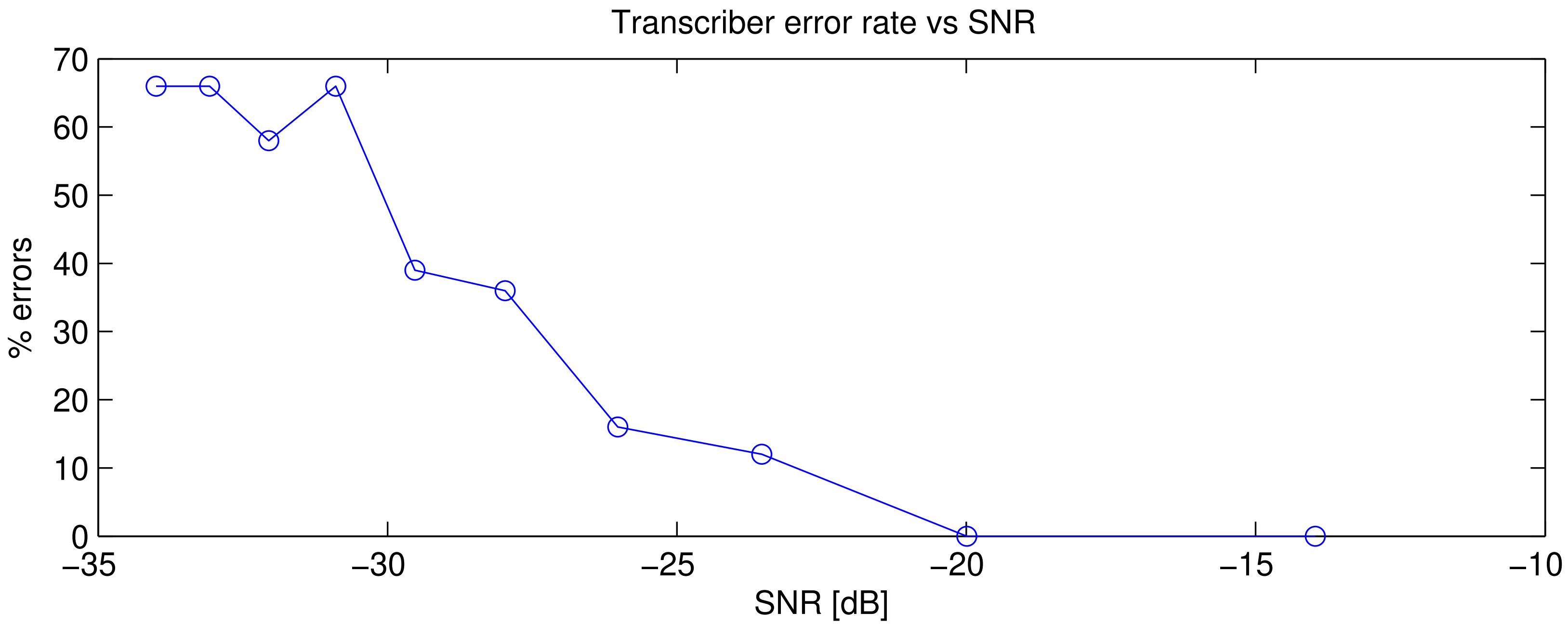
In `Julia`:

```
SNR = 10*log10(sum(x.^2) / sum(noise.^2))
```

where:
  - `x` is vector of ideal signal values
  - `noise` is vector of noise values.

Transcriber error rate plot

- SNR is on horizontal axis;
- error rate (percentage) is on vertical axis.



Touch-tone performance investigation

- Transcriber gets some digits wrong.
- Error rate: Fraction of wrongly decoded digits.
- Need to "survey" many digits to get accurate measure.
- For each SNR: use 100 digits, count # decoded incorrectly.
- Each call to `randn` generates new noise values.
- Random digits versus same digit each time?
- Plot: error rate vs SNR for several SNR levels.

Transcriber Performance: Comments

- Any potential transcriber customer will want to see your plot of error rate vs SNR.
- Below some threshold SNR, error rate will rise rapidly
- What transcriber error rate is acceptable?
- What noise level can your transcriber tolerate?
- How to achieve even better performance?
  Error-correction. Digital communications: EECS 455

Outline of Julia Program for Error Rate Study

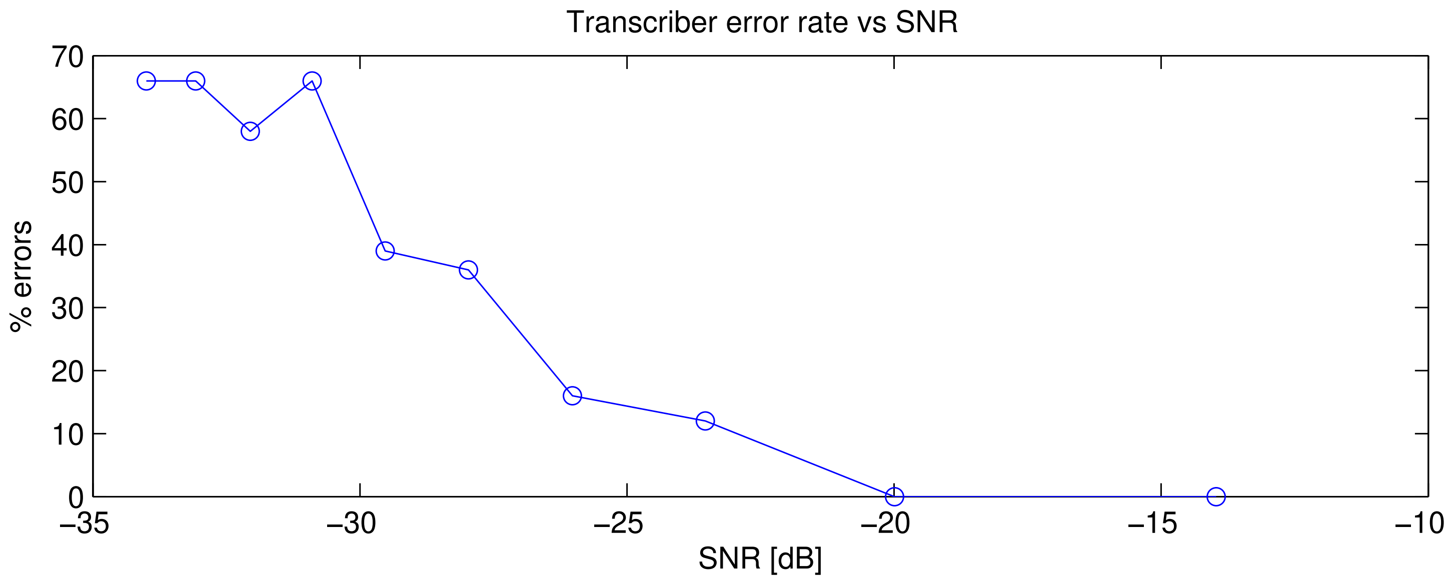
Investigating transcriber performance in noise (pseudo-code)

```
x = "?? signal_for_button_1 ??"
errors = zeros(Int, 10); snr = zeros(10)
for level in 1:10 # 10 different noise levels
  noisesum = 0
  for trial in 1:100 # 100 trials for each noise level
    noise = 5 * level * randn(size(x))
    y = x + noise # this will be very noisy!
    noisesum += sum(abs2, noise) # sum of noise.^2
    # apply your transcriber to signal "y" here
    if "?? transcriber_does_not_output_1 ??"
      errors[level] += 1 # count errors
    end
  end
  snr[level] = 10*log10(sum(abs2, x) / (noisesum/100))
end
plot(snr, errors, marker=:circle, ...)
```

Explanation of Error Rate Code Template

- Outer loop over 10 different SNR levels:
  different noise strengths because of `5*level*randn()`
- Inner loop over 100 trials;
  each trial with different random noise realization
- Use the signal for button "1" each time; makes things easier.
- Count # of errors in 100 trials;
  By using 100 trials, this will be error rate as a percentage.
- `nsum/100` is *average* noise power over 100 trials
- `sum(abs2, x)` is signal power, *i.e.*, $\sum_n |x_n|^2$
- `snr` and `errors` are both vectors of 10 values.

Typical Error Rate vs. SNR Plot



Note the threshold at an SNR of about -20 dB.

- Below this, error rate increases dramatically
- For any SNR below 0, the noise level exceeds the signal level, yet your transcriber still works down to about -20 dB! This is because correlation is very robust to noise.

Summary of Project 2

1. Reverse engineer touch-tone frequencies using `fft`
2. Design/build simple GUI for touch-tone synthesizer
3. Design/build touch-tone transcriber based on correlation
   Your transcriber will work perfectly in absence of noise
4. Test: investigate how your transcriber error rate increases
   as noise level increases (SNR decreases)

Correlation is one of several "pattern recognition" techniques used in numerous signal processing applications, including SONAR, RADAR, ultrasound imaging, fingerprint recognition, retina scans, ...

Read Project 2 before lab this week!

(Return to aliasing if time permits.)

References

[1] H. Esfahlani, S. Karkar, Hervé Lissek, and J. R. Mosig. Exploiting the leaky-wave properties of transmission-line metamaterials for single-microphone direction finding. *J. of the Acoustical Soc. of America*, 139(6):3259–66, 2016.