Eng. 100: Music Signal Processing

DSP Lecture 10

Music synthesis: Advanced methods

Curiosity:

http://www.engadget.com/2015/07/11/a-collection-of-man-made-music-and-the-machines-behind-it

http://www.sonicvisualiser.org

https://sites.research.google/tonetransfer

Announcements:

- CDR deadline extended to Monday

Outline

- Part 1. Advanced music synthesis methods
  - Amplitude variations
    - Envelope
    - Attack, Decay, Sustain, Release (ADSR)
    - Tremolo
  - Frequency / spectrum variations
    - Vibrato
    - Glissando
- Part 2. Filters (Timbre effects)
- Part 3. DSP application: Beats per minute (BPM)
- Part 4. Quantization
- Part 5. Project 3 tips
  - `reshape`
  - (Classic) P3 transcriber hints: note durations

Learning objectives

- Awareness of musical synthesis and processing effects

Part 1. Advanced music synthesis methods

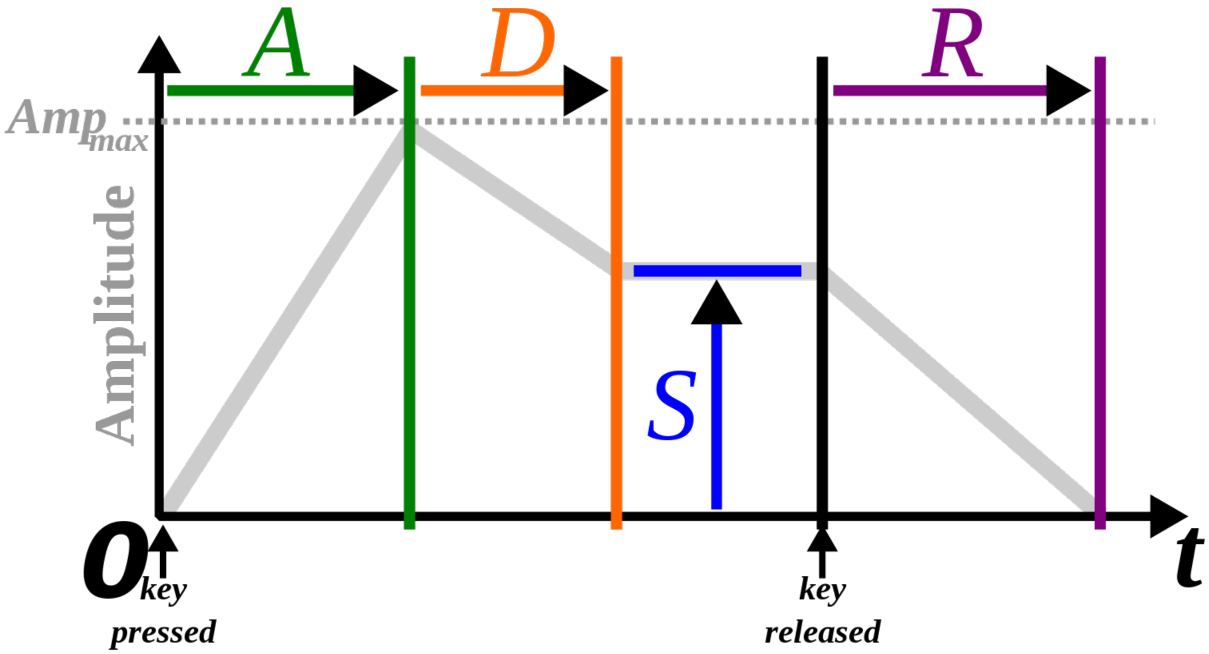
Amplitude effects

- ASDR Envelope
- Tremolo

Attack, Decay, Sustain, Release (ADSR)

Many synthesizers let user control envelope using 5 ADSR variables:

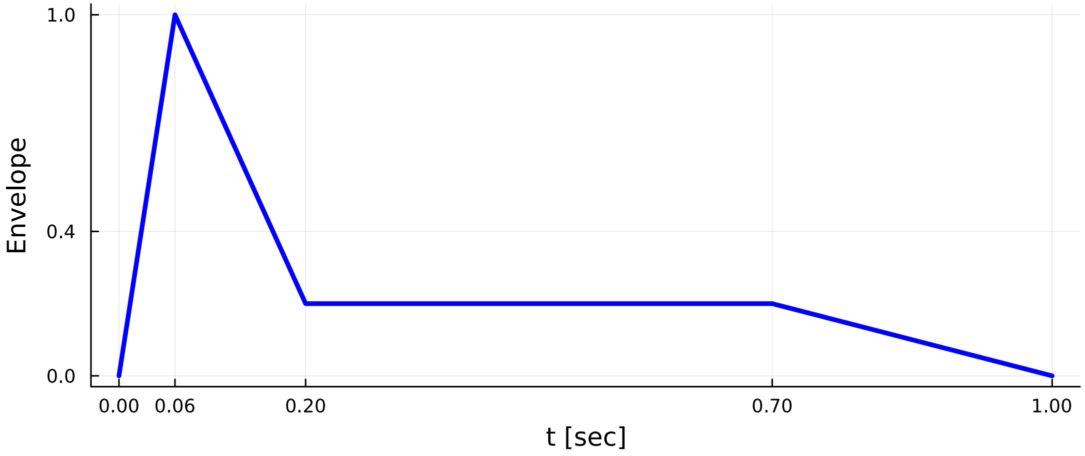
- Attack time: initial rise
- Decay time: initial fall
- Sustain level: while key is held (or sustain pedal is pressed)
- Release time: after key (or pedal) is released
- Amplitude: maximum value



https://commons.wikimedia.org/wiki/File:ADSR_parameter.svg

Example ADSR implementation in `Julia`

```
using MIRT: interp1 # one of many Julia interpolators
S = 44100; duration = 1.0
c = 1 ./ (1:2:15) # amplitudes
f = (1:2:15) * 494 # frequencies
adsr_time = [0, 0.06, 0.2, 0.7, 1] * duration
adsr_vals = [0, 1, 0.2, 0.2, 0]
t = 0:1/S:duration
env = interp1(adsr_time, adsr_vals, t) # !!
x = sin.(2π * t * f') * c
y = env .* x
```



Q0.1 What kind of synthesis is used here?

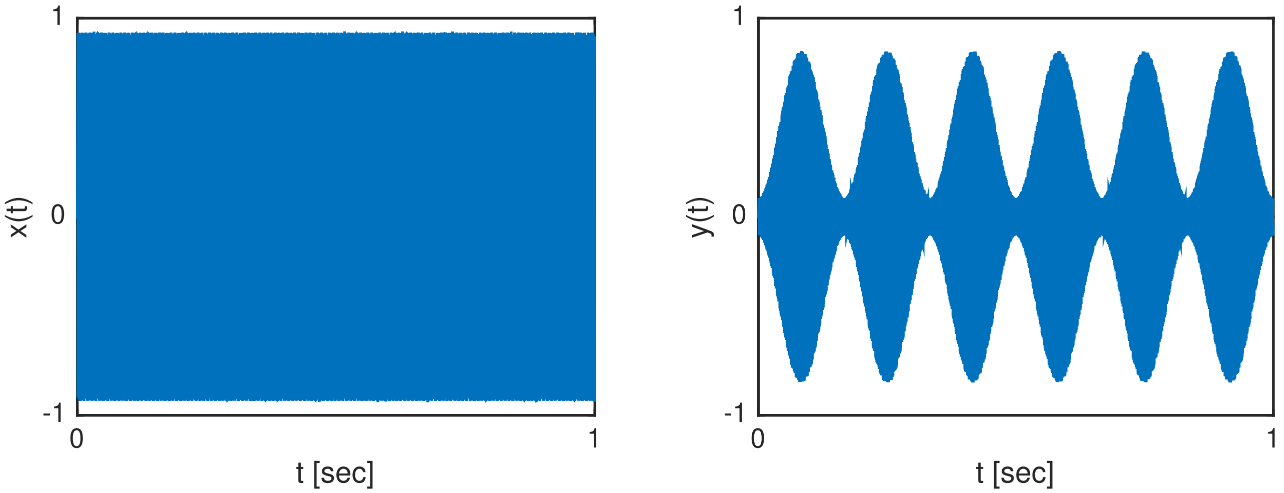
A: Additive B: Subtractive C: Pure sinusoid D: FM E: None ??

Tremolo

Tremolo implementation: LFO

This code illustrates tremolo via low-frequency oscillation (LFO)

```
S = 44100
N = Int(1 * S)
t = (0:N-1)/S
c = 1 ./ (1:2:15) # amplitudes
f = (1:2:15) * 494 # frequencies
lfo = 0.5 .- 0.4 * cos.(2π*6*t) # what frequency?
x = sin.(2π * t * f') * c # concise way
y = lfo .* x
```

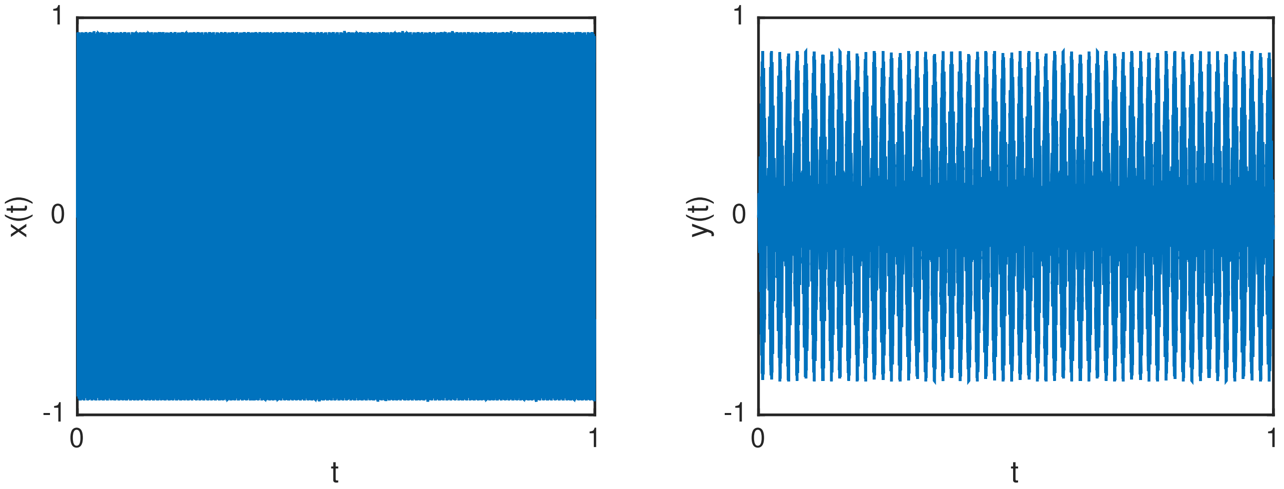


play

play

Tremolo: Why LFO?

```
S = 44100
N = Int(1 * S)
t = (0:N-1)/S
c = 1 ./ (1:2:15) # amplitudes
f = (1:2:15) * 494 # frequencies
x = sin.(2π * t * f') * c # concise way
lfo = 0.5 .- 0.4 * cos.(2π*60*t) # what frequency now?
y = lfo .* x
```



play

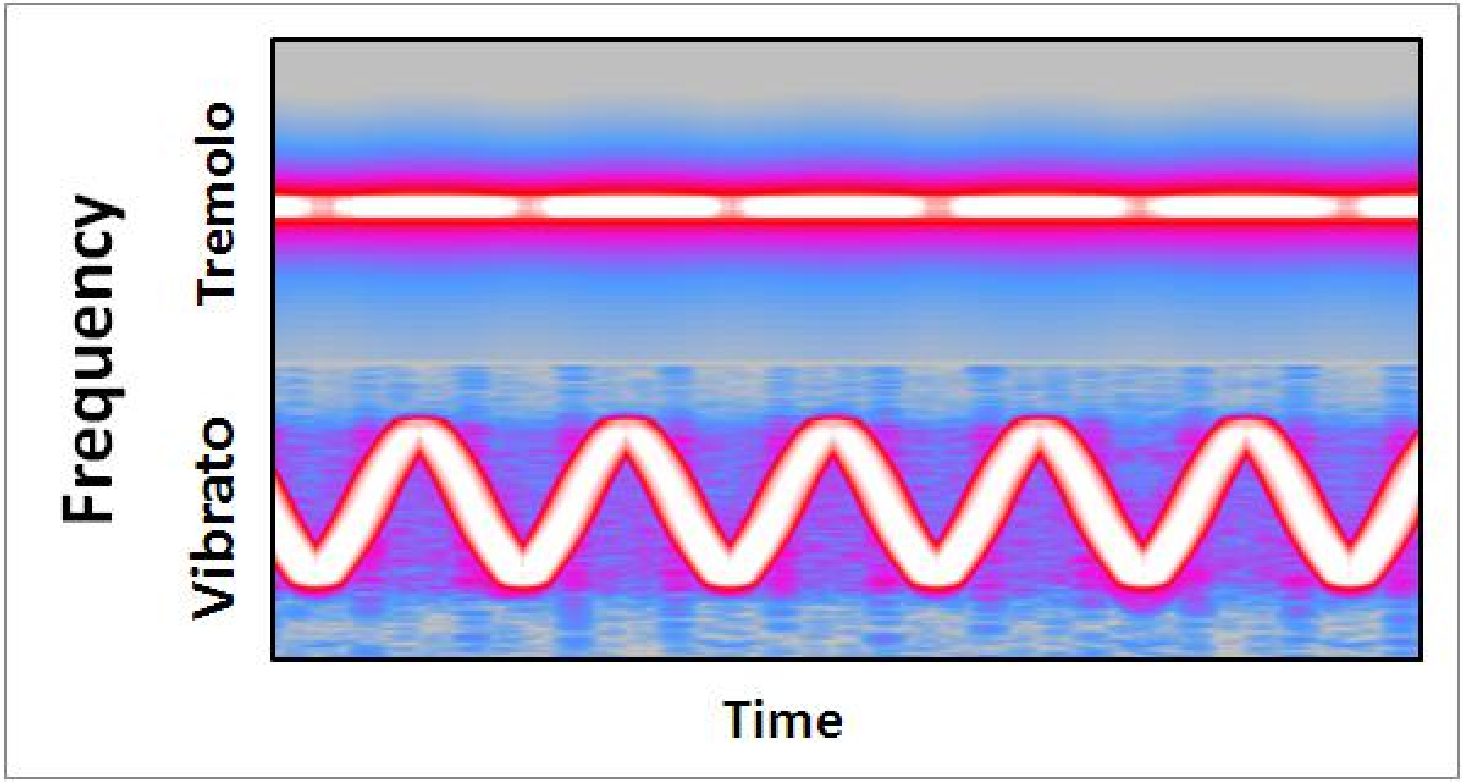
play

Frequency variations: vibrato and glissando

Vibrato

Vibrato vs Tremolo: Spectrograms

Spectrograms of vibrato and tremolo:

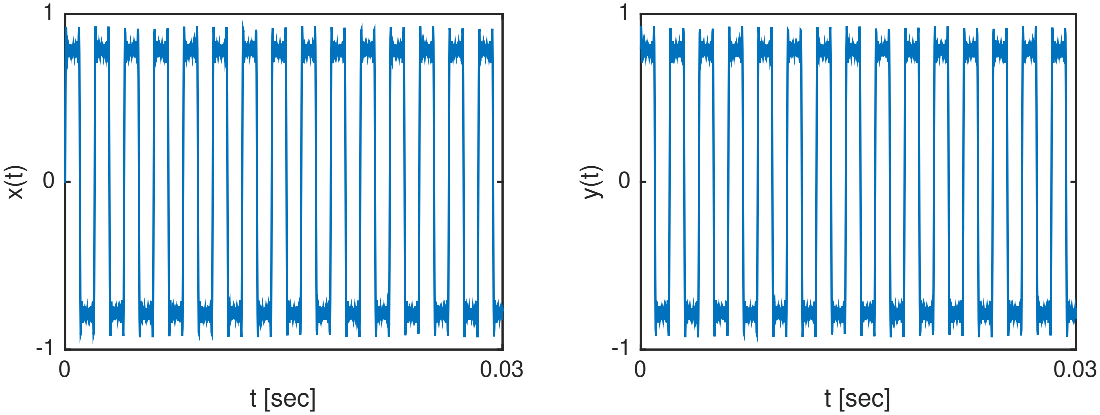


[wiki]

A Leslie speaker in a Hammond organ has both!

Vibrato implementation: LFO

```
S = 44100
N = Int(2 * S)
t = (0:N-1)/S
c = 1 ./ (1:2:15) # amplitudes
f = (1:2:15) * 494 # frequencies
x = zeros(N); y = zeros(N);
lfo = 0.001 * cos.(2π*4*t) / 4 # about 0.1% pitch variation
x = +([c[k] * sin.(2π * f[k] * t)              for k in 1:length(c)]...)
y = +([c[k] * sin.(2π * f[k] * t + f[k] * lfo) for k in 1:length(c)]...)
```



play

play

Vibrato: Why LFO?

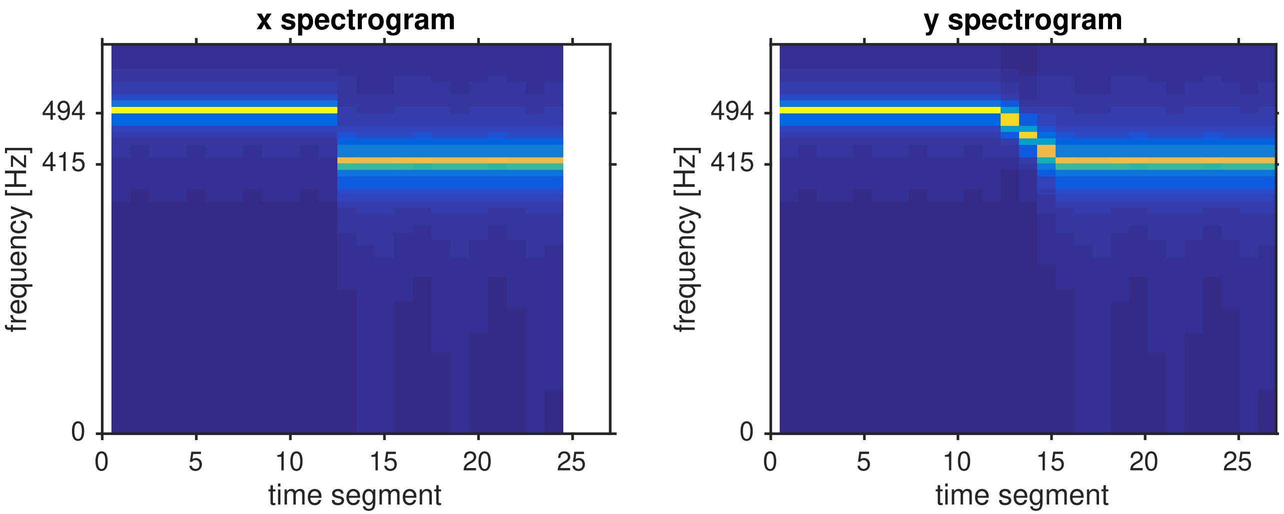
(try more; you may not like it)

Note: FM synthesis is like an extreme form of vibrato

Glissando

Glissando implementation

```
S = 44100
N = Int(1 * S)
t = (0:N-1)/S
f = [1, 2^(-3/12)] * 494 # frequencies: B G#
x = [cos.(2π * f[1] * t); cos.(2π * f[2] * t)]
tau = 0.25 # length of glissando
t2 = (0:Int(tau*S)-1)/S
gliss = cos.(2π * (f[1] * t2 + t2.^2/tau/2*(f[2] - f[1])))
y = 0.9 * [cos.(2π * f[1] * t); gliss; cos.(2π * f[2] * t)]
```



play

play

Frequency variations: Theory

(This page requires calculus and is entirely optional.)

For a (standard) sinusoid: $x(t) = \cos(2\pi f t) \Longrightarrow$

$$\frac{\mathrm{d}}{\mathrm{d}t} x(t) = -\sin(2\pi f t)\, 2\pi \underbrace{f}_{\text{frequency}}$$

For a sinusoid with time-varying phase: $x(t) = \cos(\phi(t)) \Longrightarrow$

$$\frac{\mathrm{d}}{\mathrm{d}t} x(t) = -\sin(\phi(t))\, 2\pi \underbrace{\frac{1}{2\pi} \frac{\mathrm{d}}{\mathrm{d}t} \phi(t)}_{\substack{\text{instantaneous} \\ \text{frequency } f_t}}$$

Example.

$$\phi(t) = 2\pi f t \Longrightarrow f_t = \frac{1}{2\pi} \frac{\mathrm{d}}{\mathrm{d}t} \phi(t) = \frac{1}{2\pi} 2\pi f = f.$$

Example: Glissando

Example. To make a signal with *increasing* instantaneous frequency

$$f(t) = f_1 + \frac{t}{\tau}(f_2 - f_1),$$

the Fundamental theorem of calculus says we need the following time-varying phase:

$$\begin{aligned}\phi(t) &= 2\pi \int_0^t f(t')\,\mathrm{d}t' = 2\pi \int_0^t \left[f_1 + \frac{t'}{\tau}(f_2 - f_1)\right]\mathrm{d}t' \\ &= 2\pi f_1 t + 2\pi \frac{t^2}{2\tau}(f_2 - f_1),\end{aligned}$$

so that the instantaneous frequency is

$$f_t = \frac{1}{2\pi}\frac{\mathrm{d}}{\mathrm{d}t}\phi(t) = f_1 + \frac{t}{\tau}(f_2 - f_1).$$

So the desired signal for such a glissando is

$$x(t) = \cos(\phi(t)) = \cos\left(2\pi f_1 t + 2\pi \frac{t^2}{2\tau}(f_2 - f_1)\right).$$

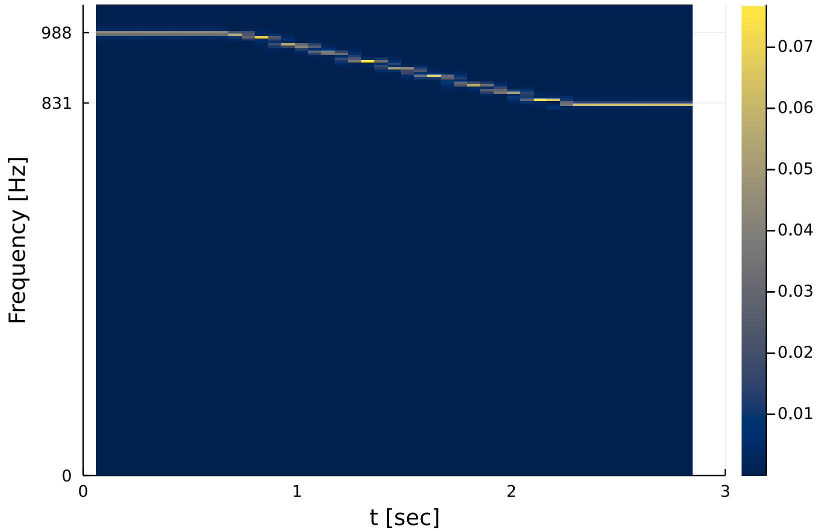
See `Julia` example on earlier slide.

Vibrato combined with glissando

(Suggested by a F11 student; dual major in EE and opera.)

```julia
# fig_gliss2.jl glissando + vibrato

S = 44100
N = 2^15
t1 = (0:N-1)/S
f = 2 * [1, 2^(-3/12)] * 494 # frequencies: B G#
x = [cos.(2π * f[1] * t1); cos.(2π * f[2] * t1)]
Ngliss = 2^16; tau = Ngliss / S # length of glissando
t2 = (0:Ngliss-1)/S
nvibe = 9
phi = 2π * (f[1] * t2 + t2.^2 / tau / 2 * (f[2] - f[1]) +
    5 * tau/nvibe/2π*cos.(2π * nvibe/tau * t2))
x = [cos.(2π * f[1] * t1); cos.(phi); cos.(2π * f[2] * t1)]
```

play



Q0.2 What would spectrum of $x(t)$ look like?

??

Electronic glissando example


https://en.wikipedia.org/wiki/File:Deliaderbyshire.jpg

Delia Derbyshire (1937-2001)

Her 1963 Doctor Who work was one of the first television themes to be created and produced entirely with electronics.

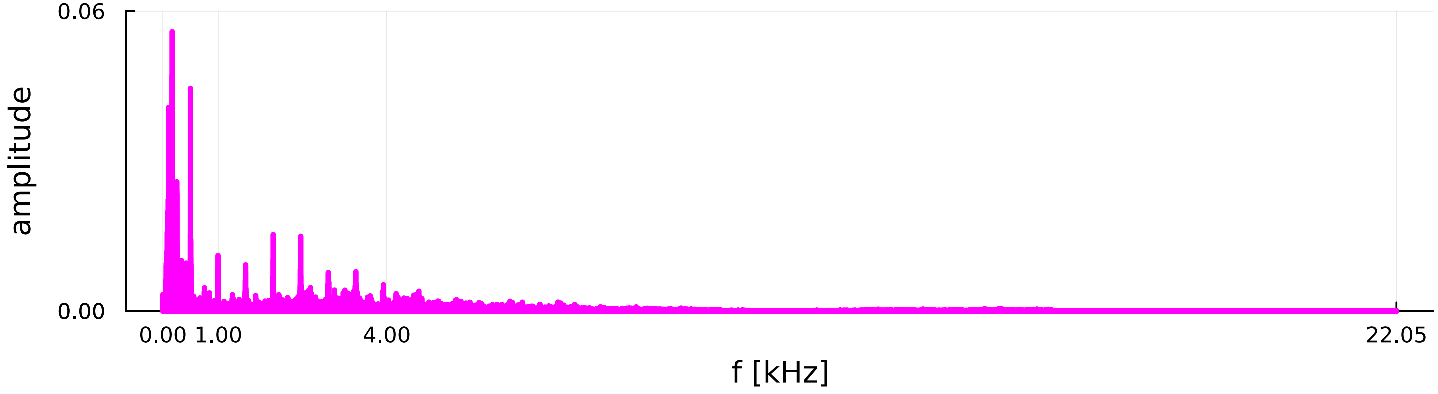
play
https://en.wikipedia.org/wiki/File:Doctor_Who_theme_excerpt.ogg

Part 2: Filters
Filter sweeps / Windows

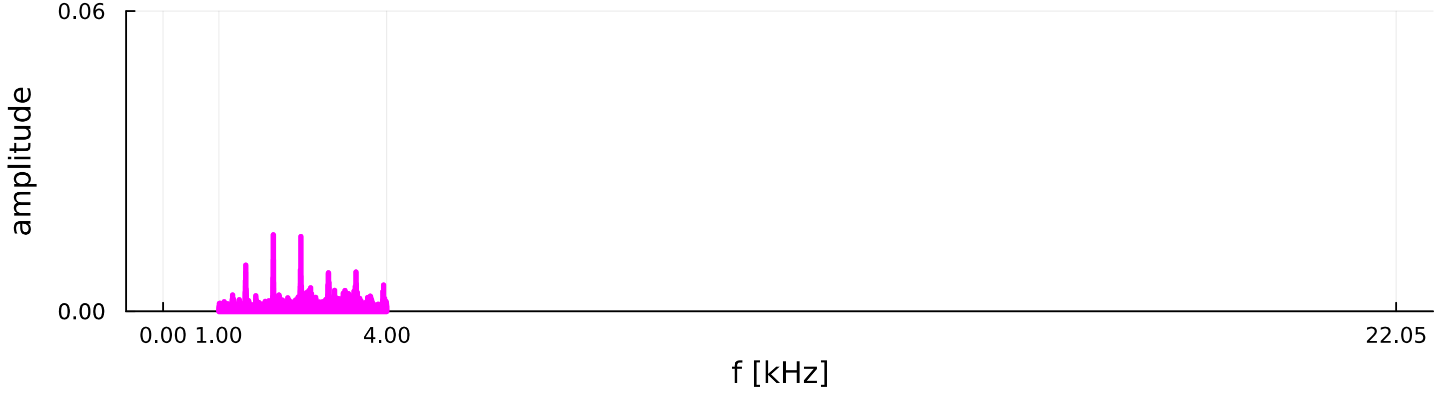
Timbre effects

Band-pass filtering example - spectra

Spectrum of original signal x(t)



Spectrum of filtered signal z(t)



- Retaining only 1000-4000 Hz frequency components gives an “AM Radio” effect.
- This DSP operation is called band-pass filtering.
- You removed MSU song in Lab3 with a low-pass filter.

Band-pass filter code

```
using FFTW: fft, ifft

function filter_bp(x::AbstractVector, lo::Real, hi::Real, S::Real)
    N = length(x)
    cutoff_hz = [lo, hi] # frequency range to retain (pass)
    cutoff_index = round.(Int, cutoff_hz/S*N) # k = (f/S)*N
    fx = fft(x) # spectrum
    fz = zeros(eltype(fx), size(fx))
    pass = (1+cutoff_index[1]):min(1+cutoff_index[2], N)
    fz[pass] .= fx[pass] # pass band
    return 2*real(ifft(fz)) # convert back to time domain
end
```

- `x` input signal vector
- `lo` low-frequency cutoff in Hz
- `hi` high-frequency cutoff in Hz
- `S` sampling rate in Hz
- *cf.* Lab 3
- $k = (f/S)N$

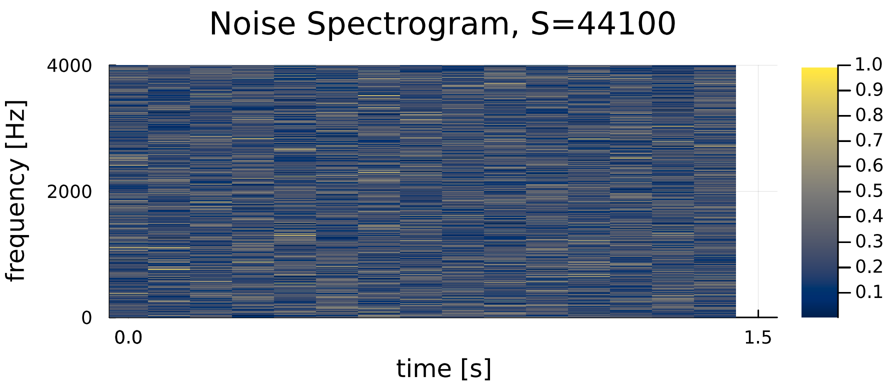
Music filtering example - ala Lab 3

```
# fig_filter1.jl illustrate band-pass filtering
using WAV: wavread
include("filter-bp.jl") # filter_bp()

filter_bp(x::AbstractMatrix, lo, hi, S) = # for stereo!
   mapslices(col -> filter_bp(col, lo, hi, S), x; dims=1)

N = Int(1e5)
x, S = wavread("../synth/cars.wav"; subrange=N)
cutoff_hz = [1000, 4000]
z = filter_bp(x, cutoff_hz..., S)
```

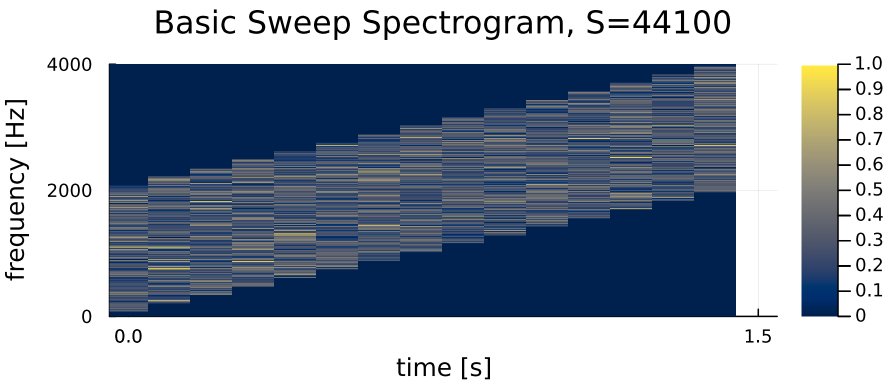
Q0.3 After filtering, what is the lower bound on the sampling rate (in Hz) needed to avoid aliasing?

??

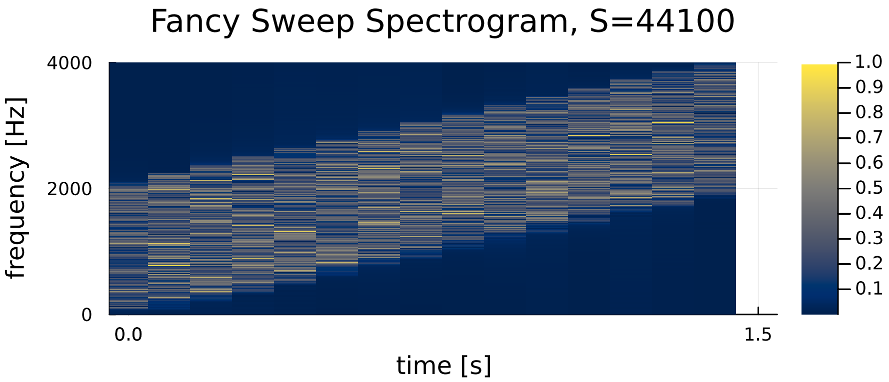
Effect: Band-pass filter sweep with noise



play



play



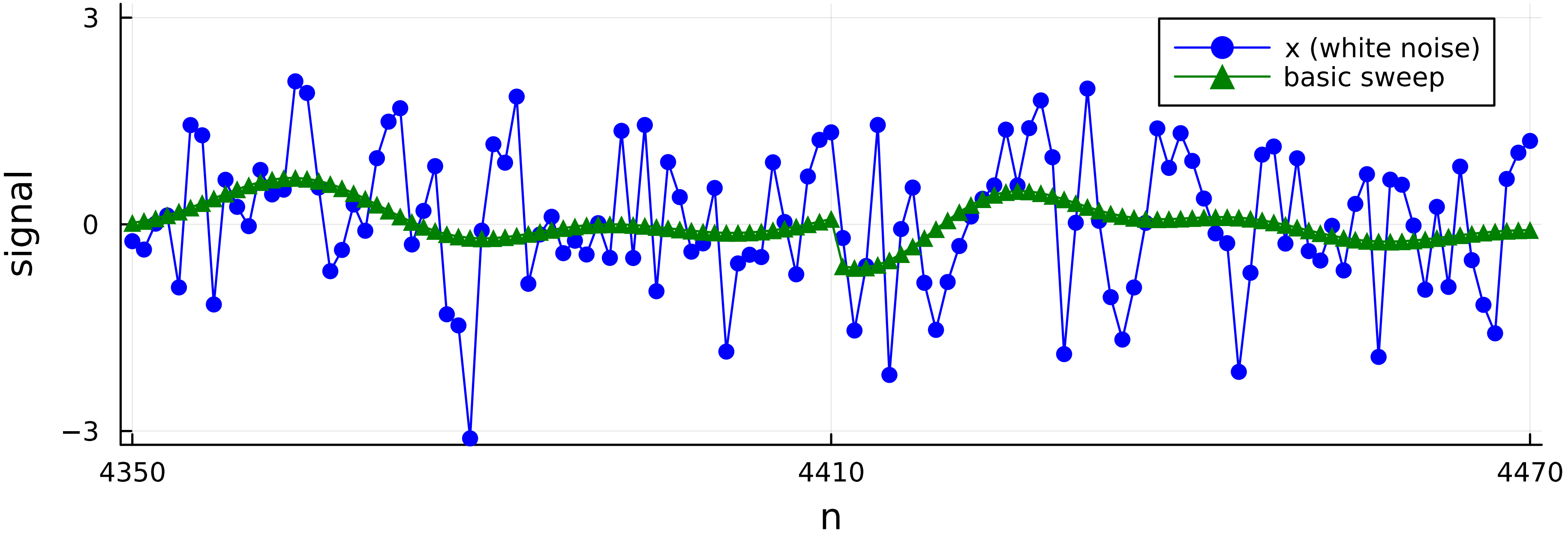
play

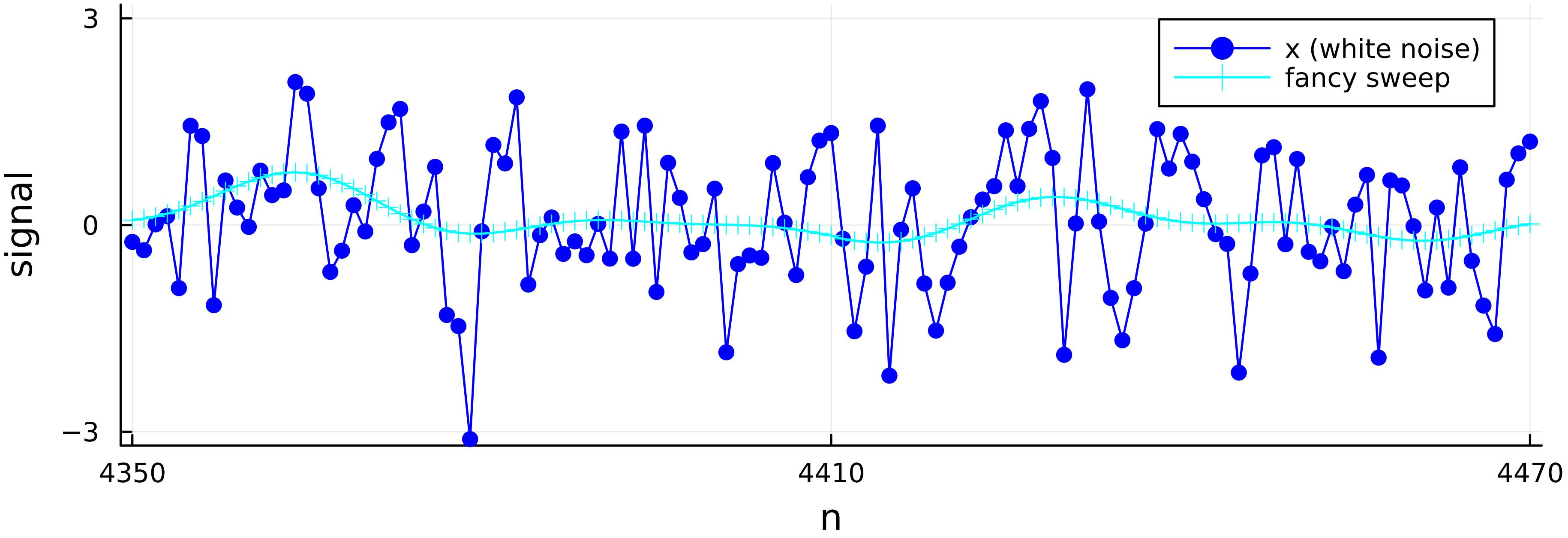
precursor to a dubstep drop?

Sweep code: Basic version

```julia
S = 44100
N = Int(1.5 * S) # 1.5 seconds
x = randn(N) # random noise
M = S ÷ 10 # 0.1 sec segments
Nseg = N ÷ M

cutoff_lo = range(100, 2000, Nseg)
cutoff_hi = cutoff_lo .+ 2000

y = zeros(size(x))
for seg in 1:Nseg
    index = (1:M) .+ (seg-1)*M
    y[index] = filter_bp(x[index], cutoff_lo[seg], cutoff_hi[seg], S)
end
```

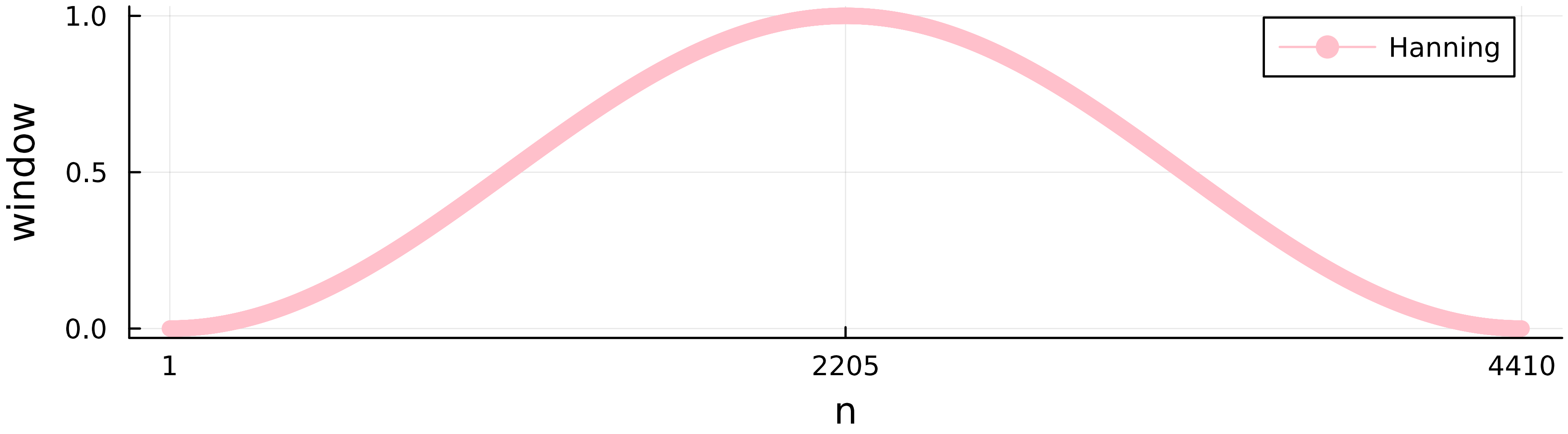
- Apply band-pass filter to each signal segment and directly append segments together.
- Leaves discontinuities at segment boundaries: “click”


x (white noise)
basic sweep
signal
3
0
−3
4350
4410
4470
n


x (white noise)
fancy sweep
signal
3
0
−3
4350
4410
4470
n

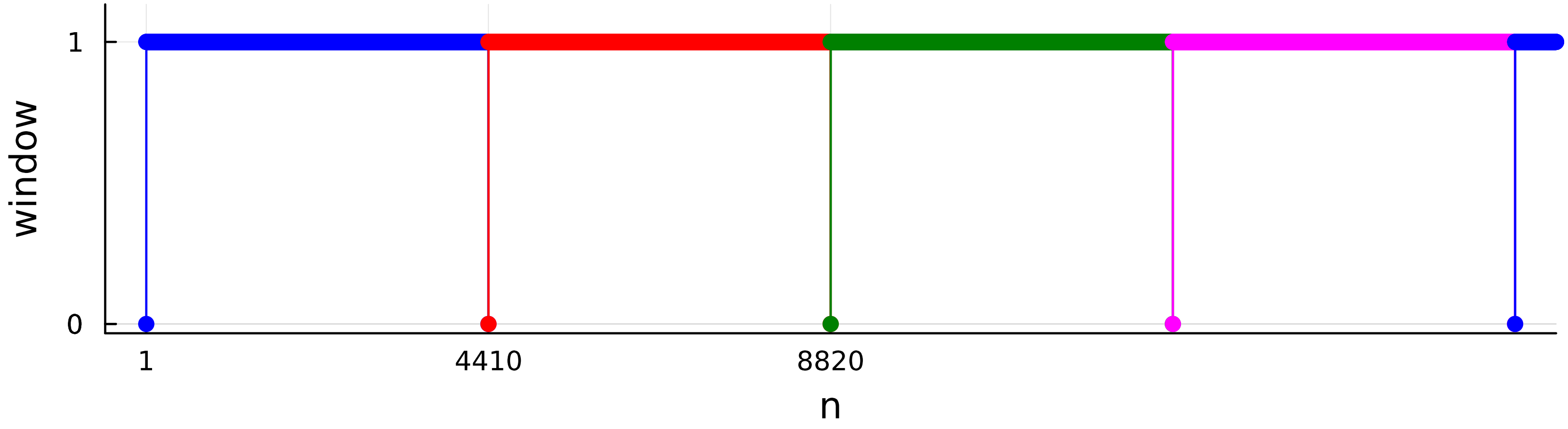
Sweep code: Fancy version

```
using DSP: Windows
window = Windows.hanning(M+1)[1:M] # Hanning window

z = zeros(size(x))
for seg in 1:(2Nseg-1)
    index = (1:M) .+ (seg-1)*(M÷2) # overlapping segments
    xwin = x[index] .*= window # apply Hanning window
    iseg = (seg+1)÷2
    z[index] .+= filter_bp(xwin, cutoff_lo[iseg], cutoff_hi[iseg], S)
end
```

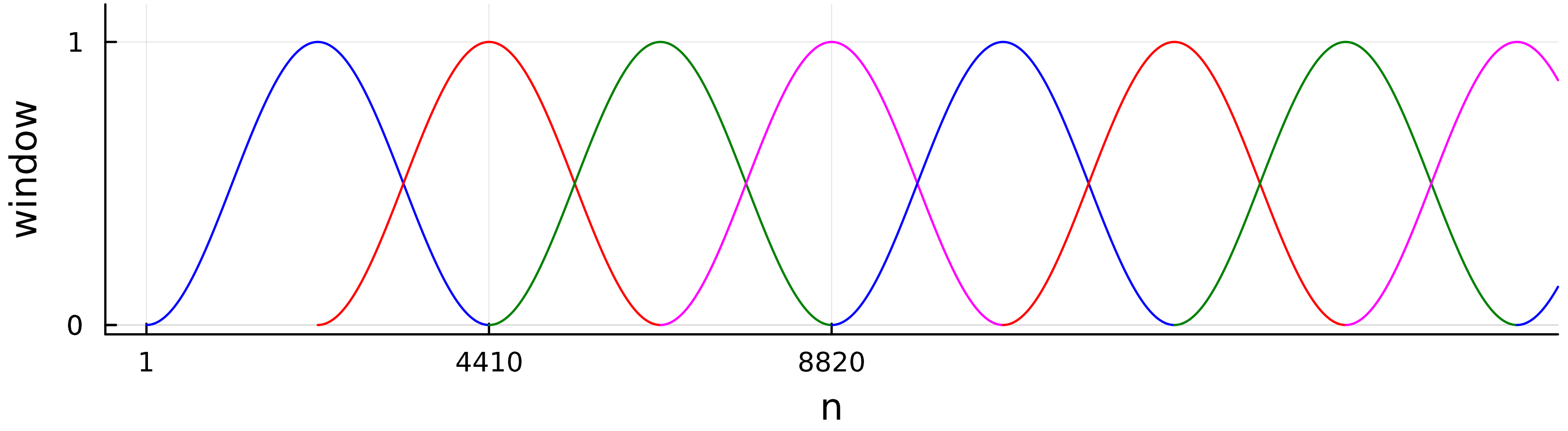


Comparing window functions

Basic (rectangular window):



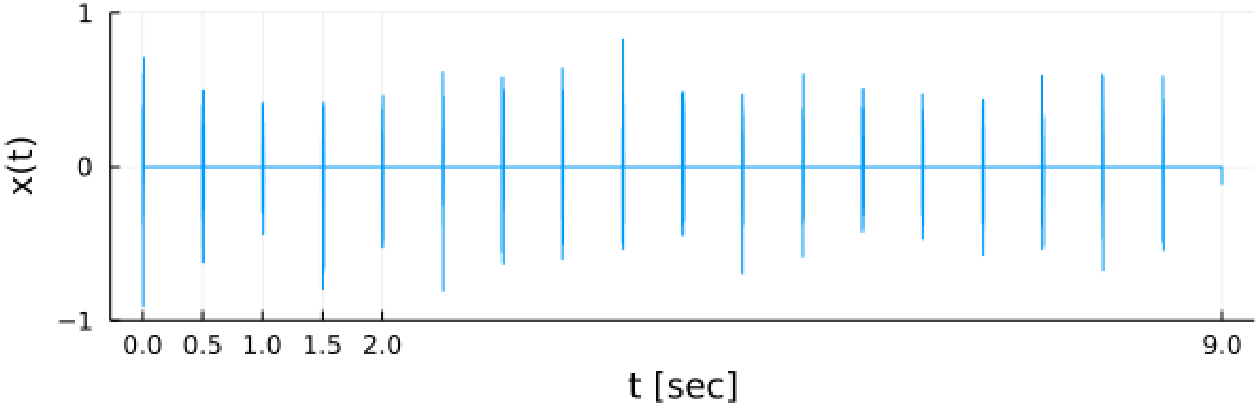
Fancy (Hanning window):



Part 3. Beats per minute (BPM) estimation

Metronome signal

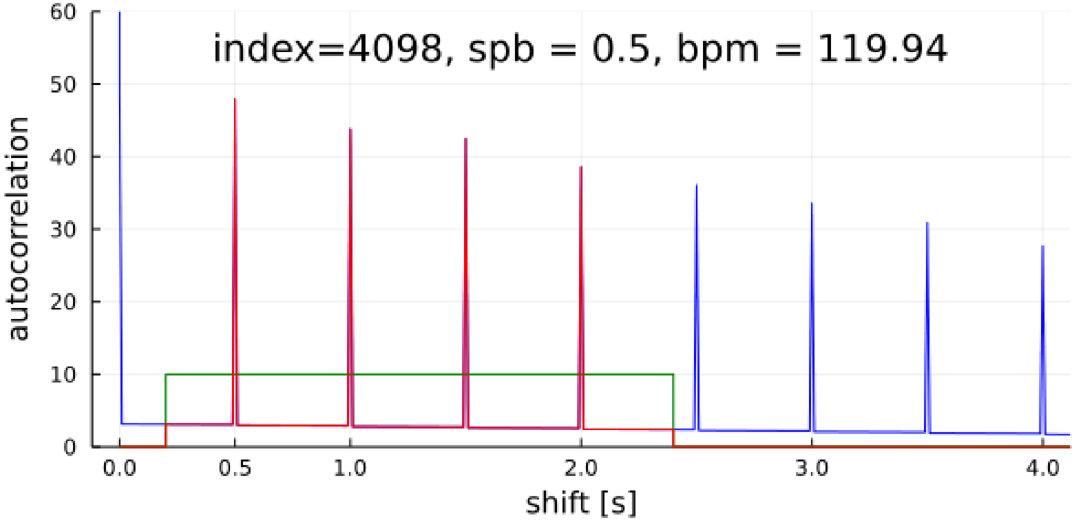
```
# bpm1_gen.jl generate metronome tick signal to test bpm estimator

S = 8192
bpm = 120
bps = bpm / 60 # beats per second
spb = 60 / bpm # seconds per beat
t0 = 0.01 # each "tick" is this long
tt = 0:1/S:9 # 9 seconds of ticking

f = 440
#x = 0.9 * cos.(2π*440*tt) .* (mod.(tt, spb) .< t0) # tone
x = randn(length(tt)) .* (mod.(tt, spb) .< t0) / 4.5 # click via "envelope"
```



play

Metronome BPM

```
# bpm1_find.jl : simple beats-per-minute (BPM) estimator

using FFTW: fft, ifft
using WAV: wavread
x, S = wavread("bpm1a.wav"); x = vec(x); N = length(x)

auto = real(ifft(abs.(fft([abs.(x);zeros(N)])).^2)) # why abs?
spacing = (0:(2N-1))/S # why?
good = (spacing .> 60/300) .& (spacing .< 60/25) # min and max reasonable bpm
index = argmax(auto .* good) # highest correlation for reasonable bpm range
tmp = "index=$index, spb = $(round(index / S, digits=3)), bpm = $(round(60*S/index, digits=2))"

using Measures: mm
using Plots; default(label="", size=(600,300), left_margin = 2mm, bottom_margin = 4mm)
plot(spacing, auto, color=:blue, annotate = (2, 55, tmp), xwiden=true)
plot!(spacing, 10*good, color=:green, xlims=(0,4), ylims=(0,60), xticks=[0; 0.5; 1:4])
plot!(spacing, auto .* good, color=:red, xlabel="shift [s]", ylabel="autocorrelation")
#savefig("fig_bpm1b.png")
```



BPM Summary

Q0.4 The preceding "metronome" signal is periodic ??

A: True B: False ??

This small example illustrates several useful ideas.

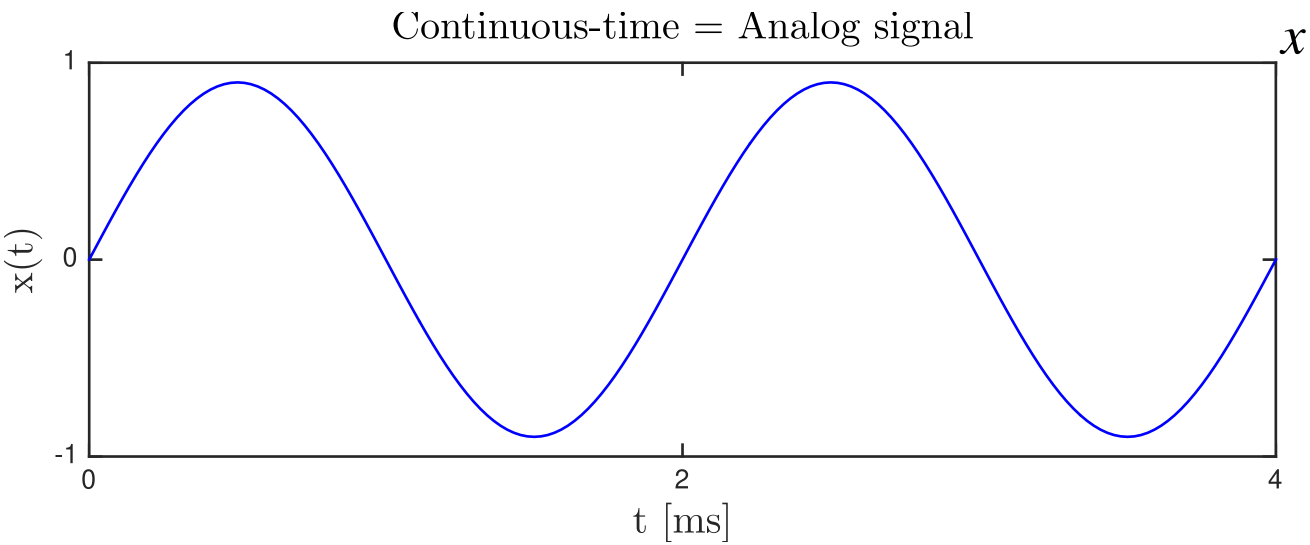
- Noise blips
- Using modulo `mod` for repeating patterns
- Using logical operations like `<` to make binary signals.
- Constraining `argmax` to reasonable search range
- Looking for correlation between bursts of noisy signals using `abs`
- A few lines of `Julia` code can do sophisticated DSP operations

Summary: (auto)correlation is quite widely useful

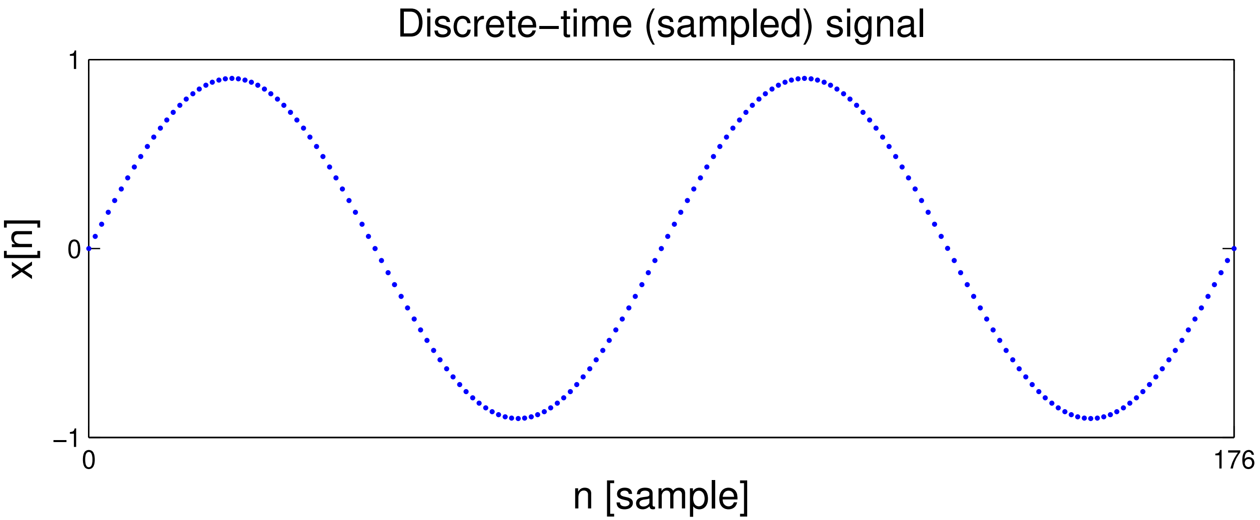
2017 DSP challenge on real-time beat tracking:
http://signalprocessingsociety.org/uploads/email/SP_Cup_2017.html

Part 4. Digital signals and quantization

Continuous-time signals and discrete-time signals



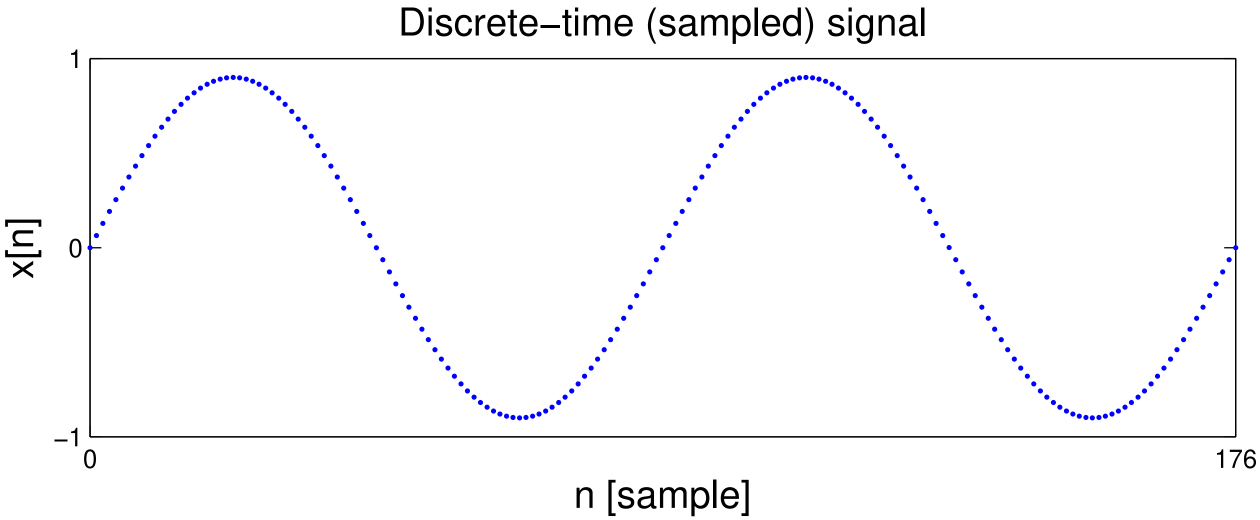
$$x(t) = \cos(2\pi f t)$$



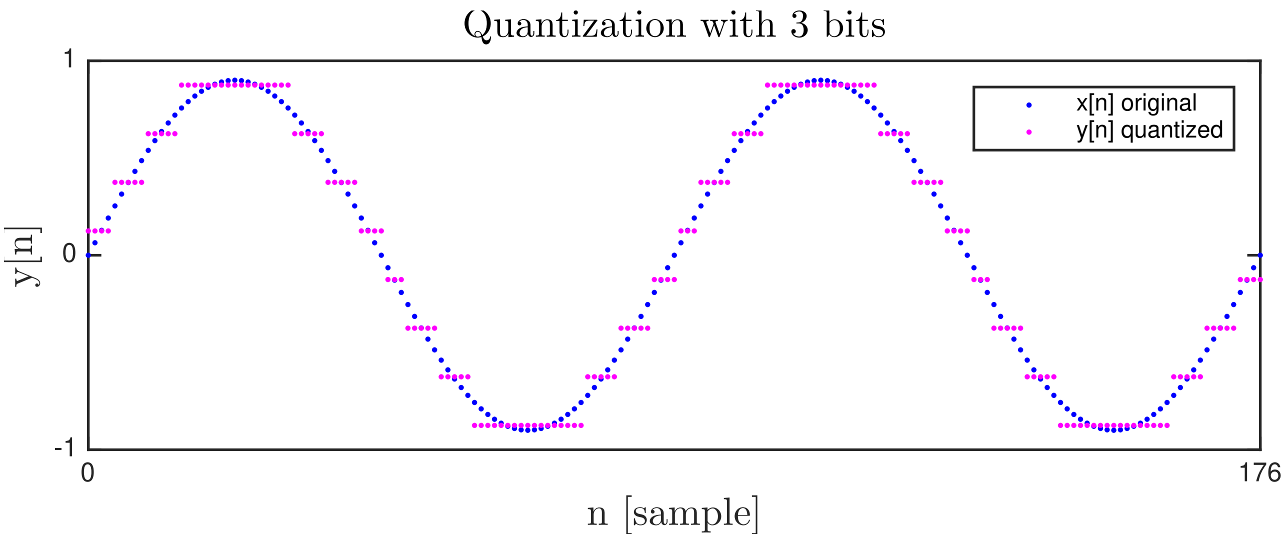
$$x[n] = \cos(2\pi f n / S)$$

play

Digital signals and quantization

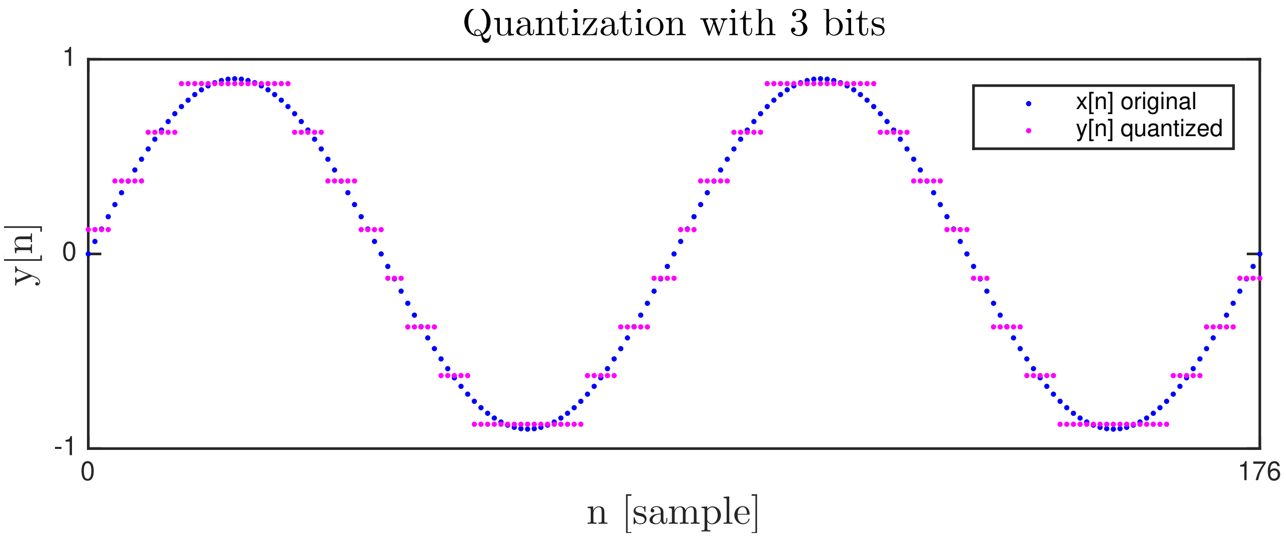


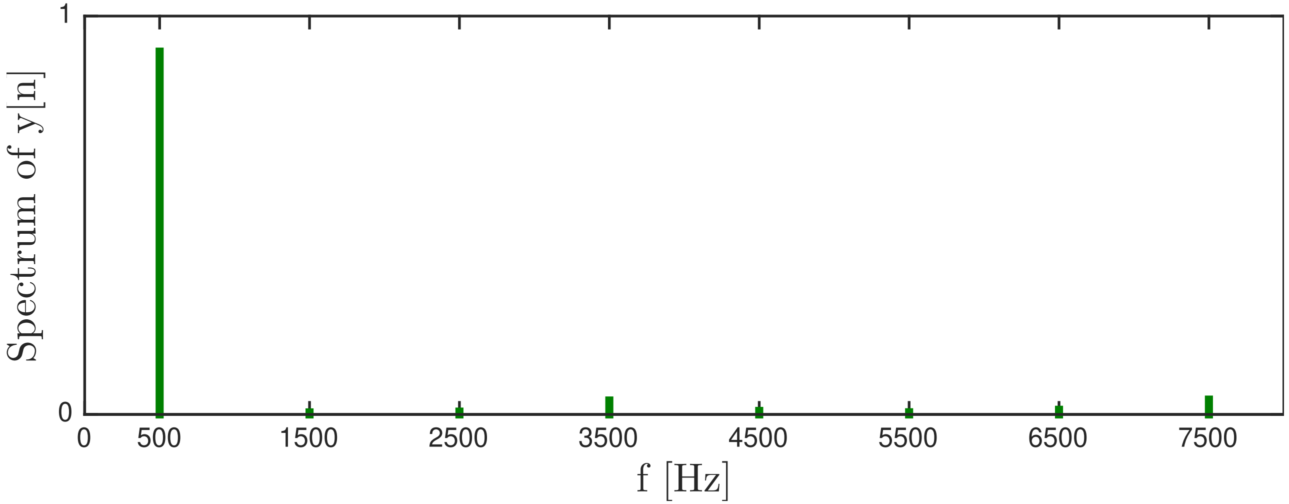
play



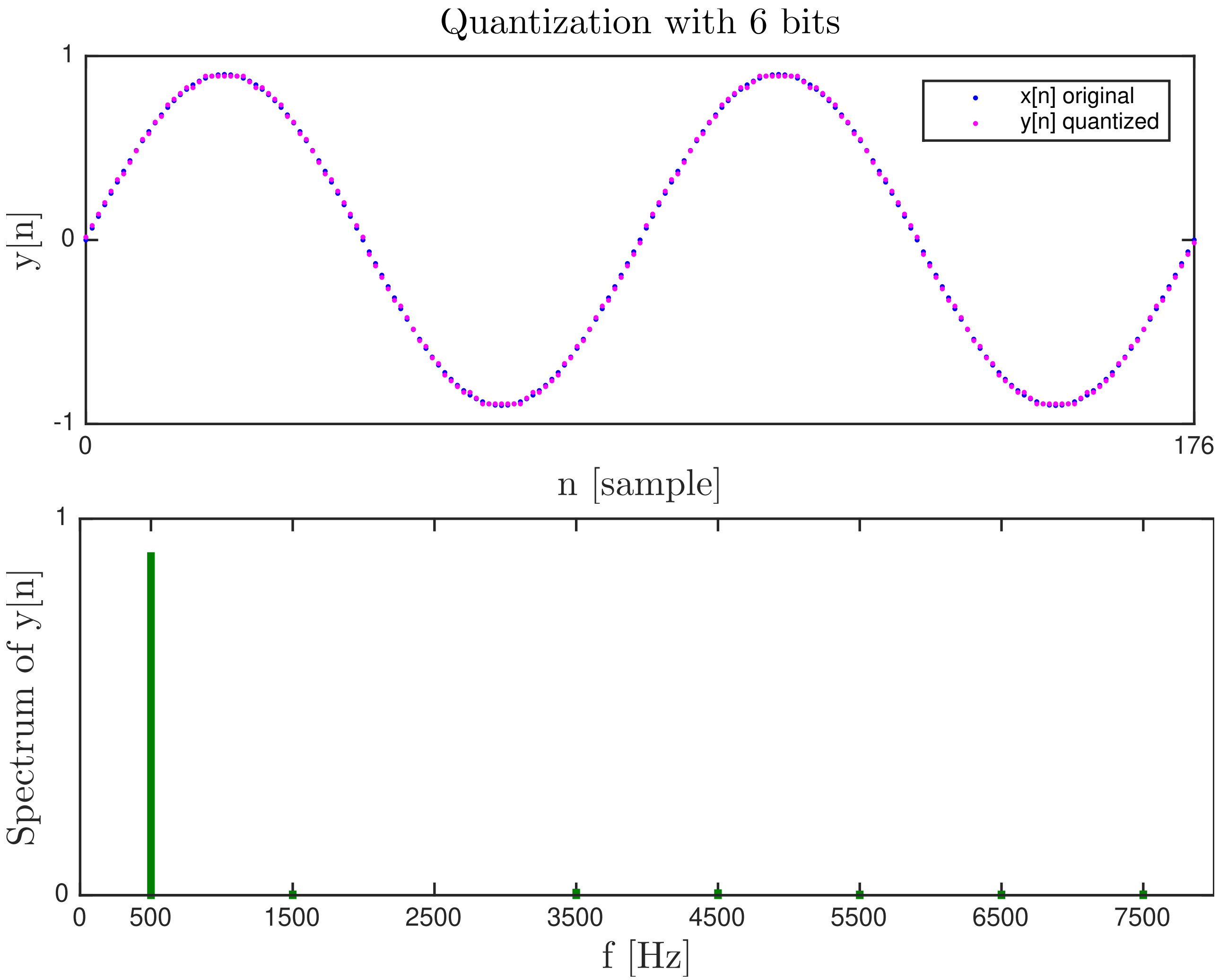
play

Spectrum of quantized signal (3 bits)





Spectrum of quantized signal (6 bits)



original: play 3 bit: play 6 bit: play 8 bit: play

The default for `.wav` files is 16 bits. (Accepts 8, 16, 24, or 32.)

Part 5. Project 3 tips

Using reshape to simplify indexing

Given a `Julia` vector `x` containing samples of 3 instruments, each playing 12 notes, with `N = 1000` samples per note.

How do we access the 4th note of the 3rd instrument?

One way: `y = x[27001:28000]`

Slightly better way: `y = x[27000 .+ (1:N)]`

Still better way: `y = x[(2*12+3)*N .+ (1:N)]`

Elegant way using 3D array slicing:

`y = reshape(x, N, 12, 3)[:,4,3]`

Transcriber hints: note durations / locations

P3 classic synthesizer includes 100 zeros at end of each note to help find note duration.

Note	Whole	Half	Quarter	1 second
Length	$32668 + 100$	$16284 + 100$	$8092 + 100$	$S = 44100$
	$32768 = 4 \times 8192$	$16384 = 2 \times 8192$	8192	

Transcriber must locate those zeros. How?

```
N = 8192 # quarter note (shortest)
a = reshape(x, N, :) # N × M
b = a[end-99:end,:] # possible zeros
c = sum(abs, b, dims=1)
e = findall(iszero, vec(c))
f = [0; e[1:end-1]]
```

Sizes of each variable? $\boxed{??}$ $\boxed{??}$ $\boxed{??}$ $\boxed{??}$ $\boxed{??}$ $\boxed{??}$

Touch-tone debrief

Dual-Tone-Multi-Frequency (DTMF) standard began use in 1963.

DTMF was originally decoded by tuned (analog) filter banks.
By late 20th century, DSP became the predominant technology

Hertz	1209	1336	1477
697	1	2	3
770	4	5	6
852	7	8	9
941	*	0	#

BUSY SIGNAL	480 Hz & 620 Hz
DIAL TONE	350 Hz & 440 Hz

- Why not one frequency per button?
- Why these frequencies?