Perception & Multimedia Computing

Week 20 – Big wrap-up

Michael Zbyszyński
Lecturer, Department of Computing
Goldsmiths University of London

Topics this term

Signals as sinusoids

Fourier Analysis

Speech & Music Perception

Signals & Systems

- *FIR & IIR Filters*

Sliding windows (FFT)

Perceptual Audio Features

Planned slides for weeks 16 -19 are on the VLE

Mick & I are planning review sessions

- Near beginning of Term 3
- Covering Term 1 & 2 material

Today

FIR & IIR Filters

Sliding windows (FFT)

Perceptual Audio Features

Any signal can be represented as weighted sum of delayed unit impulses.

The response of a LTI system for this signal is a weighted sum of delayed impulse responses.

$y[n] = x[n] * h[n]$ for any $x[n]$

This means that convolution with h[n], the impulse response of H, is by definition equivalent to applying the system H to a signal!

Real-world example

I like how music sounds when played in this giant cathedral. I wish I could make all my music sound like it's recorded here.

What is system?

Cathedral

What is impulse response?

Live recording of cathedral when an impulse is played in cathedral

How to apply cathedral reverb to any new sound?

Convolve new sound with impulse response

Filter equations

What is a filter equation?

the current output

Express y[n]
as a weighted sum of
x[n], x[n-1], x[n-2], etc.

the current
input value

the input value
1 sample ago
(i.e., the previous input)

the input value
2 samples ago

Generic form

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] + \ldots + b_Q * x[n-Q]$$

b_0, b_1, etc are called *filter coefficients*

Example:

$y[n] = 0.5 * x[n] + 1 * x[n-1]$

What is output when x = [1, 2] ?

Impulse response & filter equation

$h[n] = [b_0, b_1, b_2, \ldots, b_Q]$

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] + \ldots + b_Q * x[n-Q]$$

check by computing y when x is unit impulse [1]

Exercises

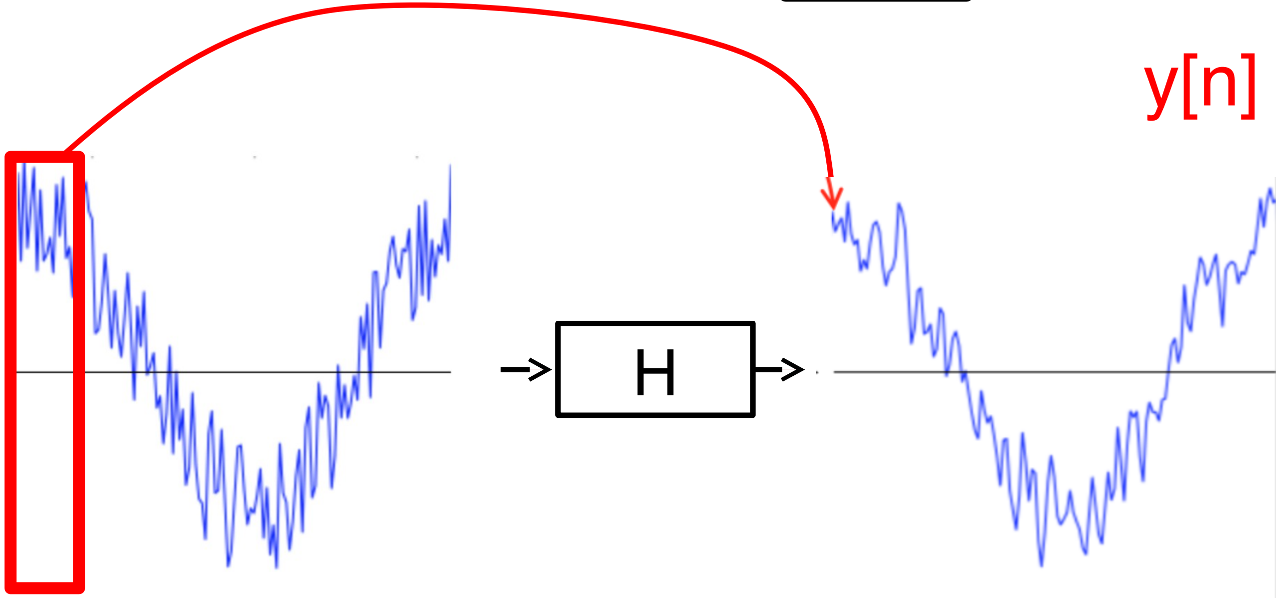
Convolve:

1. x = [1, 2, 3], h = [2]
2. x = [2, 3, 4], h = [0.5, 0.5]
3. x = [1, 1, 1], h = [-1, 1]

A simple smoothing system

$$[1] = [1, 0, 0, \ldots] \rightarrow \boxed{H} \rightarrow h[n] = [0.5, 0.5]$$

$$y[n] = .5x[n-1] + .5x[n]$$



A VERY useful property

If $y[n] = x[n] * h[n]$, then $Y_k = X_k \cdot H_k$

In English:

You can compute the output of a system H by convolving the input with the impulse response of the system, h.

You can equivalently compute the spectrum of the output by multiplying the spectrum of the input, bin by bin, with the spectrum of the impulse response (which is the frequency response).

A VERY useful property

Colloquially:

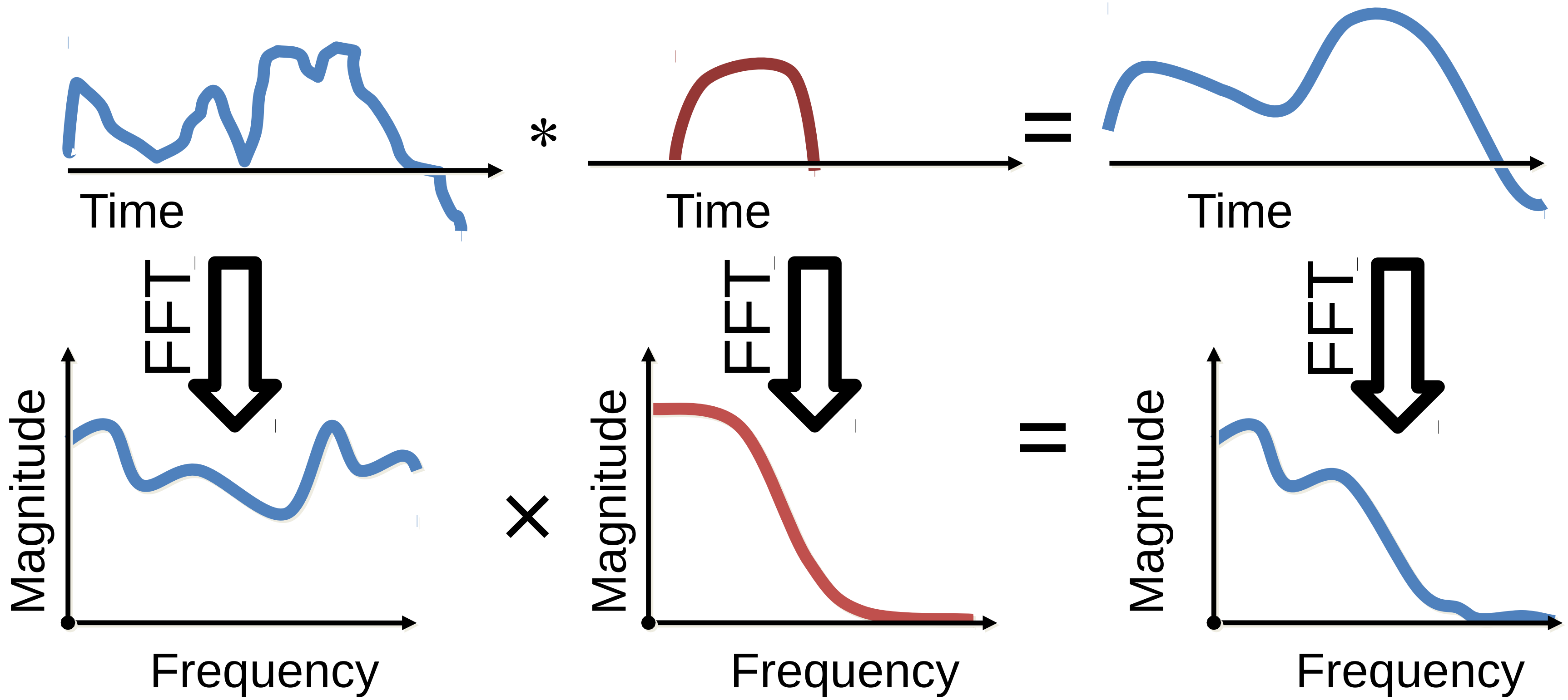
Convolution in the time domain is equivalent to multiplication in the frequency domain.

Also: Multiplication in the time domain is equivalent to convolution in the frequency domain.

(more on this later)

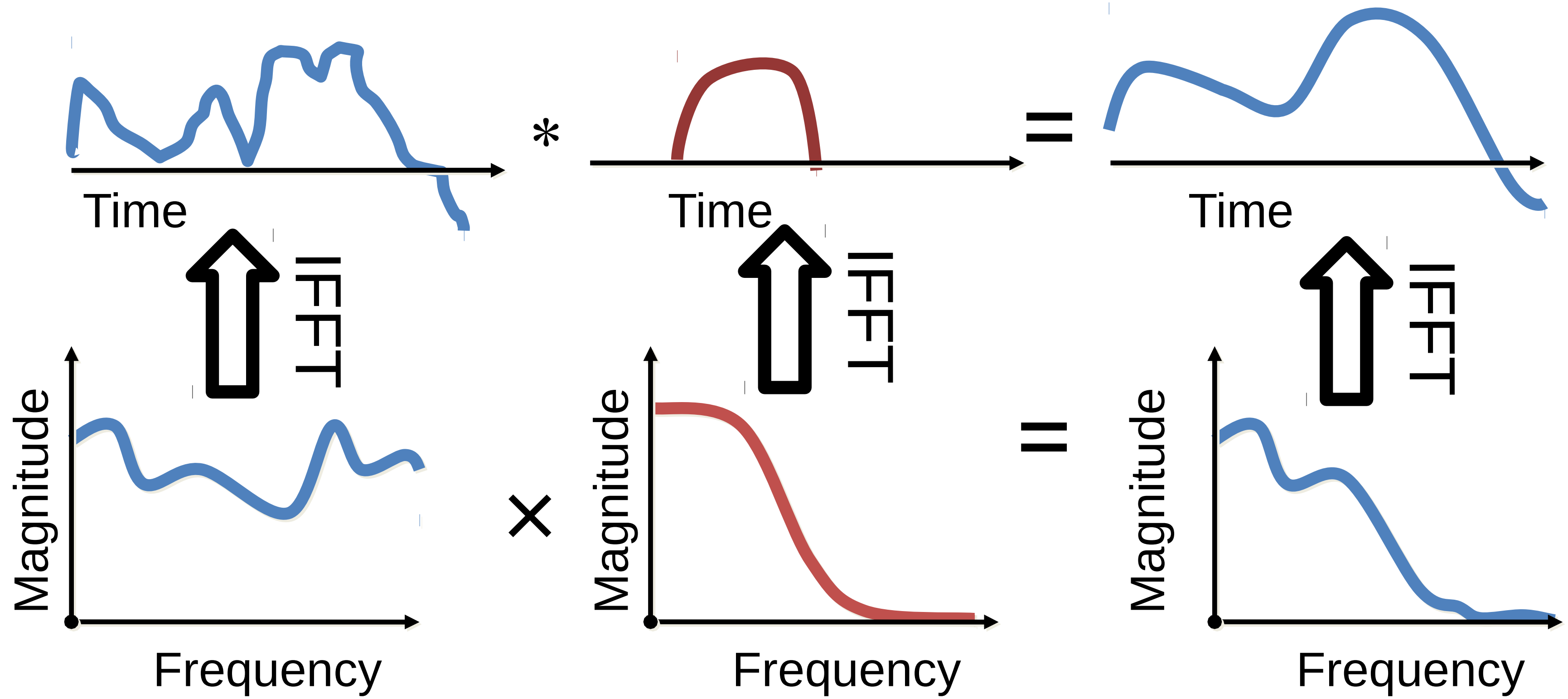
Convolution & Multiplication

Reasoning about system effects on an input:



Convolution & Multiplication

Reasoning about system effects on an input:



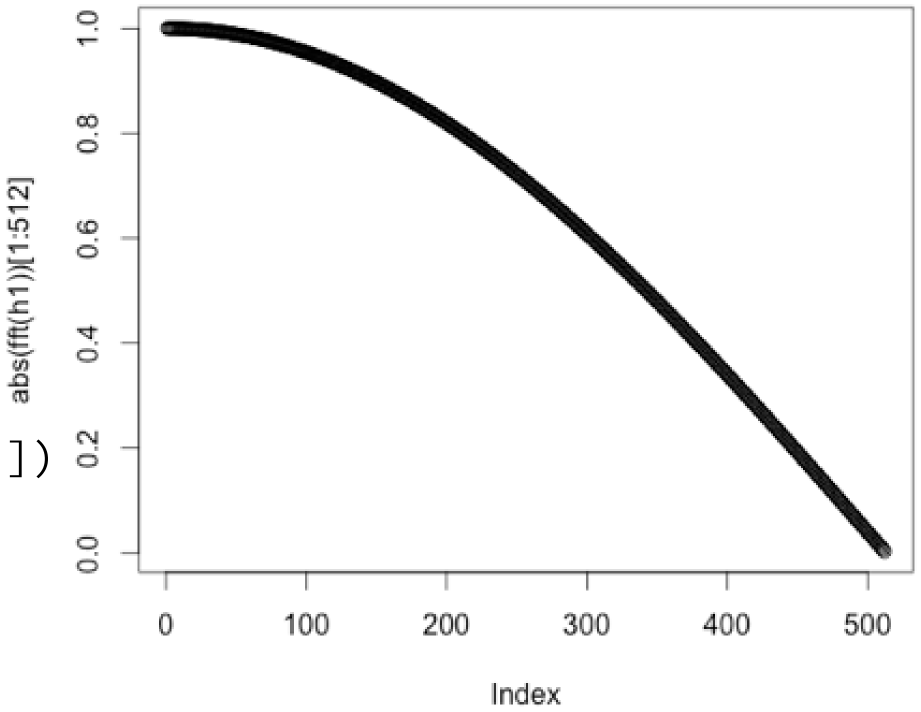
Using this to understand our smoothing filter

If h[n] = [0.5, 0.5]

FFT of h[n] = H_k =

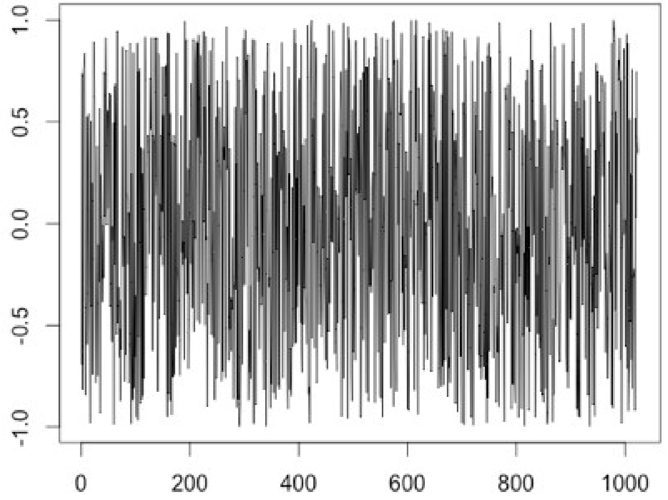
In Python:

```
h = [0.5, 0.5]
h_4096 = concatenate([h, zeros(4094)])
plot(abs(fft.fft(h_4096)[0:2048]))
```



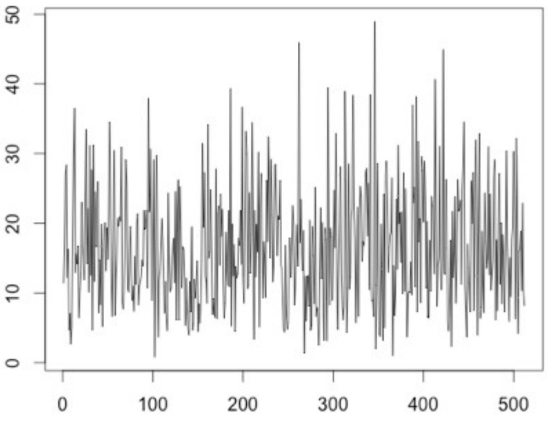
Applying our smoother to random noise

x[n] =



```
noise = (rand(4096)-0.5)*2
plot(noise)
```

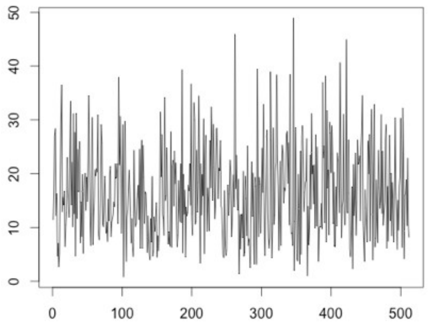
X_k (spectrum) =



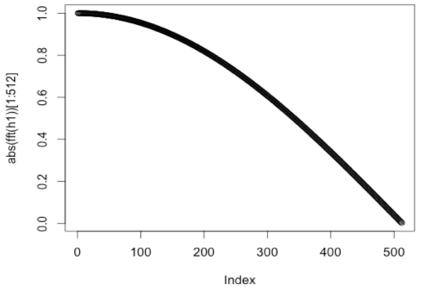
```
plot(abs(fft.fft(noise)[0:2048]))
```

Applying our smoother to random noise

$X_k =$

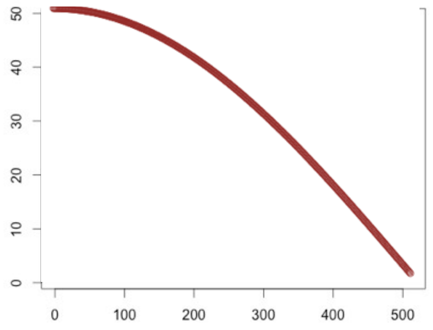


$H_k =$

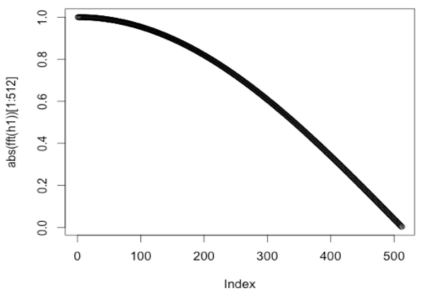


Applying our smoother to random noise

$X_k =$



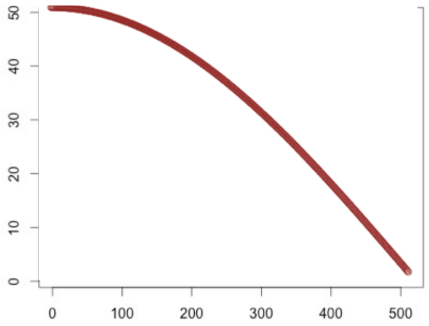
$H_k =$



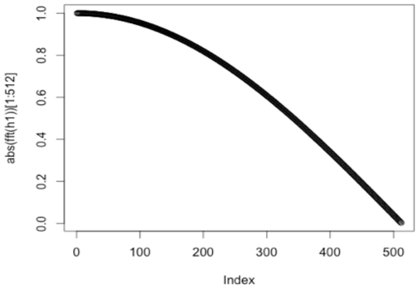
Multiply to get spectrum of $x[n] * h[n]$:

Applying our smoother to random noise

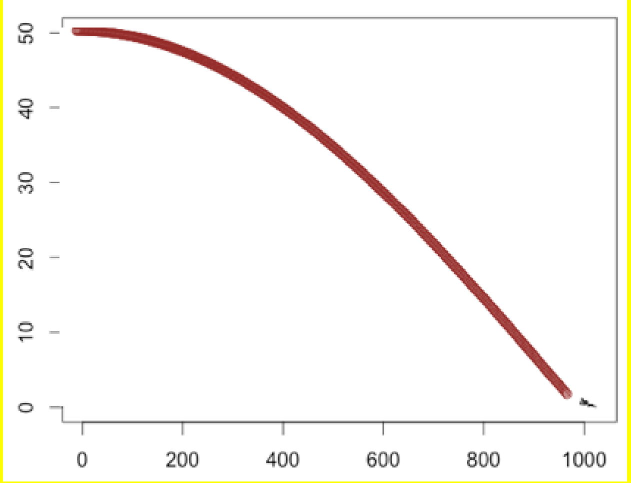
$X_k =$



$H_k =$

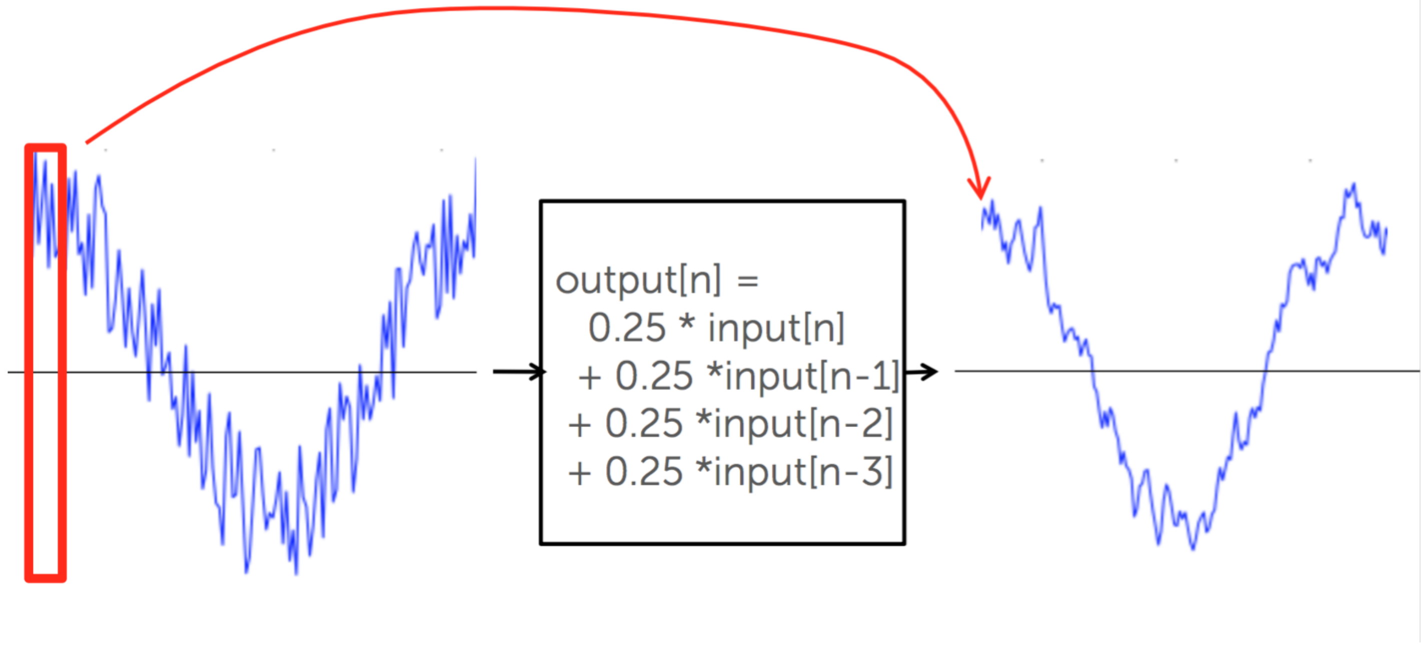


Multiply to get spectrum of $x[n] * h[n]$:



How to improve our smoothing filter?

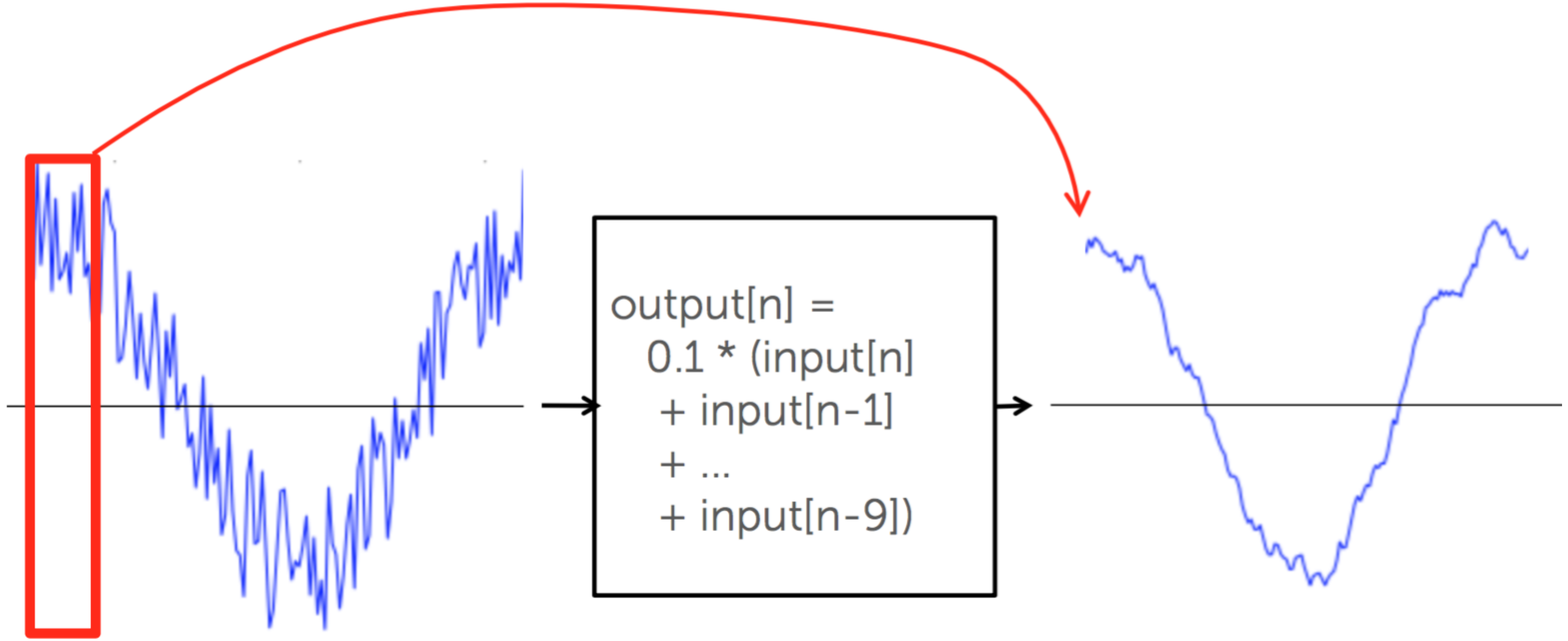
Can use h=[0.25, 0.25, 0.25, 0.25]:
Output is average of last four inputs



output[n] =
0.25 * input[n]
+ 0.25 *input[n-1]
+ 0.25 *input[n-2]
+ 0.25 *input[n-3]

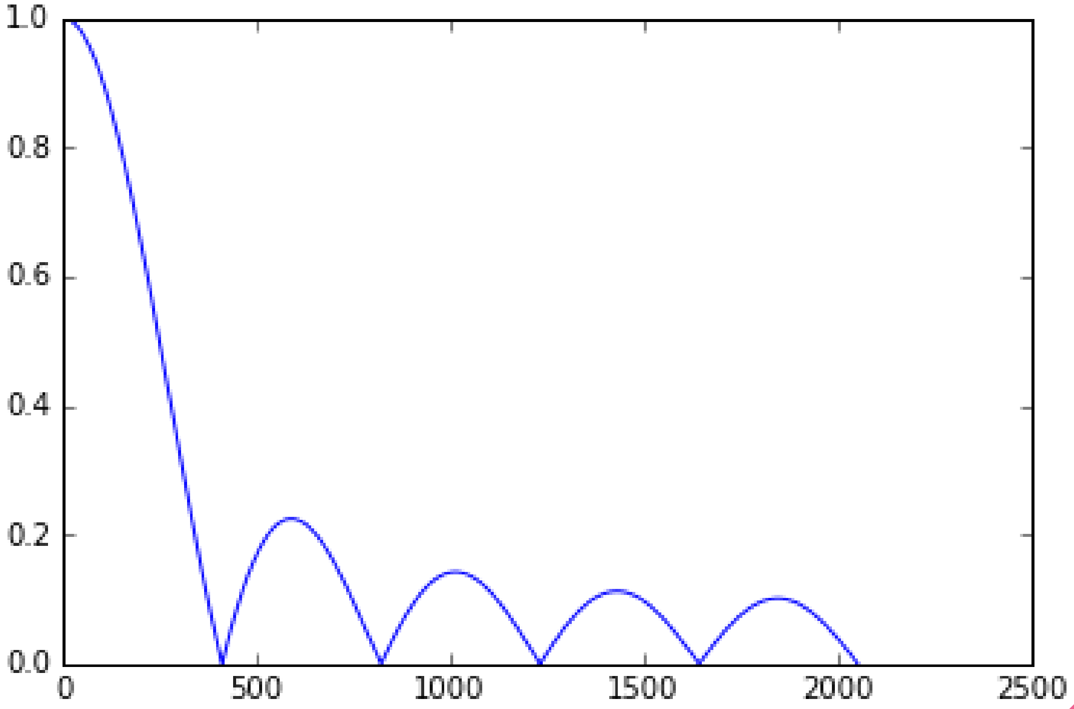
How to improve our smoothing filter?

Can use h=[0.1, 0.1, 0.1, 0.1,0.1, 0.1, 0.1, 0.1, 0.1, 0.1]:
Output is average of last ten inputs



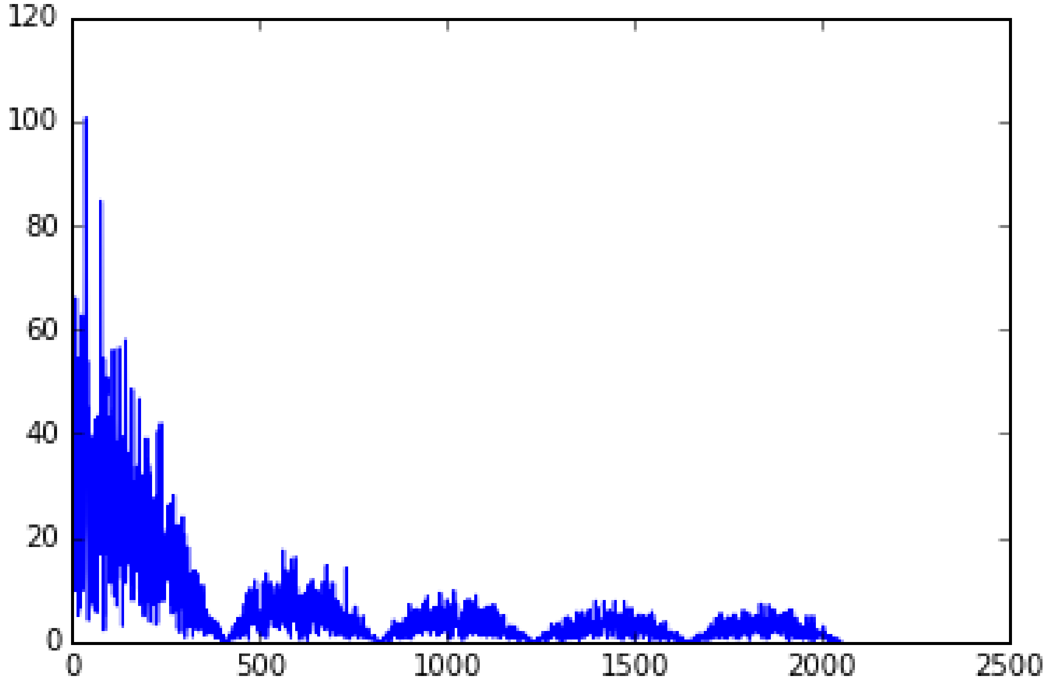
Problem:

Spectrum of h = [0.1, 0.1, 0.1, 0.1,0.1, 0.1, 0.1, 0.1, 0.1, 0.1]



Problem:

Spectrum of noise filtered with this filter



How to build useful effects / systems

Method 1:

Design a useful impulse response

We have to know how we want the time-domain sound signal to be changed by the system.

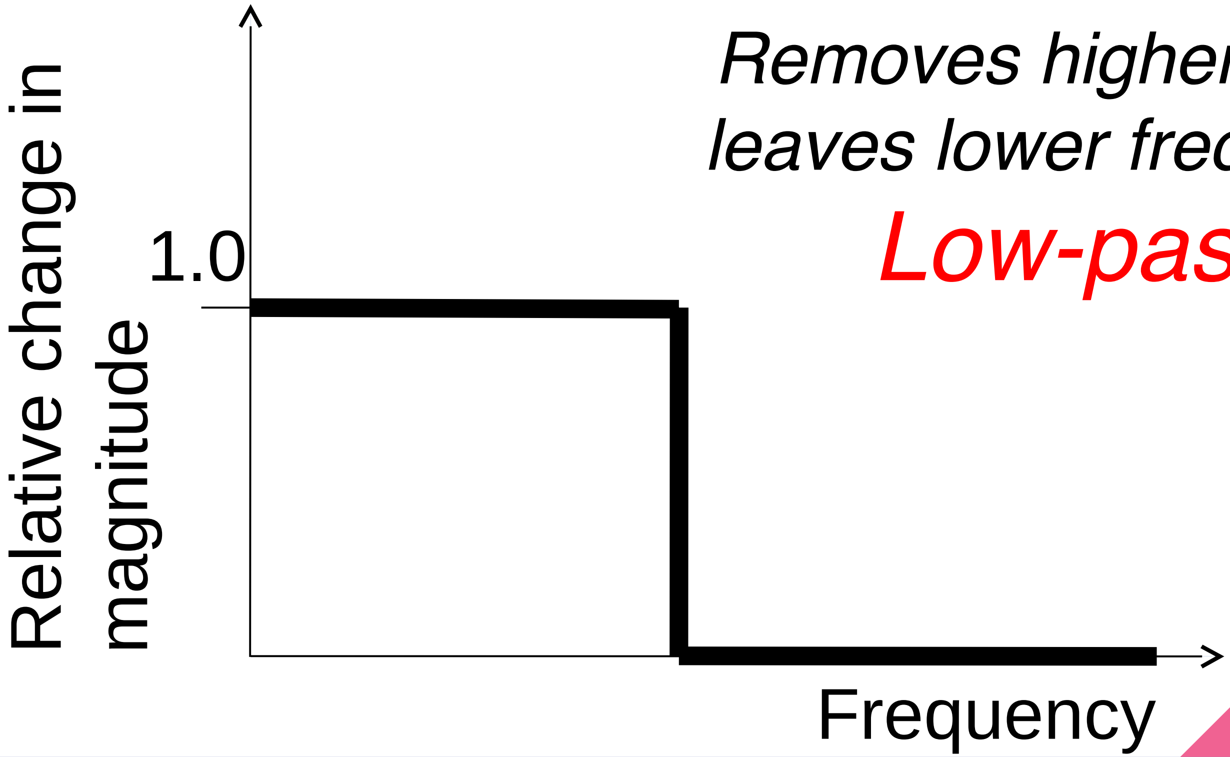
Method 2:

Design a useful frequency response

Instead, we can decide how we want the spectrum of the sound to be changed by the system.

Frequency response

Any LTI system has the ability to change the spectrum of a sound

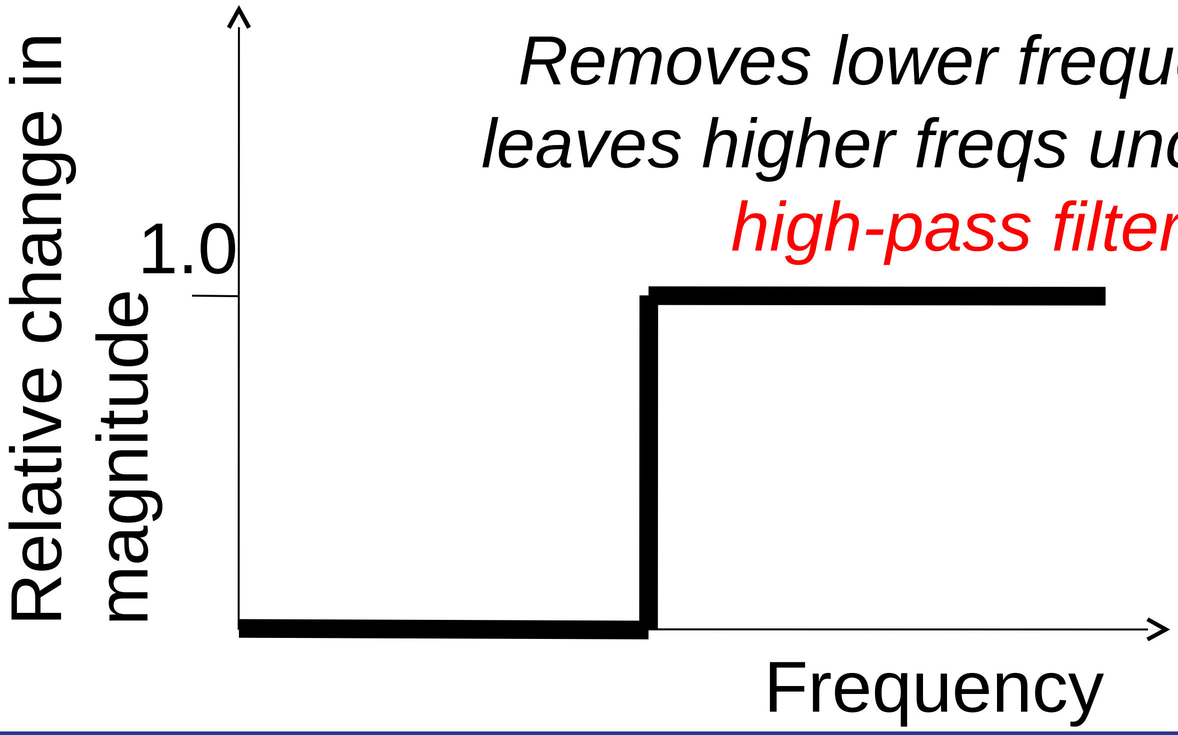


Removes higher frequencies, leaves lower freqs unchanged

Low-pass filter

Frequency response

Any LTI system has the ability to change the spectrum of a sound

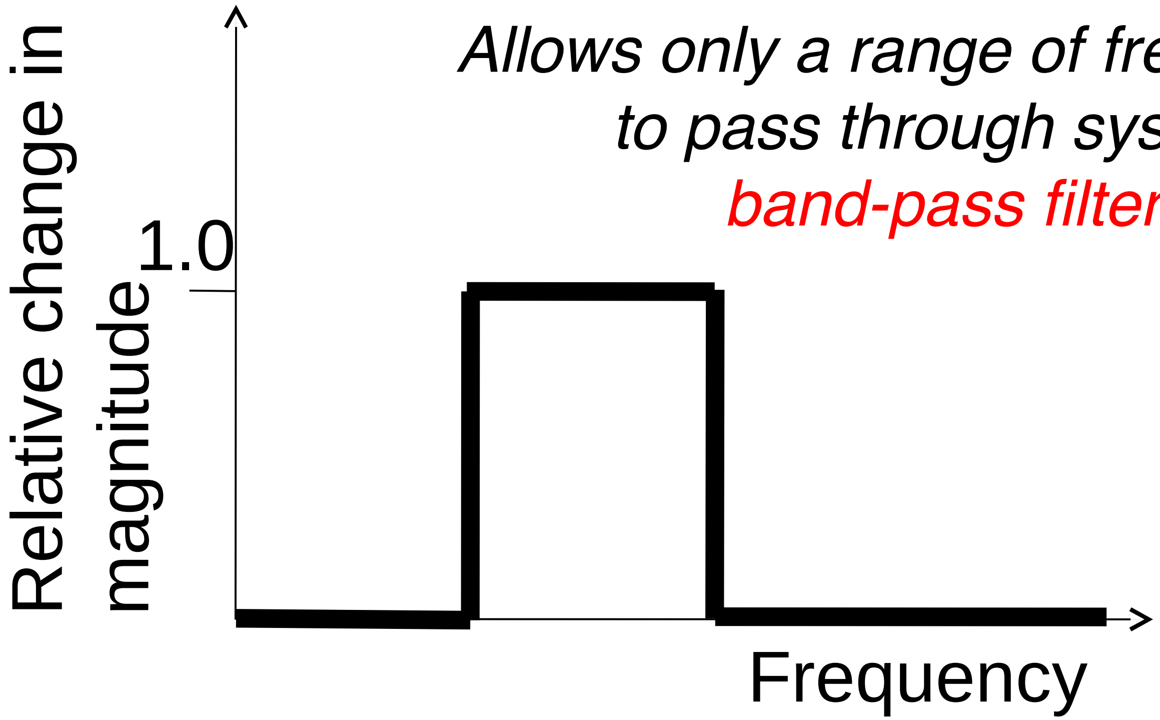


Removes lower frequencies, leaves higher freqs unchanged

high-pass filter

Frequency response

Any LTI system has the ability to change the spectrum of a sound

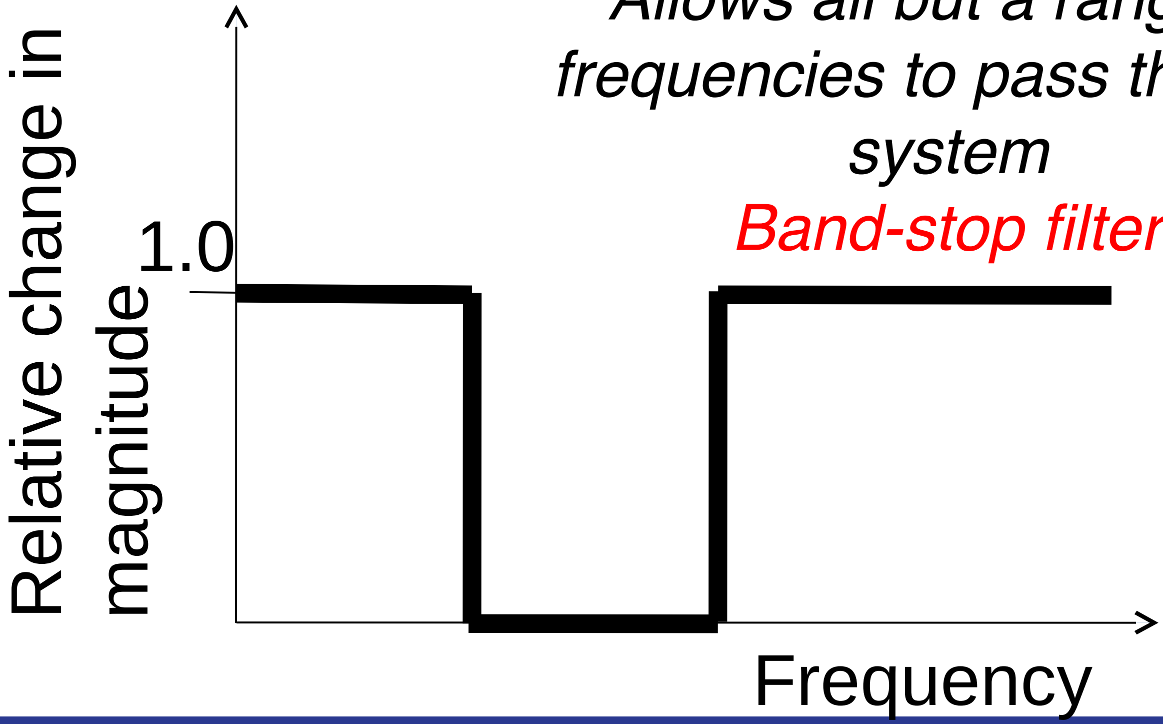


Allows only a range of frequencies to pass through system

band-pass filter

Frequency response

Any LTI system has the ability to change the spectrum of a sound

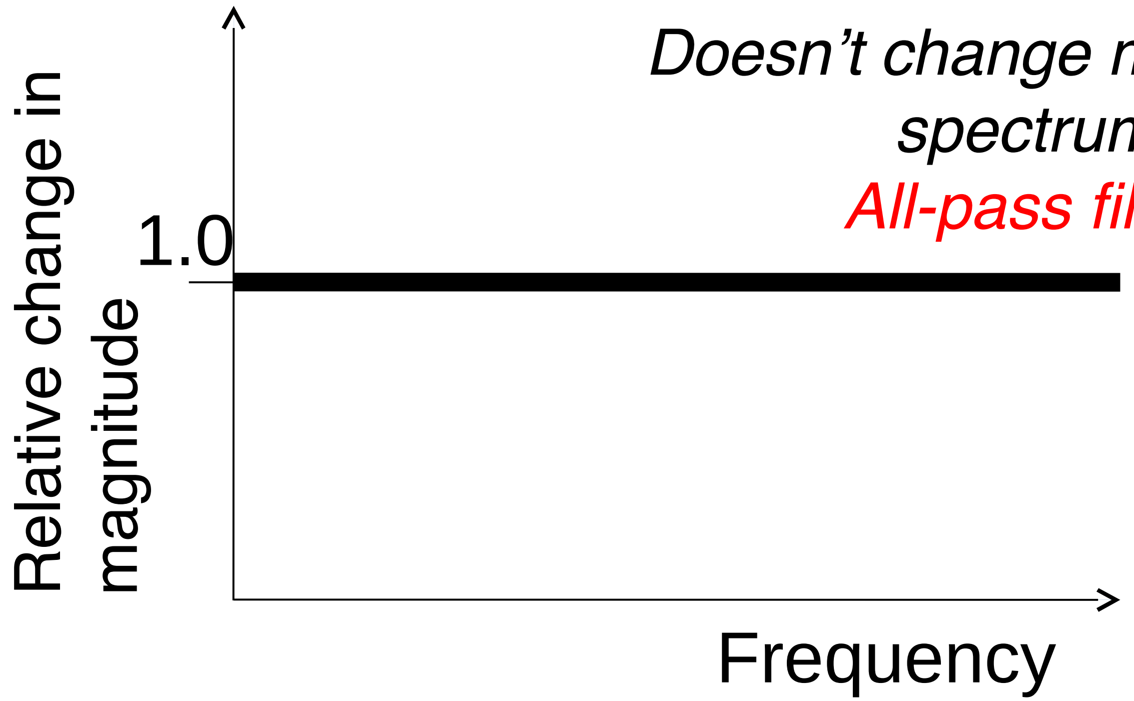


Allows all but a range of frequencies to pass through system

Band-stop filter

Frequency response

Any LTI system has the ability to change the spectrum of a sound

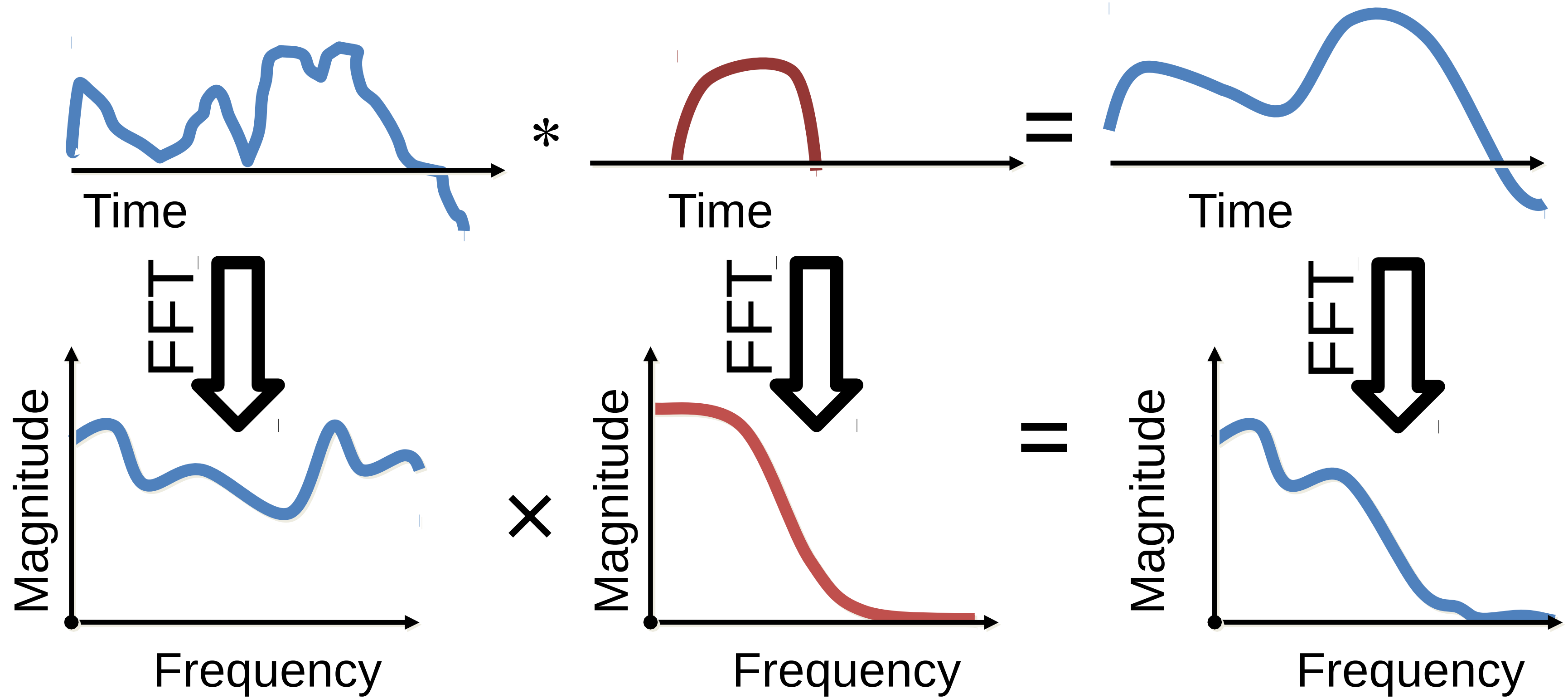


Doesn't change magnitude spectrum

All-pass filter

Convolution & Multiplication

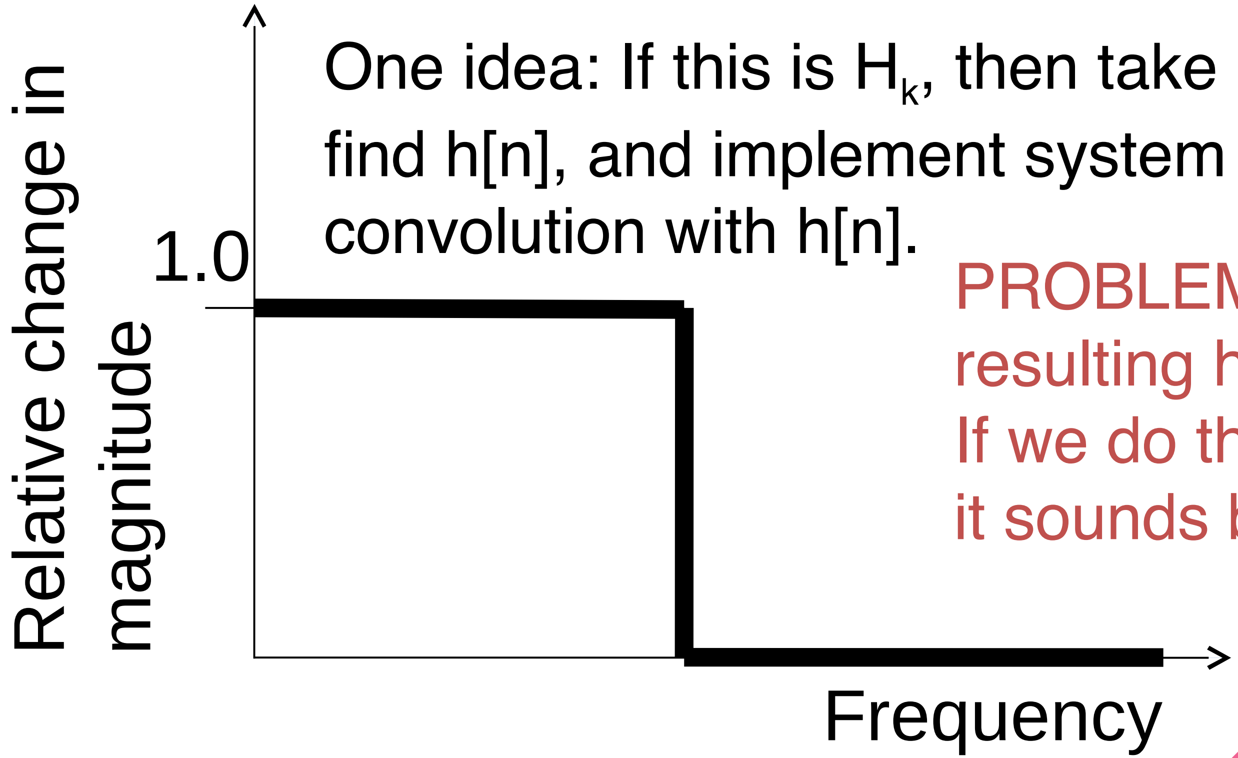
You can start with the desired frequency response.



Consequences

1) Can take the FFT of h[n] to understand what an arbitrary system with known h[n] will do to a spectrum
2) Can design filters to have specific desired effects on the spectrum (e.g., an EQ that boosts midrange, or a low-pass filter used in an ADC)

Constructing a low-pass filter

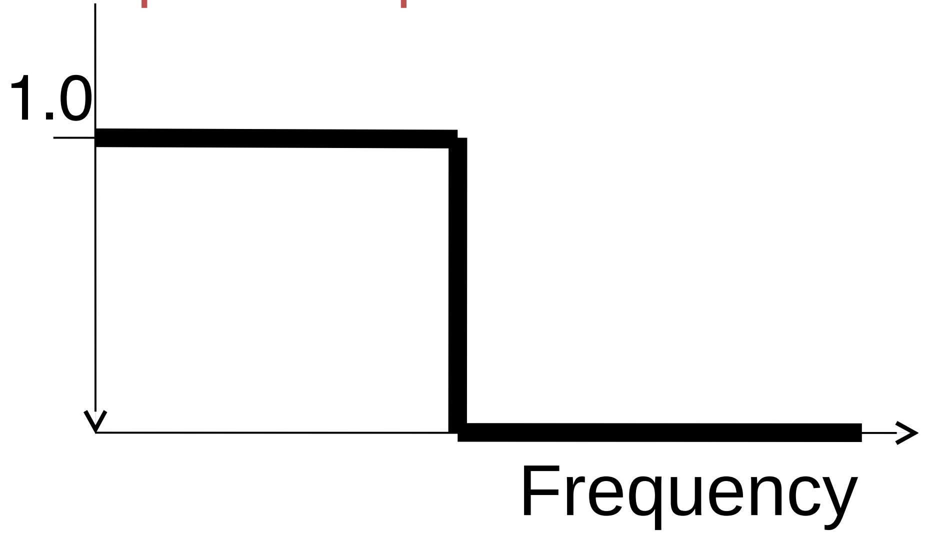


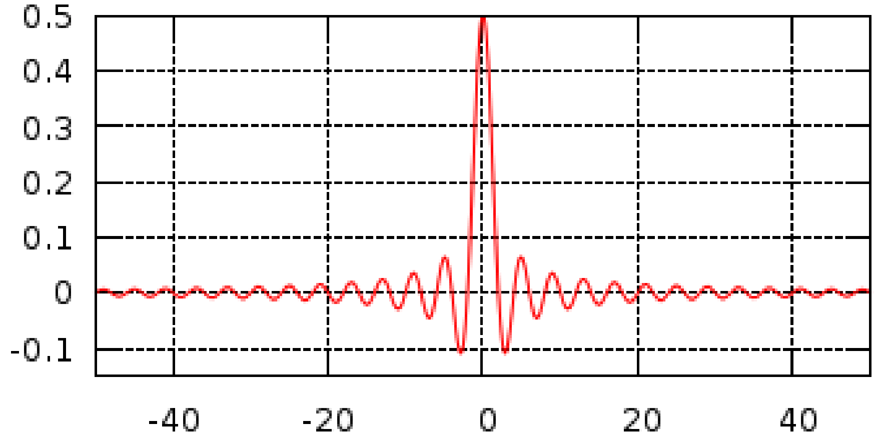
One idea: If this is H_k, then take IFFT to find h[n], and implement system as convolution with h[n].

PROBLEM: if we do this, then resulting h[n] is infinitely long! If we do this directly with FFT, it sounds bad!!

Big problems

The sharper the change in frequency, the longer the impulse response.

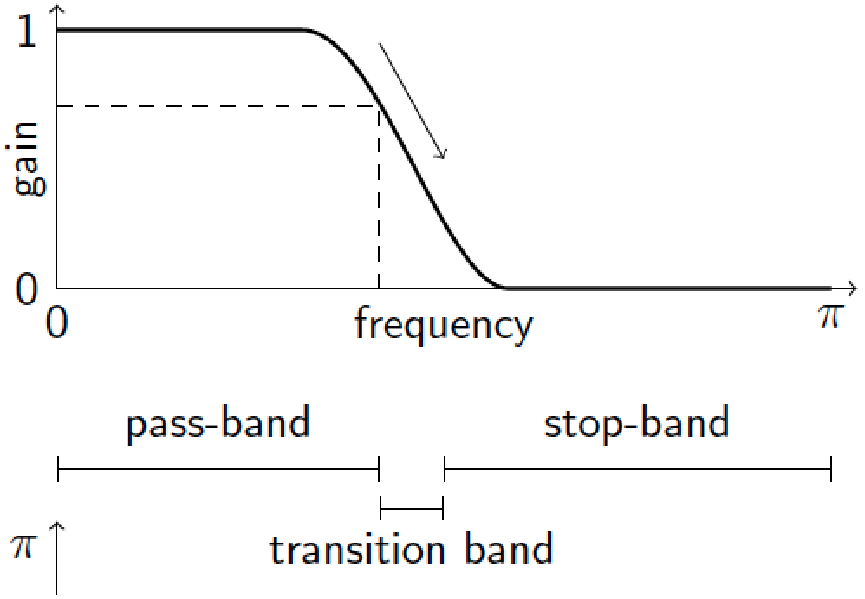




impulse response

A practical lowpass filter

- Gain near 1 for low freq, 1/sqrt(2) at cutoff freq, approaches 0 thereafter
- Gentler slope of frequency response in transition band:



Filter design tools allow you to make this sort of filter!

- You specify type of algorithm, filter and parameters
  - hi/low-pass: cutoff frequency, order (how many terms in filter computation function)
  - band pass/stop: centre frequency and "Q" (quality)
- You get back an impulse response / filter equation with a nice, smooth frequency response!

Filter design tools in python:

signal.firwin: Design a filter of a given type, cutoff frequency, transition width, & number of coefficients

signal.freqz: Plot the frequency response for a given impulse response / set of filter coefficients

FIR & IIR filters

Generic filter equation

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] + \ldots + b_Q * x[n-Q]$$

Q: How many non-zero elements in the impulse response (at most)?
A: Q + 1
This is a *finite impulse response (FIR) filter*

What if we make y[n] dependent on previous *output* values as well?

$$y[n] = b_0 * x[n] + b_1 * x[n-1]$$
$$+ b_2 * x[n-2] + \ldots + b_Q * x[n-Q]$$
$$+ a_1 * y[n-1] + a_2 * y[n-2]$$
$$+ \ldots + a_M * y[n-M]$$

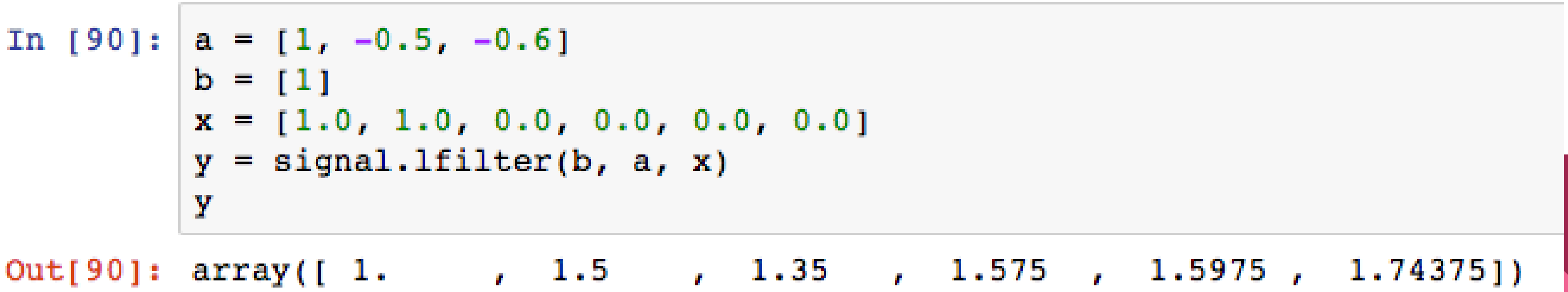
This is an *infinite impulse response (IIR)* filter, also called a feedback filter.

Exercise

y[n] = x[n] + 0.5 * y[n-1] + 0.6 * y[n-2]

What is the output for x = [1.0, 1.0] ?

Can use lfilter function in python:



```
In [90]: a = [1, -0.5, -0.6]
         b = [1]
         x = [1.0, 1.0, 0.0, 0.0, 0.0, 0.0]
         y = signal.lfilter(b, a, x)
         y

Out[90]: array([ 1.     ,  1.5    ,  1.35   ,  1.575  ,  1.5975 ,  1.74375])
```

Tradeoffs: IIR vs FIR

IIR:

- More computationally efficient: sharper frequency response with fewer coefficients
  - e.g., biquad filter: 3 feedforward & 2 feedback coefficients
- May do weird things to phase alignment
- Can “blow up” if you don’t design them right

FIR:

- More computationally expensive: need more coefficients (also introduces more delay)
- Can be “linear phase”same relative phase shift for all frequencies

Resources

Python: scipy.signal.lfilter, scipy.signal.iirfilter

Biquad filter calculator:

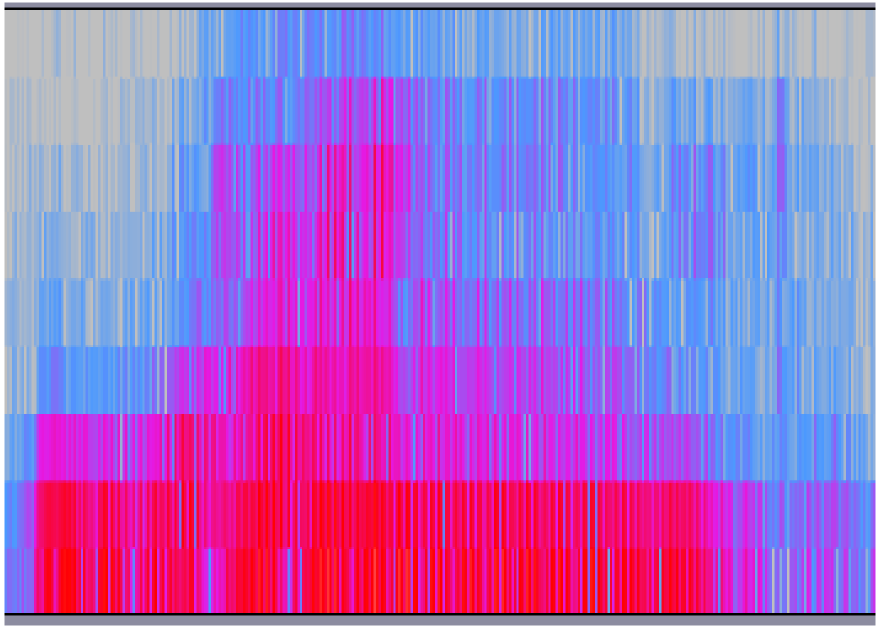
http://www.earlevel.com/main/2013/10/13/biquad-calculator-v2/

Longer discussion of IIR vs FIR:

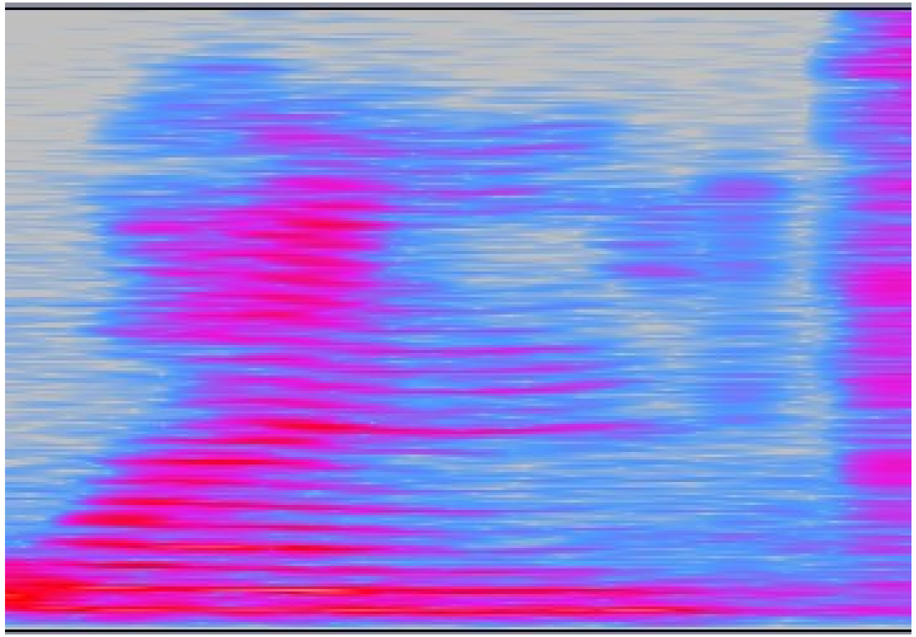
https://www.minidsp.com/applications/dsp-basics/fir-vs-iir-filtering

Revisiting the FFT

Time/Frequency tradeoff

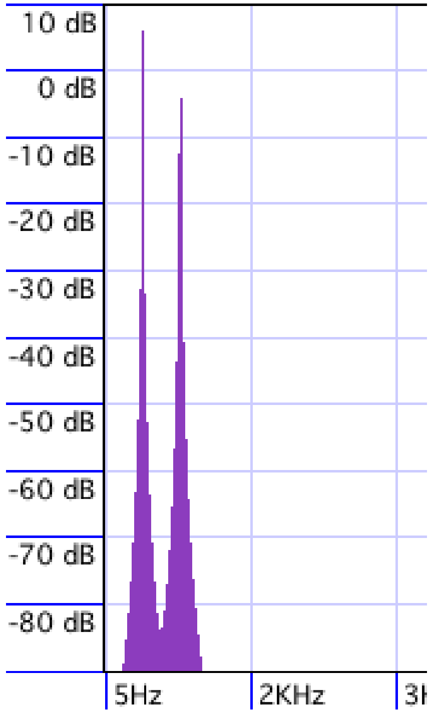


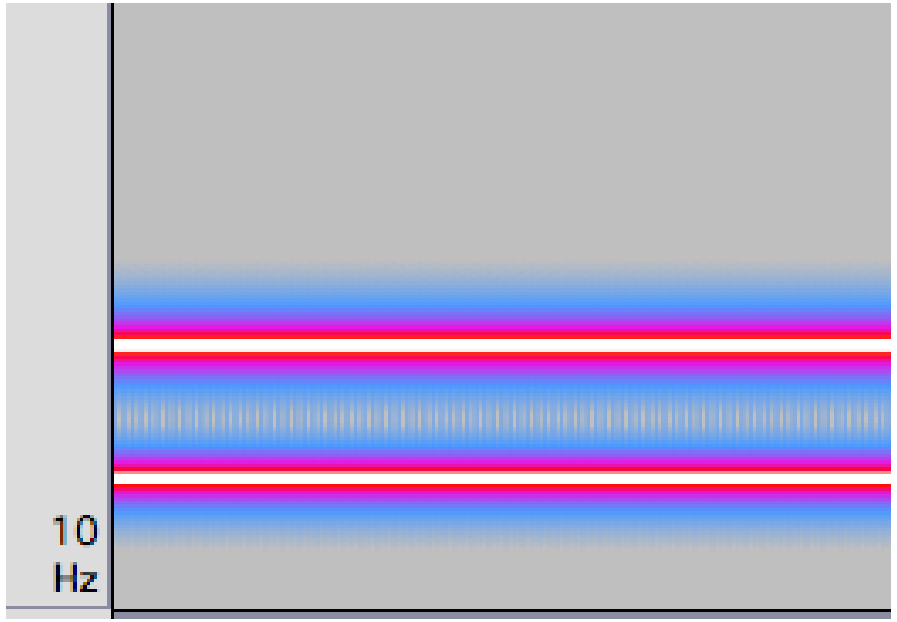
N=64



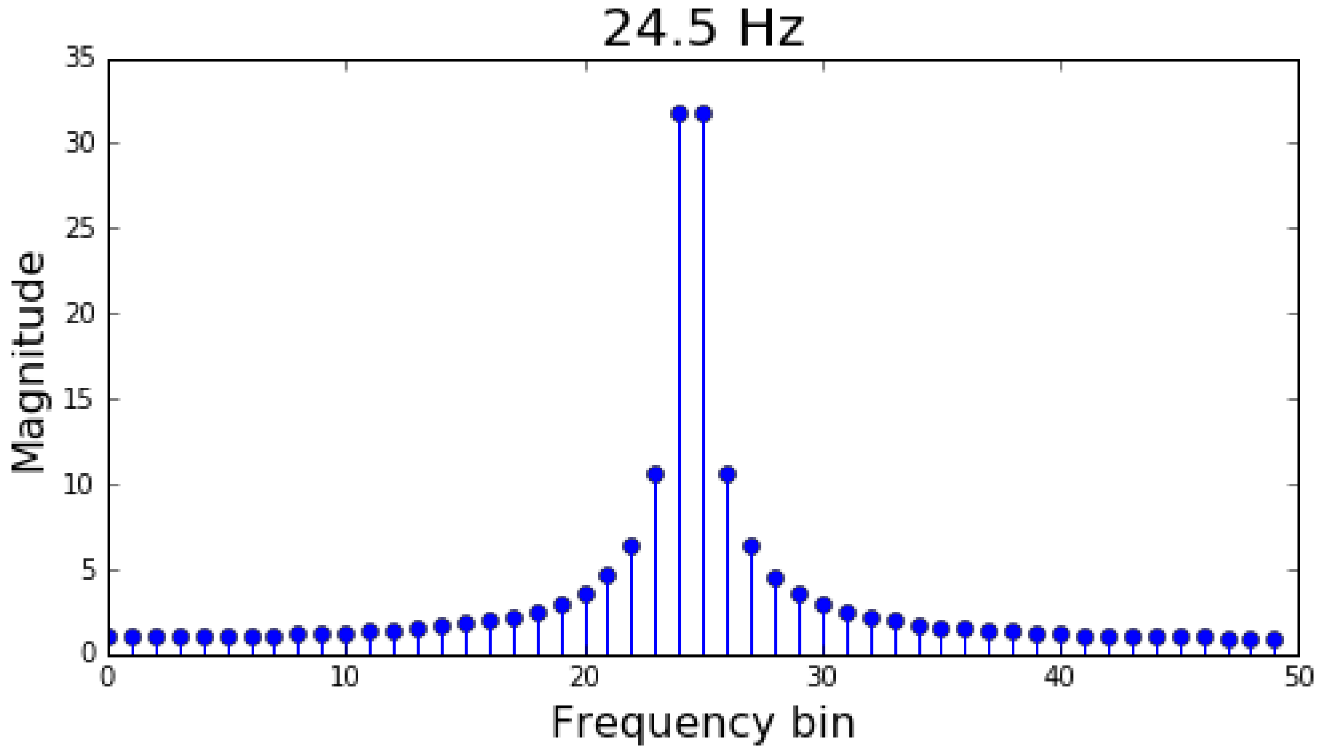
N=4096

What's all that extra stuff in the spectrum?

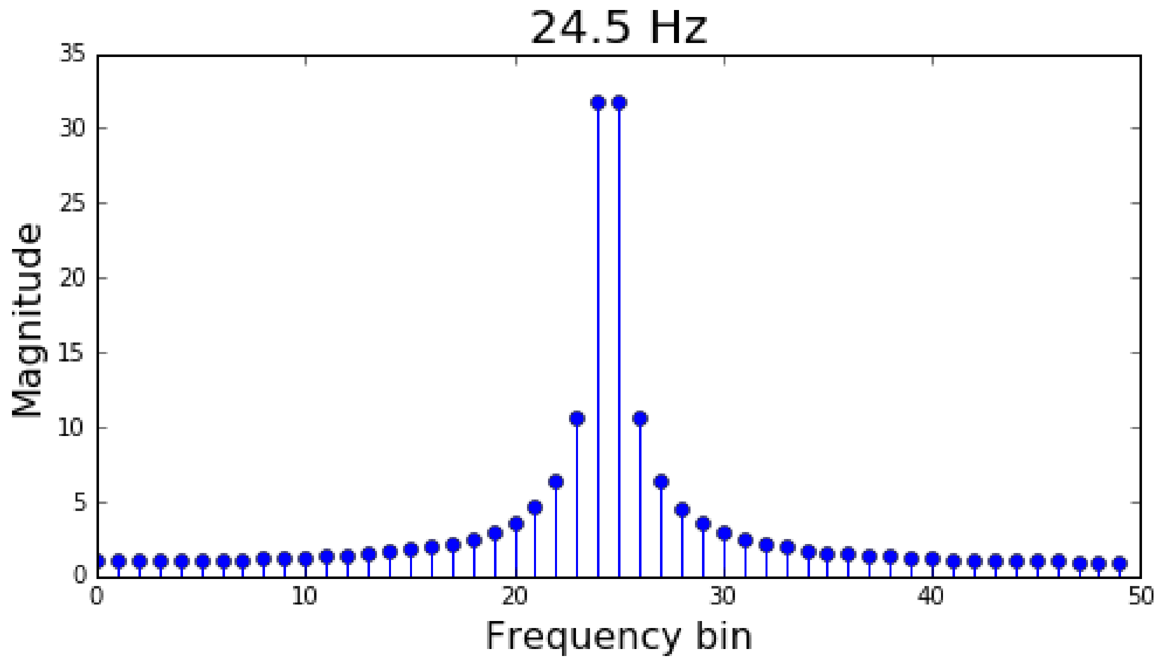




Not just clean peaks at frequencies and 0 elsewhere…

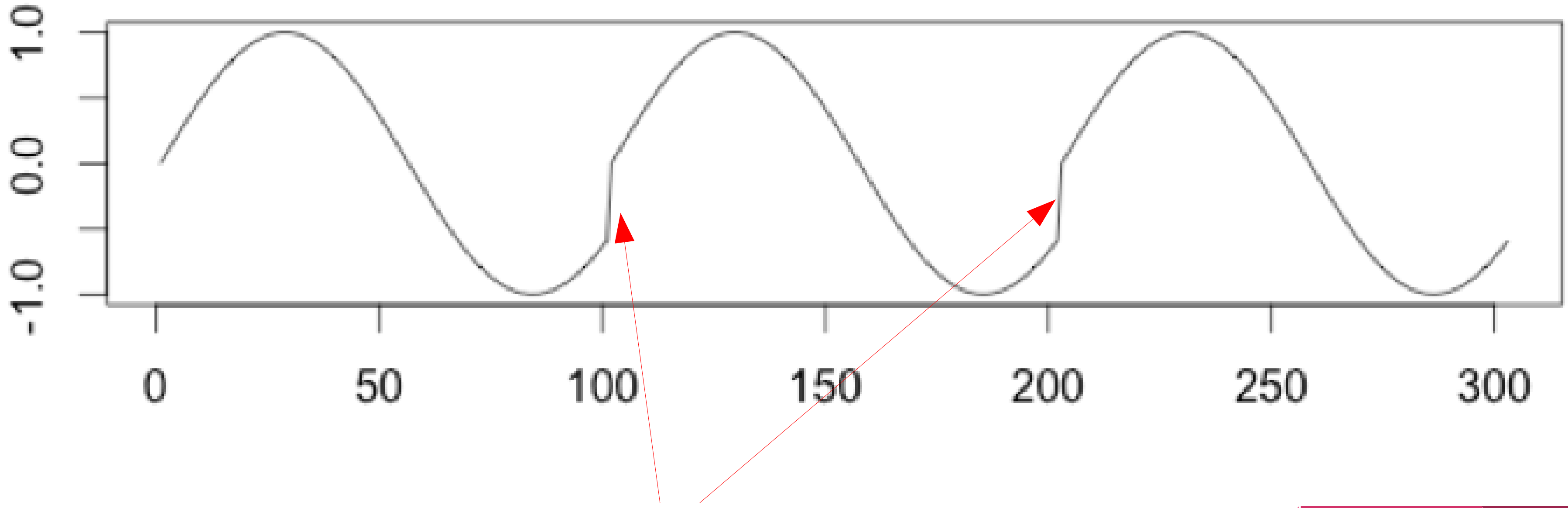


FFT answers the question, how can I combine sinusoids with the specific frequencies 0, (1/N)*SR, (2/N)*SR, ... SR/2 to re-create my signal?



FFT answers the question, how can I combine sinusoids with the specific frequencies 0, (1/N)*SR, (2/N)*SR, ... SR/2 to re-create ~~my signal~~ an infinitely repeating signal, where one repetition looks like my analysis frame?

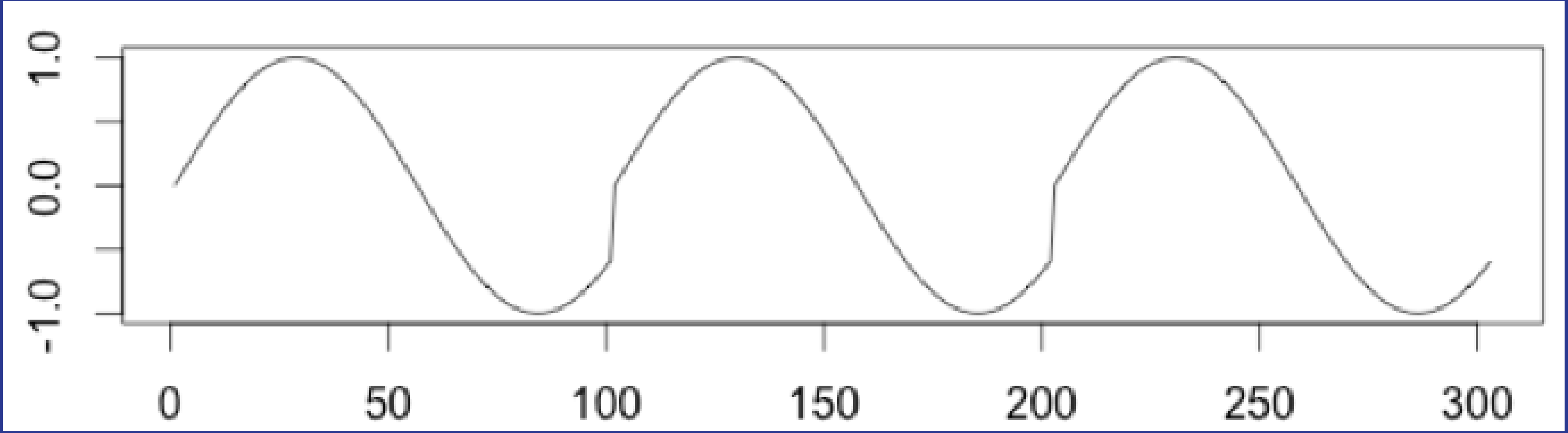
FFT treats your analysis frame as one period of an infinite, periodic signal.



“periodic” signal may have discontinuities

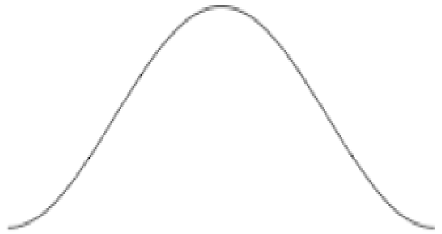
 only representable with high frequency content

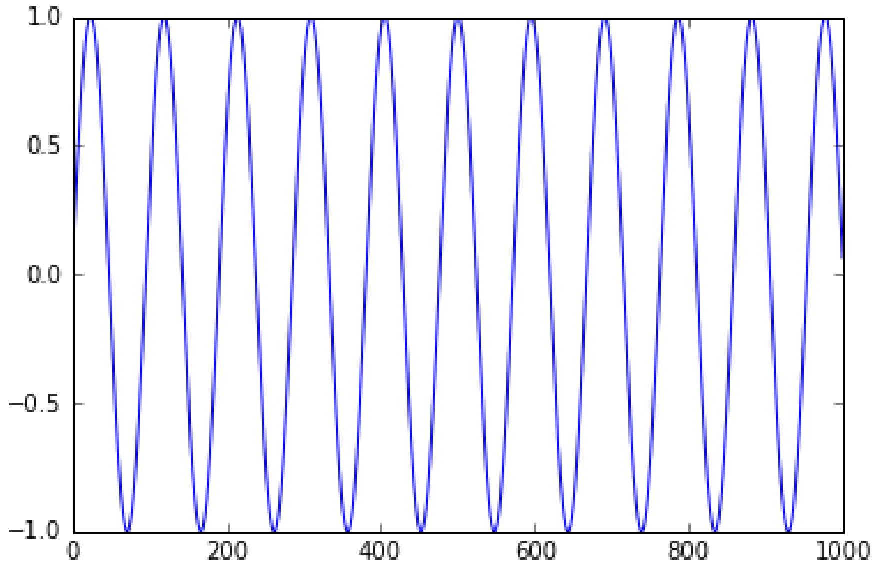
Windowing is a technique for getting rid of these discontinuities



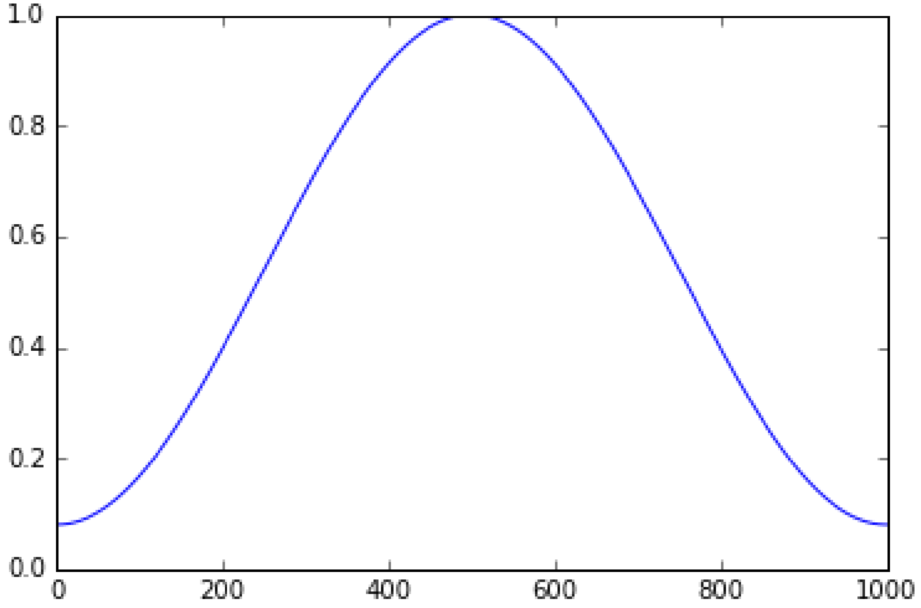
Windowing

Before taking FFT, multiply the signal with a smooth window that will get rid of sharp edges at either end of analysis frame



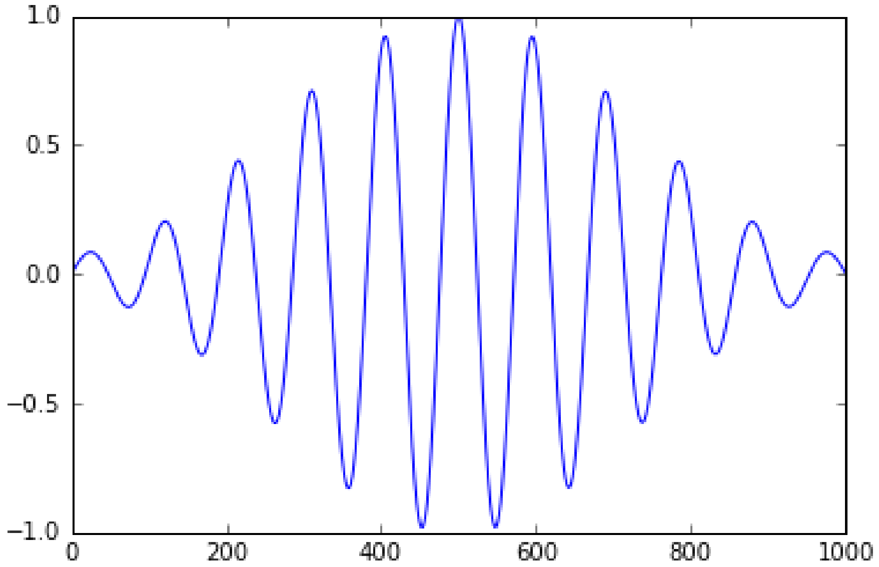


```
t = np.arange(0, 1, 1/1000)
s = sin(2*pi*10.5*t)
plot(s)
```



```
t = np.arange(0, 1, 1/1000)
s = sin(2*pi*10.5*t)
w = np.hamming(1000)
plot(w)
```

The windowed signal

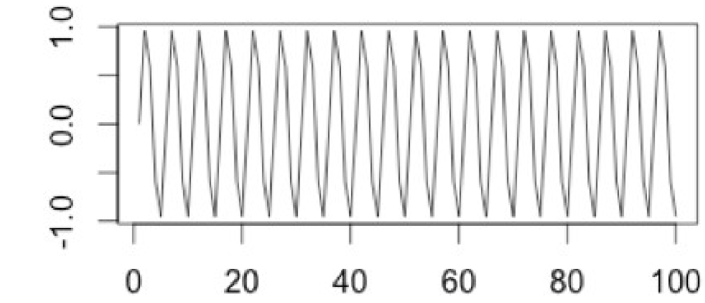


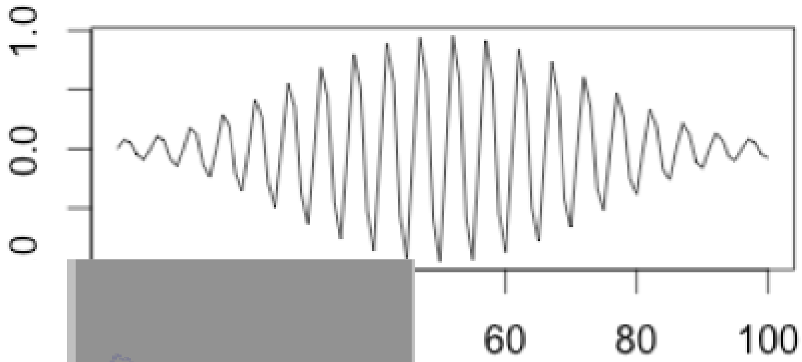
```
t = np.arange(0, 1, 1/1000)
s = sin(2*pi*10.5*t)
w = np.hamming(1000)
plot(w*s)
```

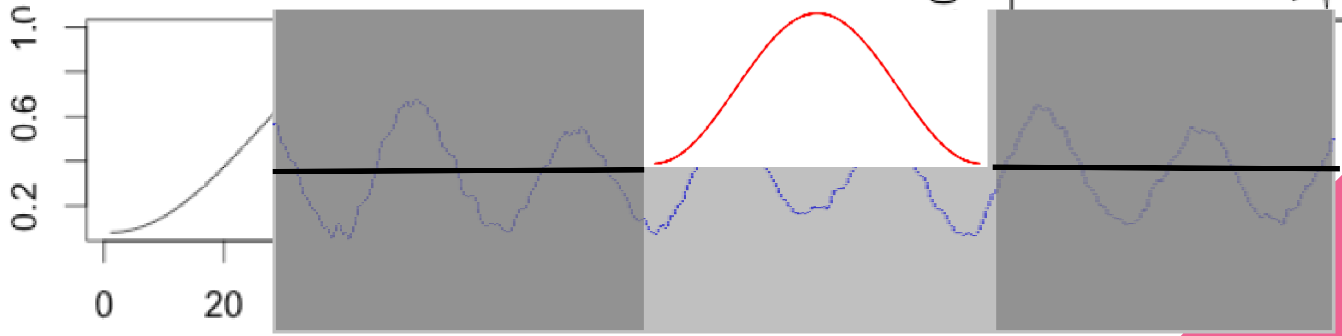
Windowing process

1. point-wise multiply signal with window:

2. Then apply FFT to the result

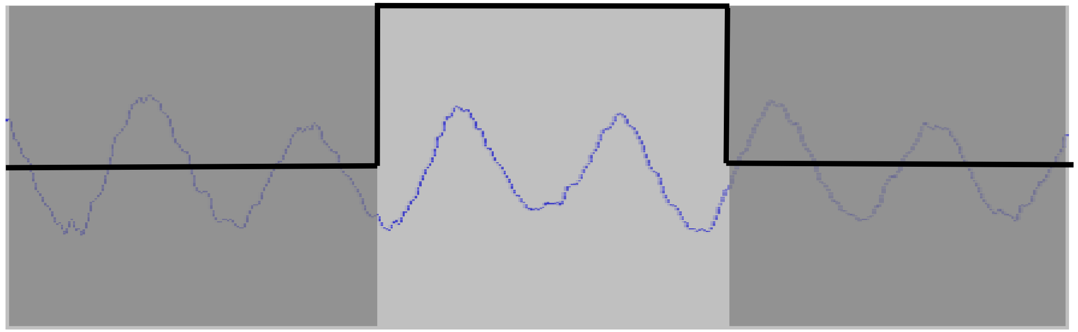






You’ve been windowing without knowing it...

“Selecting” N time-domain samples is like point-by-point multiplication with a rectangular function (“window”):



Recall:

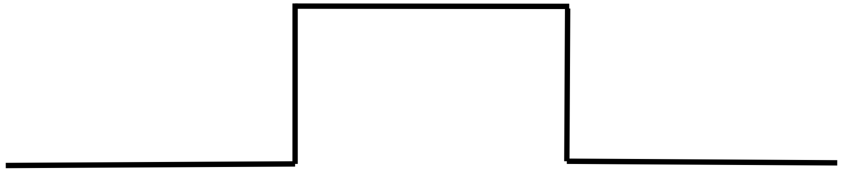
Convolution in the time domain is equivalent to multiplication in the frequency domain.

Also: Multiplication in the time domain is equivalent to convolution in the frequency domain.

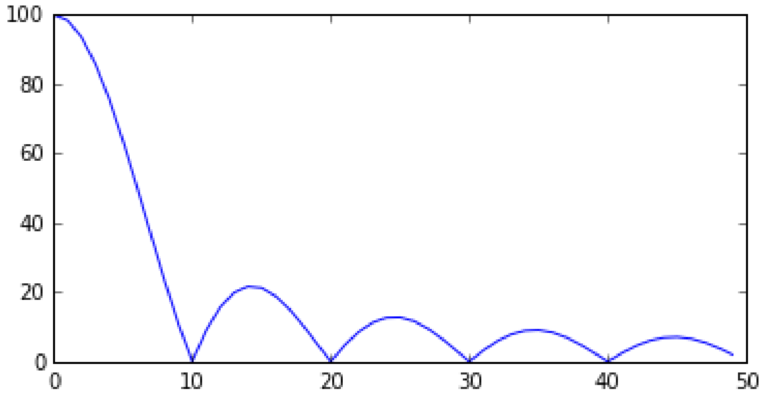
Windowing

A rectangular signal has a very “messy” spectrum!

Signal:

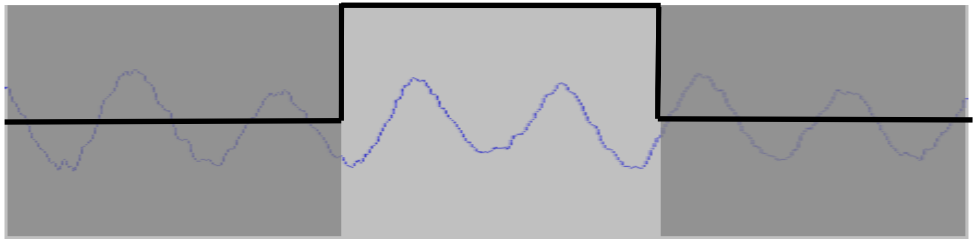


Spectrum:

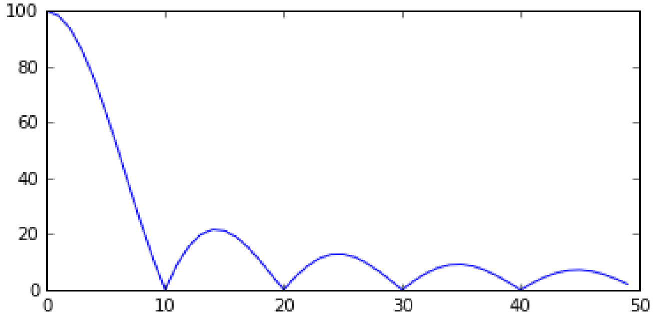


Windowing

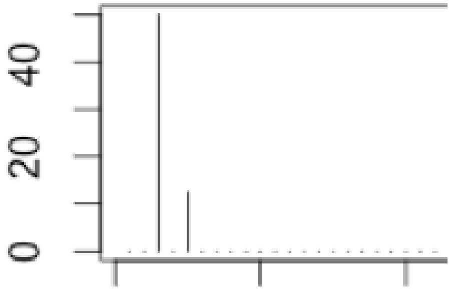
Multiplying a signal by a rectangle in time…



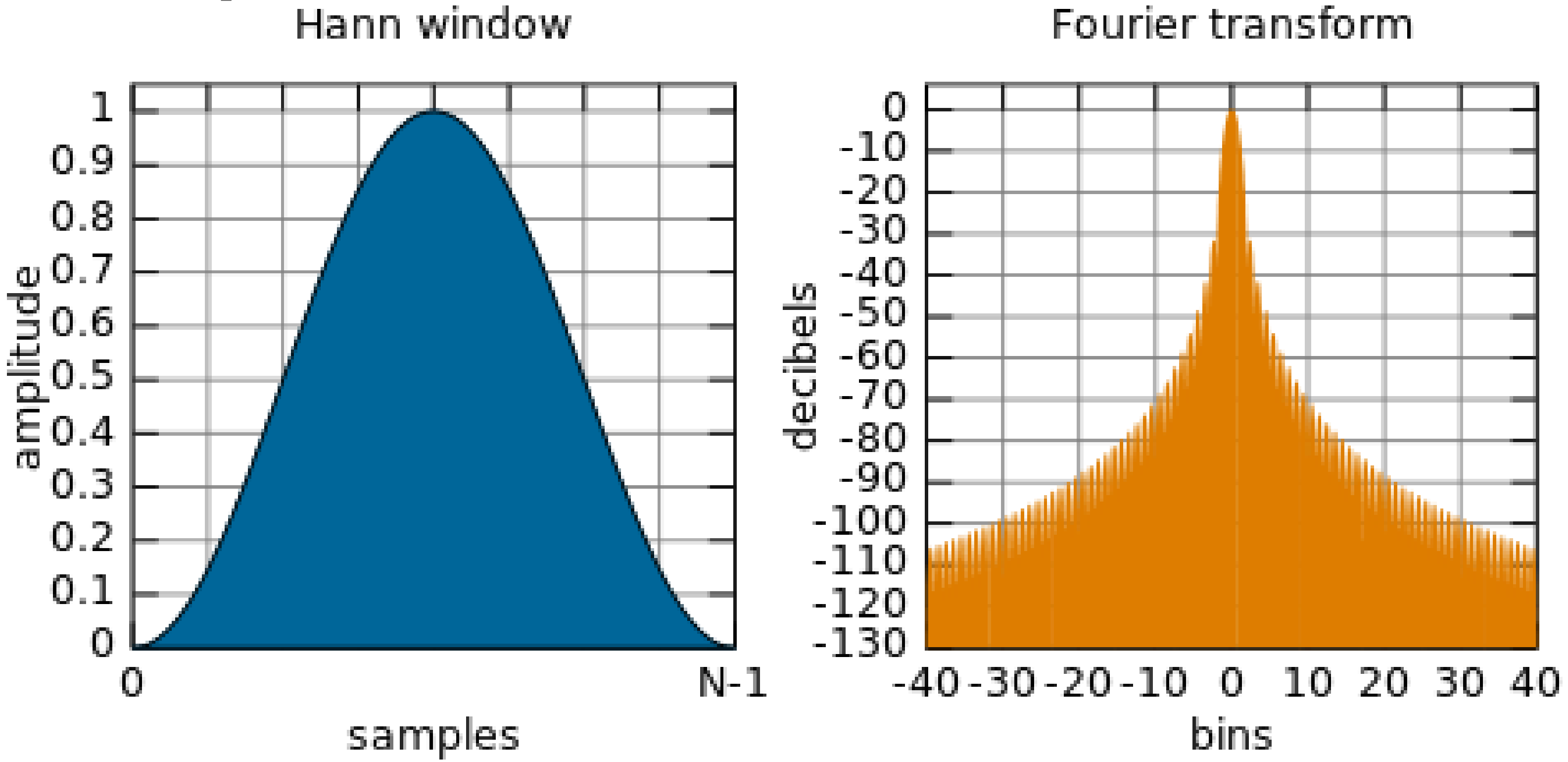
is equivalent to convolving their spectra:



$*$

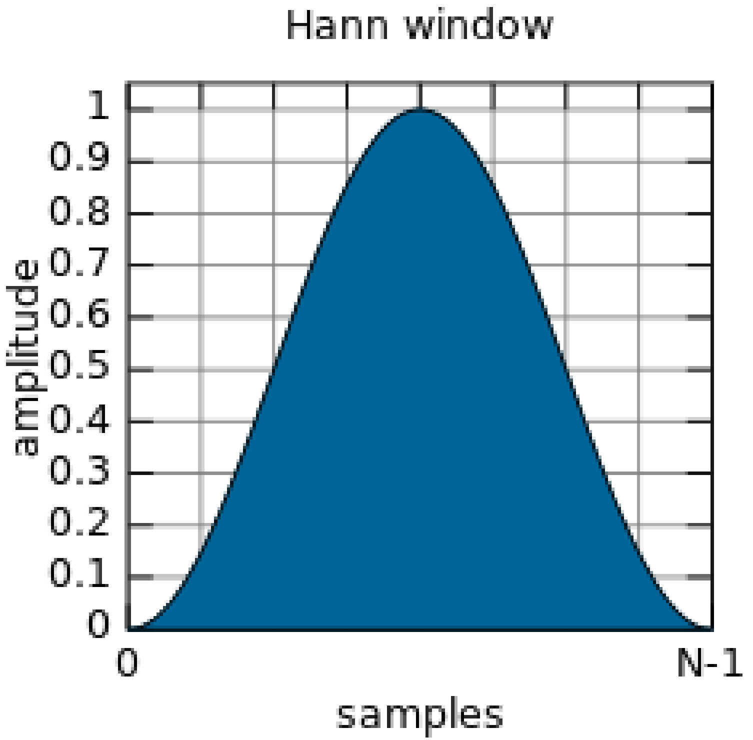


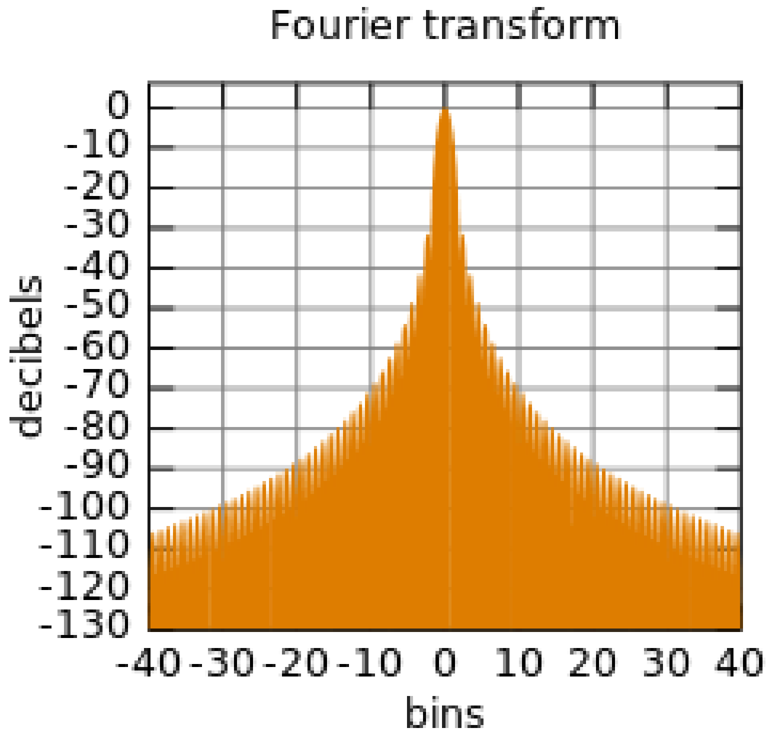
Example windows



From http://en.wikipedia.org/wiki/Window_function

Example windows



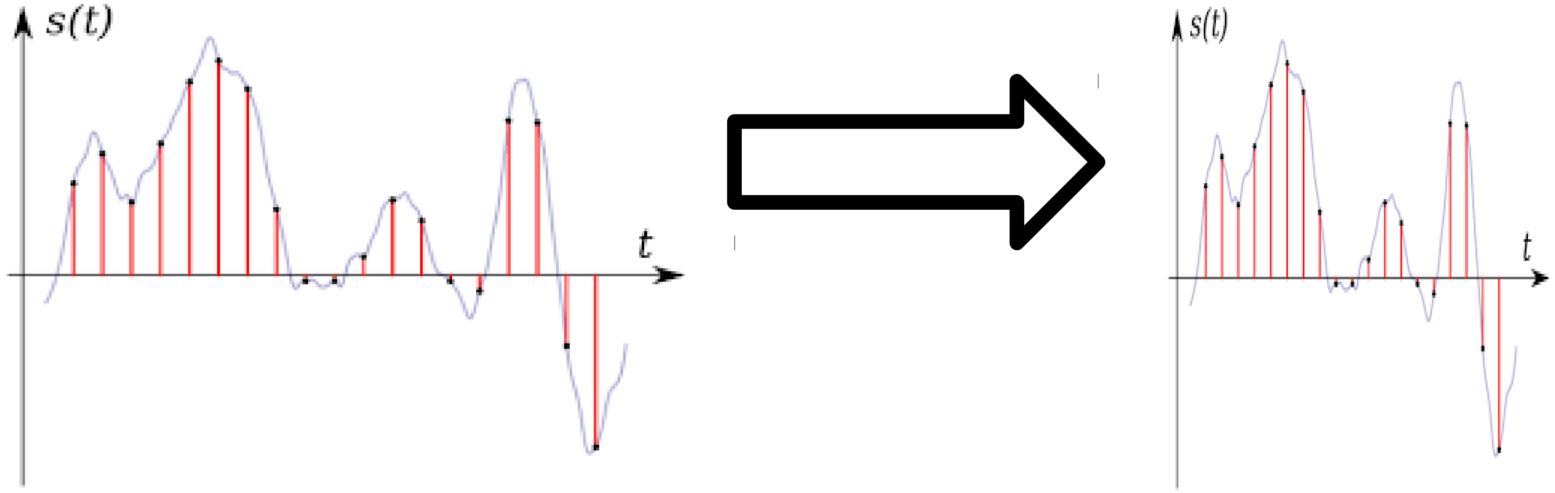


From http://en.wikipedia.org/wiki/Window_function

Cool audio effects that aren’t LTI systems

Pitch shifting & time-stretching

Technique 1: Resample

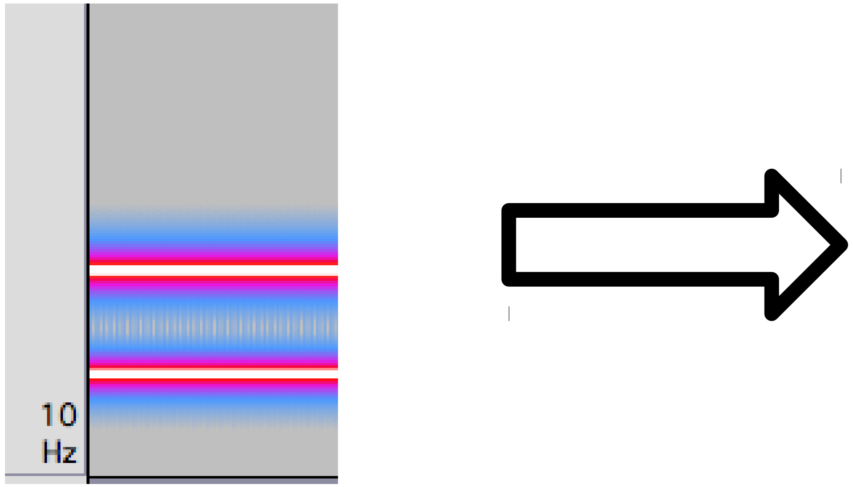


Problems?

```
file2 =  wavReadMono("~/audio/12345.wav")
play(file2, rate=80000)
```

Pitch shifting & time-stretching

Technique 2: Sinusoidal tracking

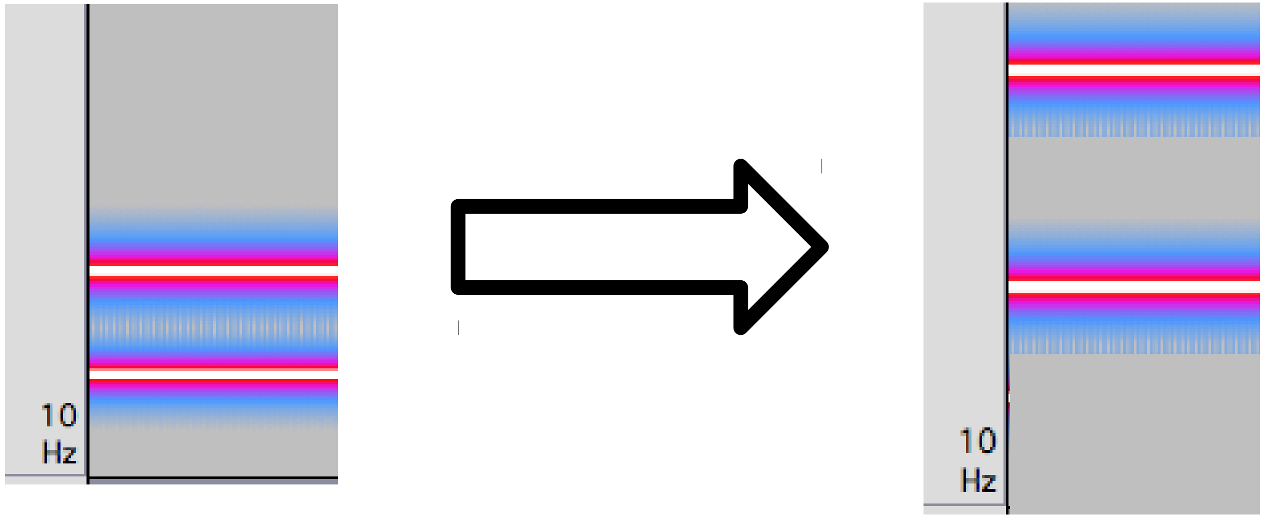


Use STFT to understand frequency content

Resynthesize a sound as sum of sine waves, control their amplitudes over time (e.g., at half the rate)

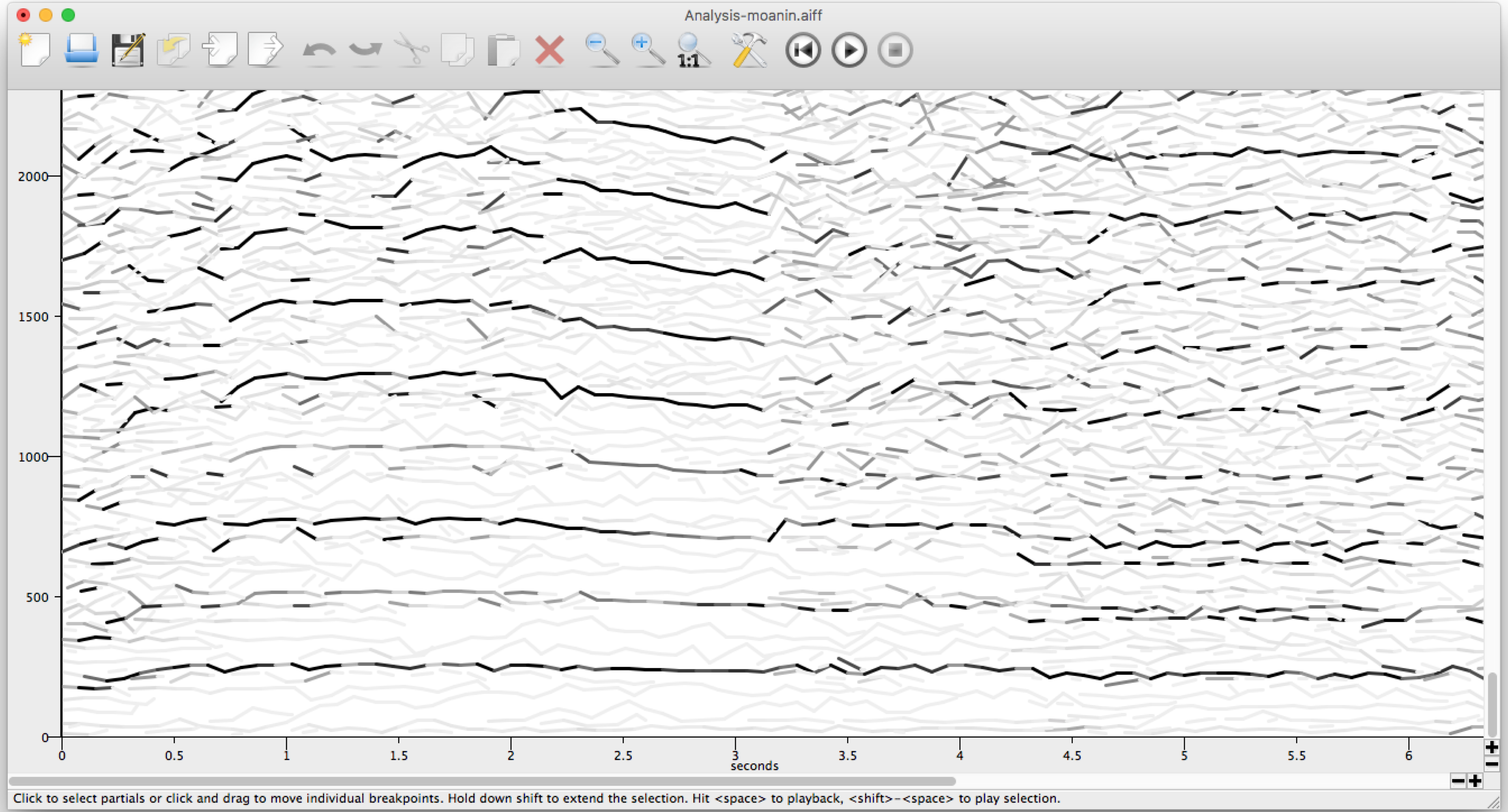
Pitch shifting & time-stretching

Technique 2: Sinusoidal tracking



Resynthesize a sound as sum of sine waves, change their frequencies (e.g., at twice the originals)

Problems?

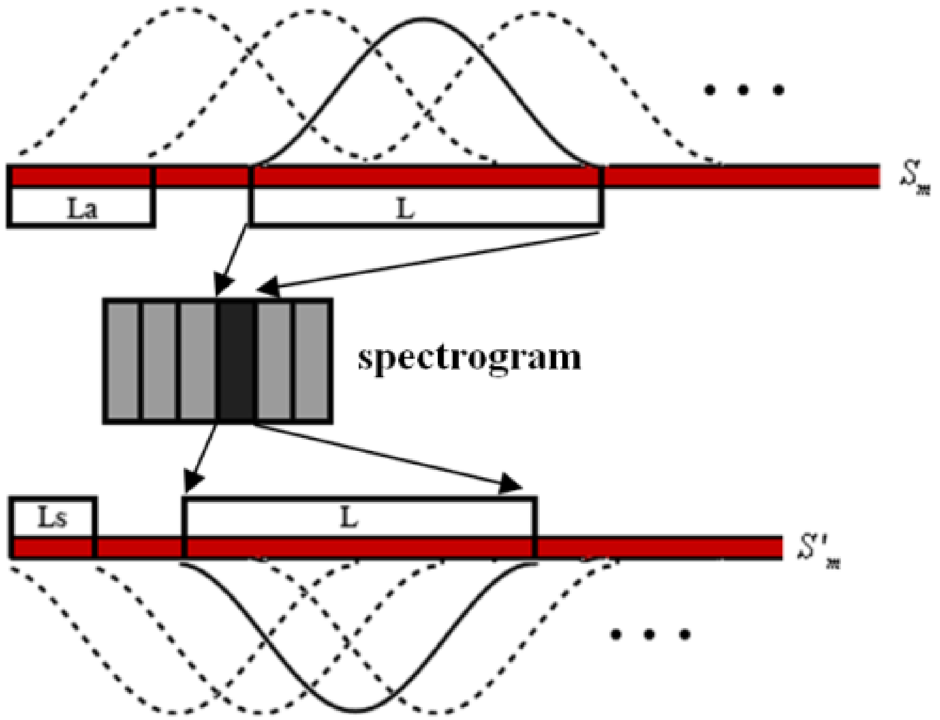


Click to select partials or click and drag to move individual breakpoints. Hold down shift to extend the selection. Hit <space> to playback, <shift>-<space> to play selection.

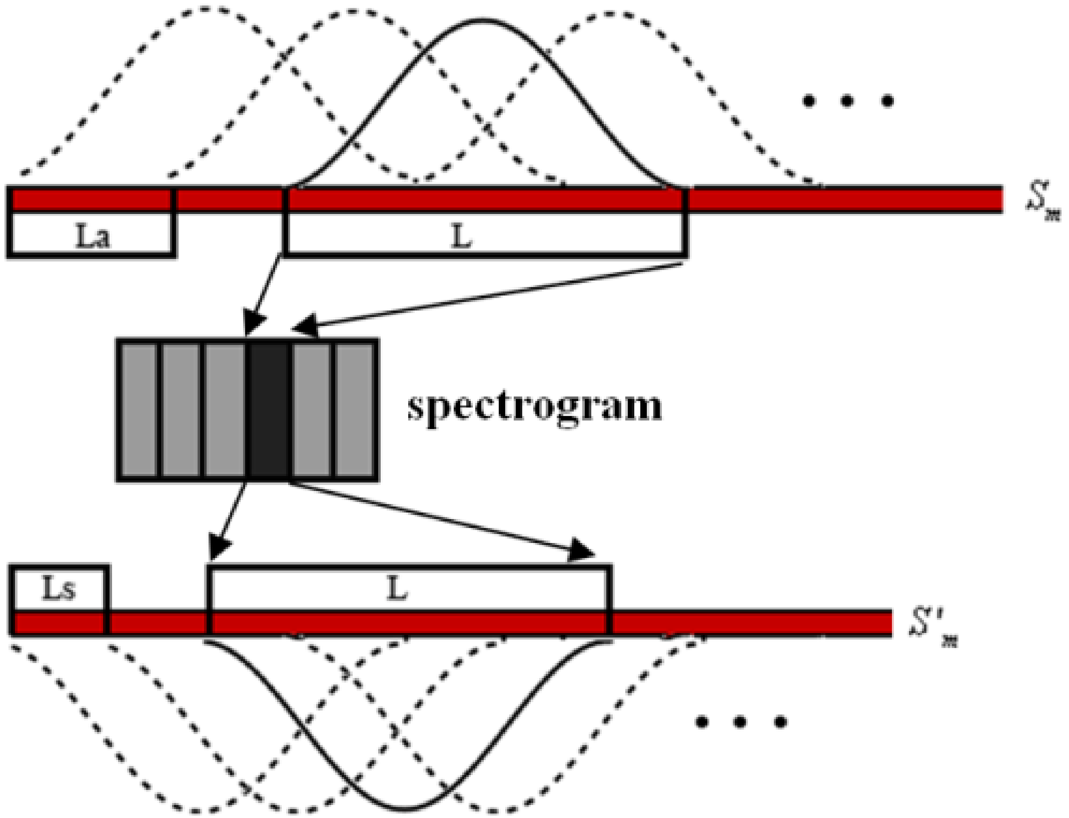
eg: http://www.klingbeil.com/spear/

Pitch shifting & time-stretching

Technique 3: "Overlap & add"



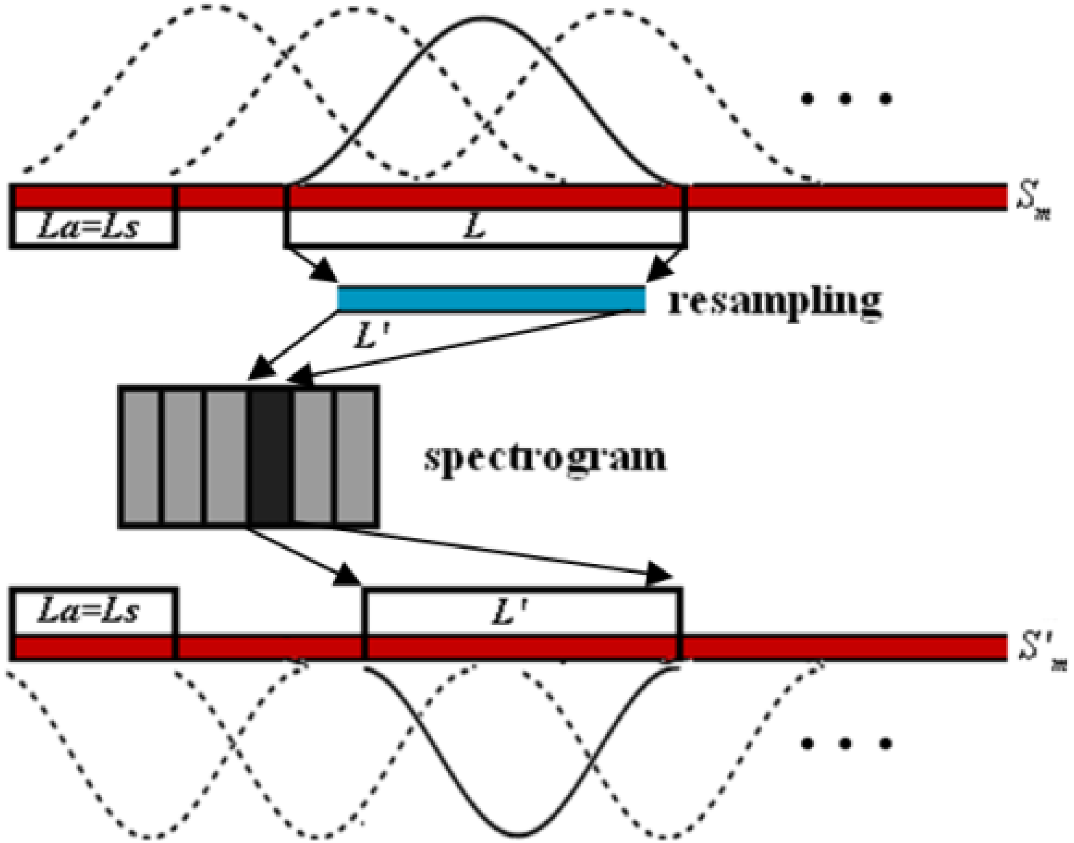
Time stretching w/ overlap-add



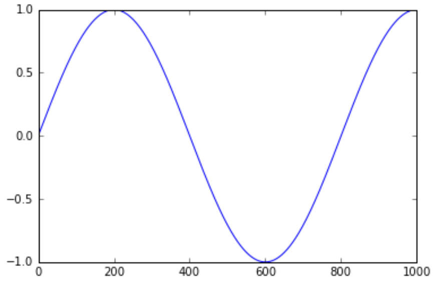
Use different hop size in input vs output

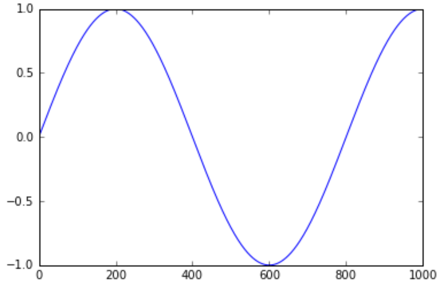
https://www.codeproject.com/Articles/245646/How-to-change-the-pitch-and-tempo-of-a-sound

Pitch shifting with overlap-add

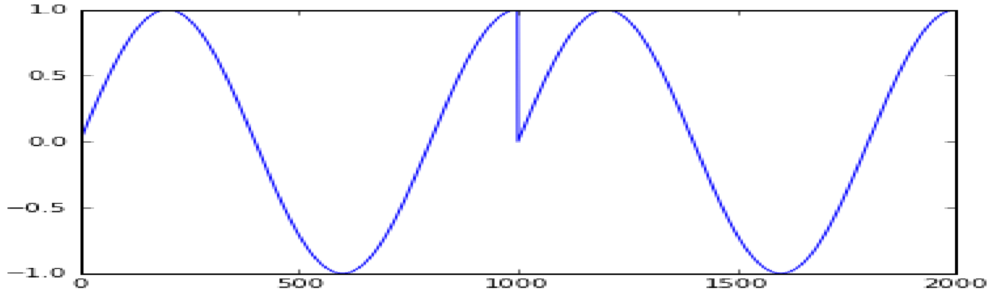


What happens when we add?

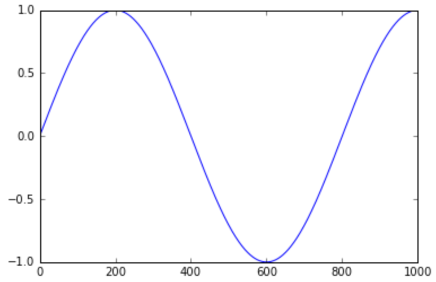


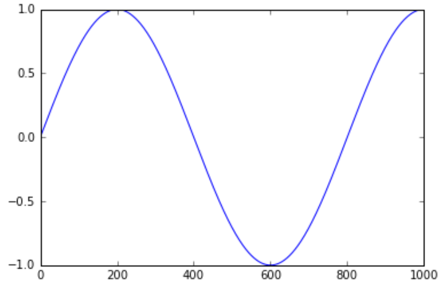


+

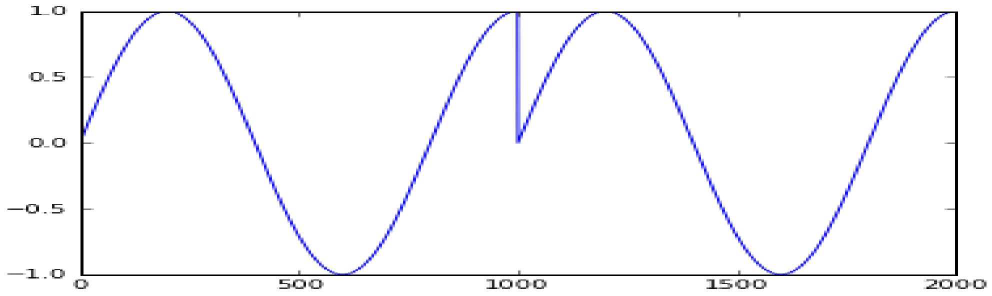


Better to keep track of phase...



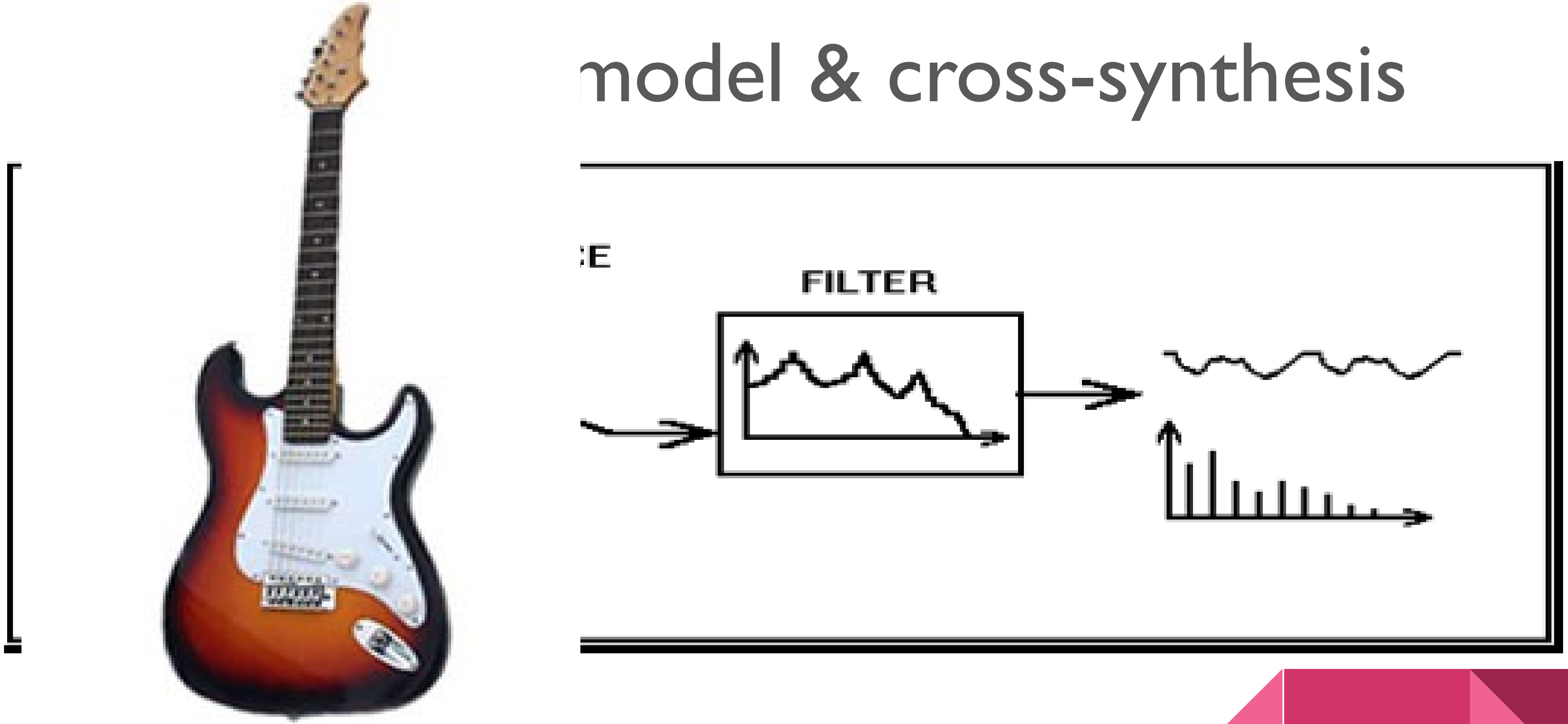


+



See:
Phase Vocoder

model & cross-synthesis



Demo:
guitar source spectrum shaped by voice spectrum

```
#cross-synthesis code
#first, load (or synthesize) sounds into file1 and file2

newSound = zeros(size(file1))
i = 0
j = 0;
width = 2048;
hop = 2048/16;
win = np.hamming(2048)

while (i+ width < size(file1)) :
    if (j + width >= size(file2)) :
        j = 0
    frame1 = win*file1[i:i+width]
    frame2 = win*file2[j:j+width]
    f1 = fft.fft(frame1)
    f2 = fft.fft(frame2)
    ms = abs(f2)
    frame3 = f1 * ms

    sig1 = fft.ifft(frame3).real
    newSound[i:i+width] = newSound[i:i+width] + sig1
    i = i + hop
    j = j + hop

newSound = 0.7*newSound/(max(abs(newSound)))
play(newSound)
plot(newSound)
```

To read more

Pitch shifting & time stretching:
http://blogs.zynaptiq.com/bernsee/time-pitch-overview/

Description of signal processing in the phase vocoder:
https://www.eumus.edu.uy/eme/ensenanza/electivas/dsp/presentaciones/PhaseVocoderTutorial.pdf

Computer “Listening”

“What does this music sound like?”

What note is playing?

What chord is playing?

When do notes happen?

What key is it in?

What tempo is it?

What genre is it?

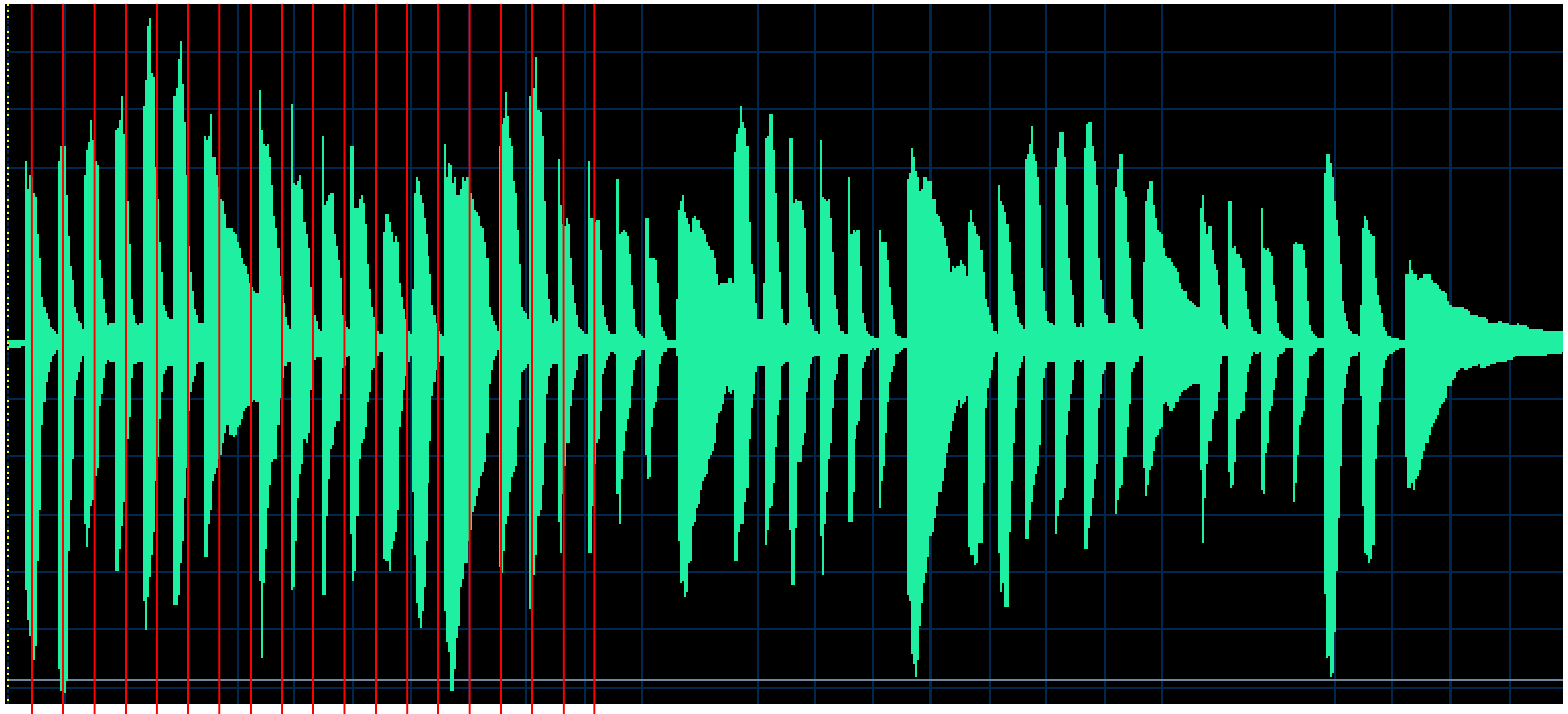
Would I like it?

Computing audio features

Feature: compute a value (or vector of values) from a sound

Sometimes a feature is useful enough on its own

Sometimes a "feature" tells us everything we need to know...



Computing audio features

Feature: compute a value (or vector of values) from a sound

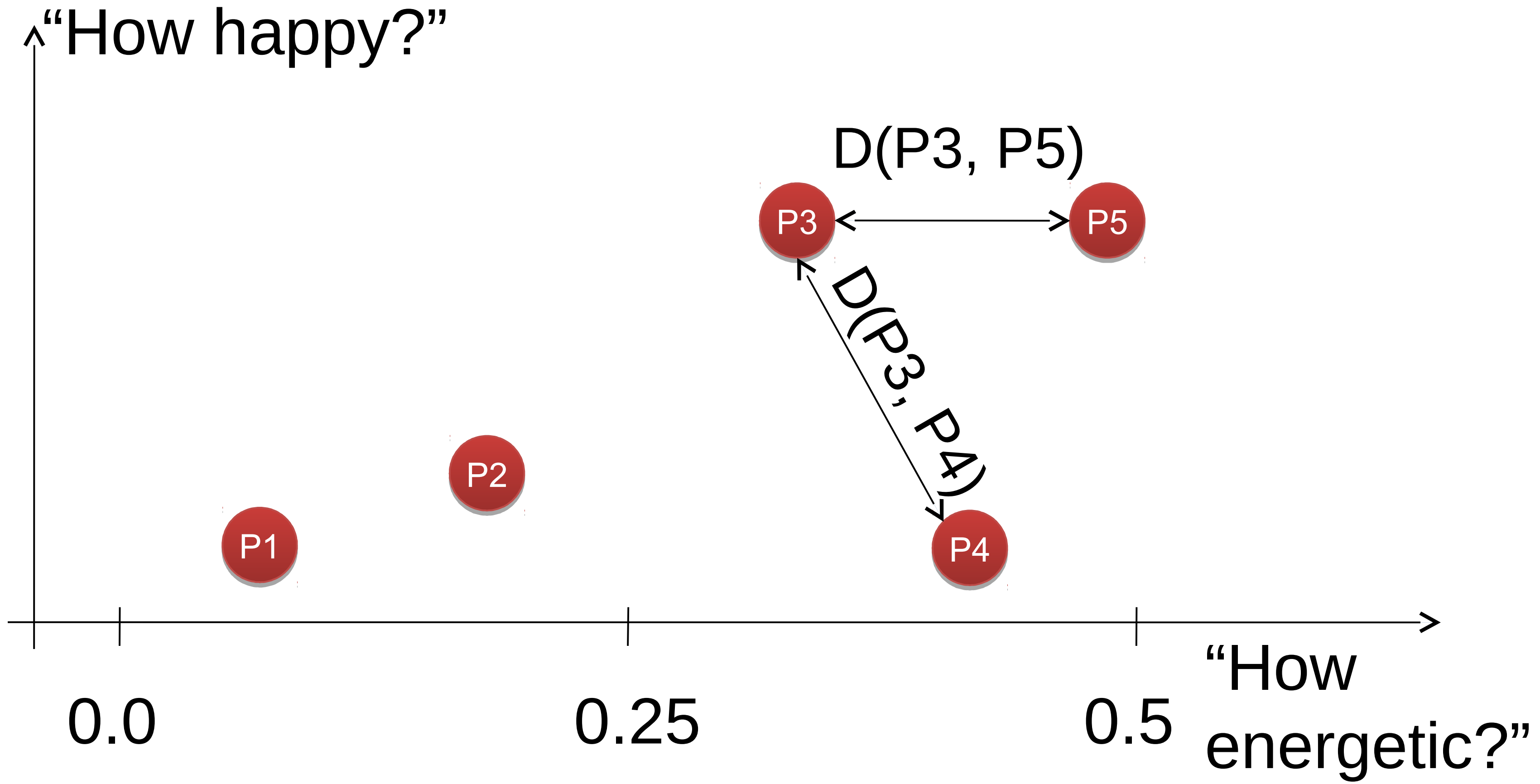
Sometimes a feature is useful enough on its own

Or you can compare it with other features:

- To some "ideal" value (e.g., "how much is this like a C major chord?")
- To some other example sound
  - Measure *similarity* between sounds: For labeling, recommendation, search, ...

Or you can use machine learning...

Euclidean distance: "As the crow flies"



Features & similarity

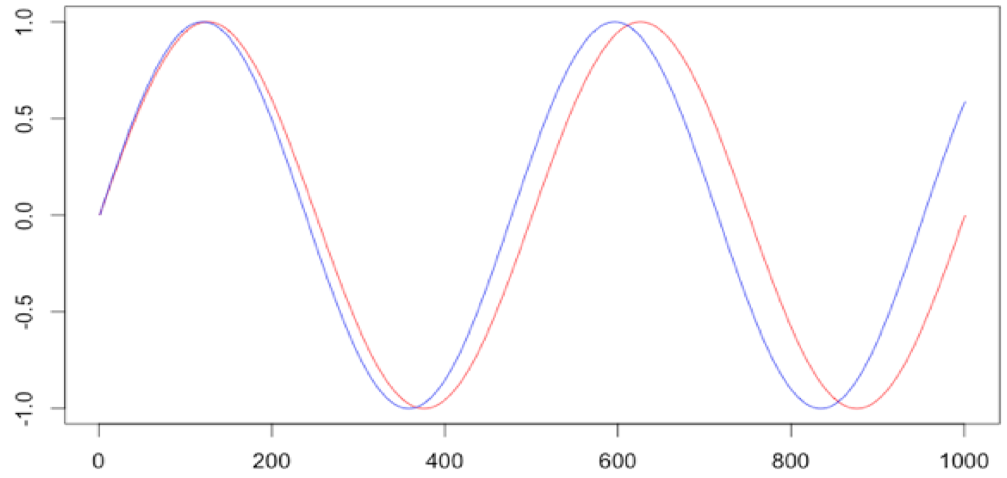
In general:

If two sounds are perceptually similar, we'd like them to have similar values for features

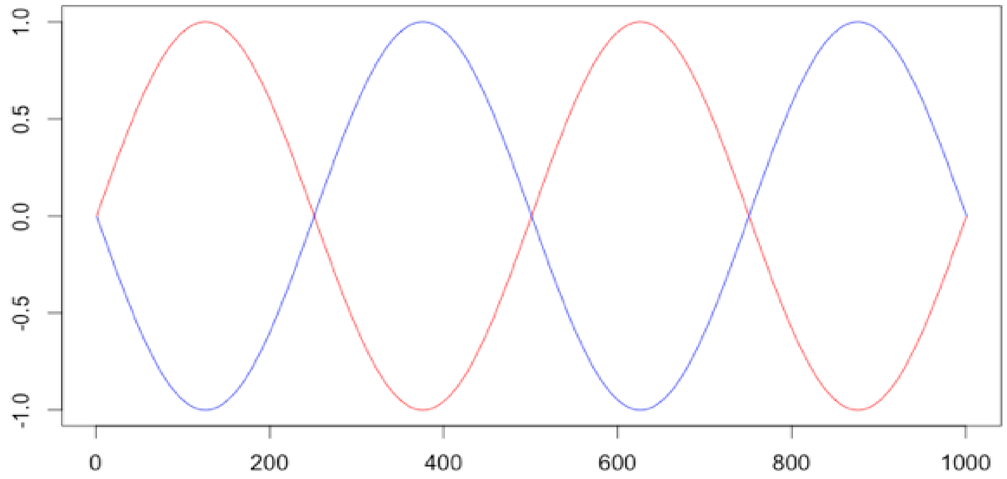
E.g., Sound 1 and Sound 2 have almost the same pitch: Let's find a feature /features for describing pitch which give them similar values.

Commonly Used Perceptual Features for Music and Sound

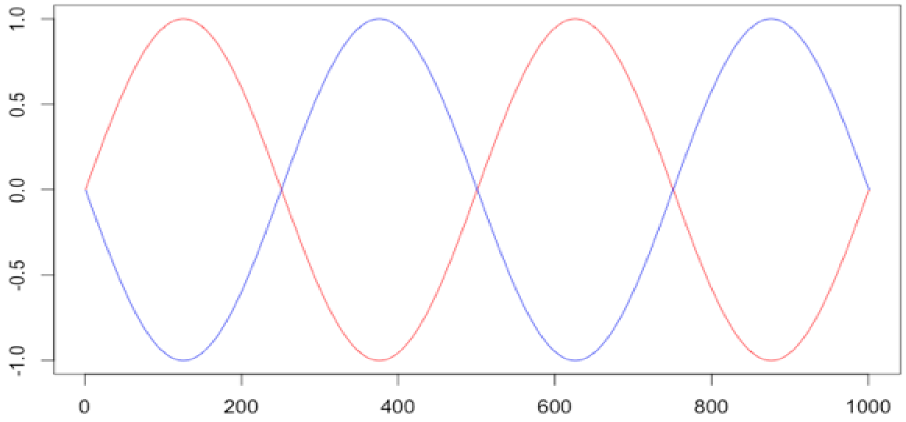
Simple audio features



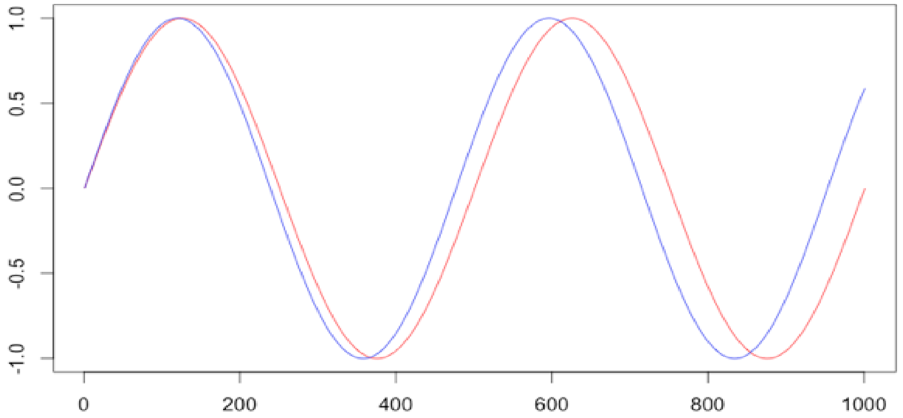
How similar will these sound?



How similar will these sound?



Sound exactly alike!
Euclidean distance
= 44.72



Sound different!
Euclidean distance
= 8.34

Computing distance in Python: `dist(s1, s2)`

☹

Big problem: Two sounds can sound *exactly* alike but have *very* different waveforms!

You should NOT use samples of the waveform as your features!

Another possibility:

- If waveform is *N* samples, then take its FFT, and use the *i*th FFT bin as the *i*th feature

- Often better than using samples! However, still problems:
  - hard to reason about how changes in pitch, timbre, volume, instrumentation, etc. will produce relative changes in distance
  - Overly sensitive to small changes in high frequency content

- There is probably a better feature representation that more precisely captures similarity for a given application…

Review: Loudness

- Loudness: related to *perceived* energy or power in audio
- Typically expressed in dB

$$10 \log_{10}\left(\frac{P_1}{P_0}\right) \text{ dB}$$

where P_0 is power @ threshold of hearing (0 dB).

- Increase of 10(?) dB roughly corresponds to doubling of loudness

http://www.indiana.edu/~emusic/acoustics/amplitude.htm

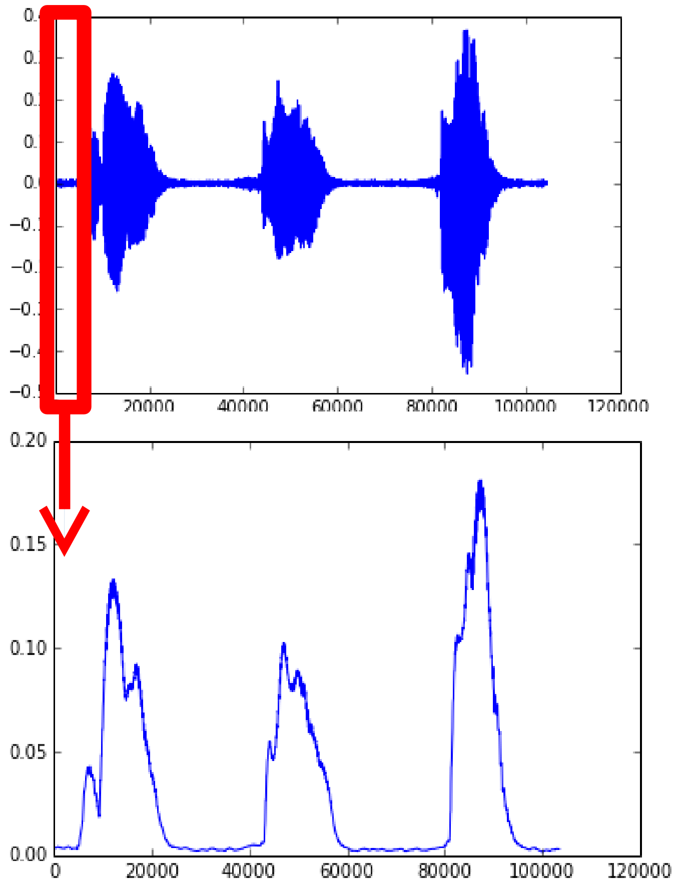
Simple features for loudness

- Do *not* use direct measures of amplitude as a feature ("peak amplitude" or "average amplitude")
    - our perception of loudness relates logarithmically to amplitude, so differences like A_1-A_2 are not meaningful
- Alternative: Use power spectrum (squared magnitude) expressed in dB (i.e., take the logarithm)
- Another common alternative: RMS ("root-mean-square") of audio waveform:

$$x_{\text{rms}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}$$

RMS (root-mean-square)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$



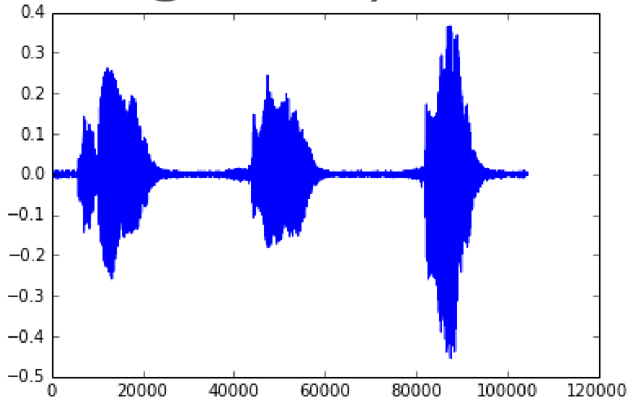
RMS of 1000-sample window, hop size 100 samples:

Reasonable for volume

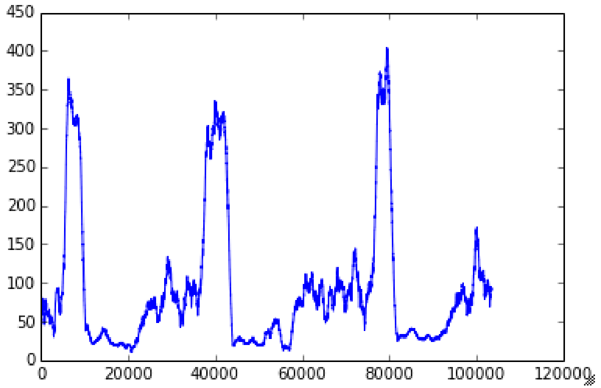
Features demo: Using sndpeek

ZCR (zero-crossing rate)

How many times does the signal cross zero?



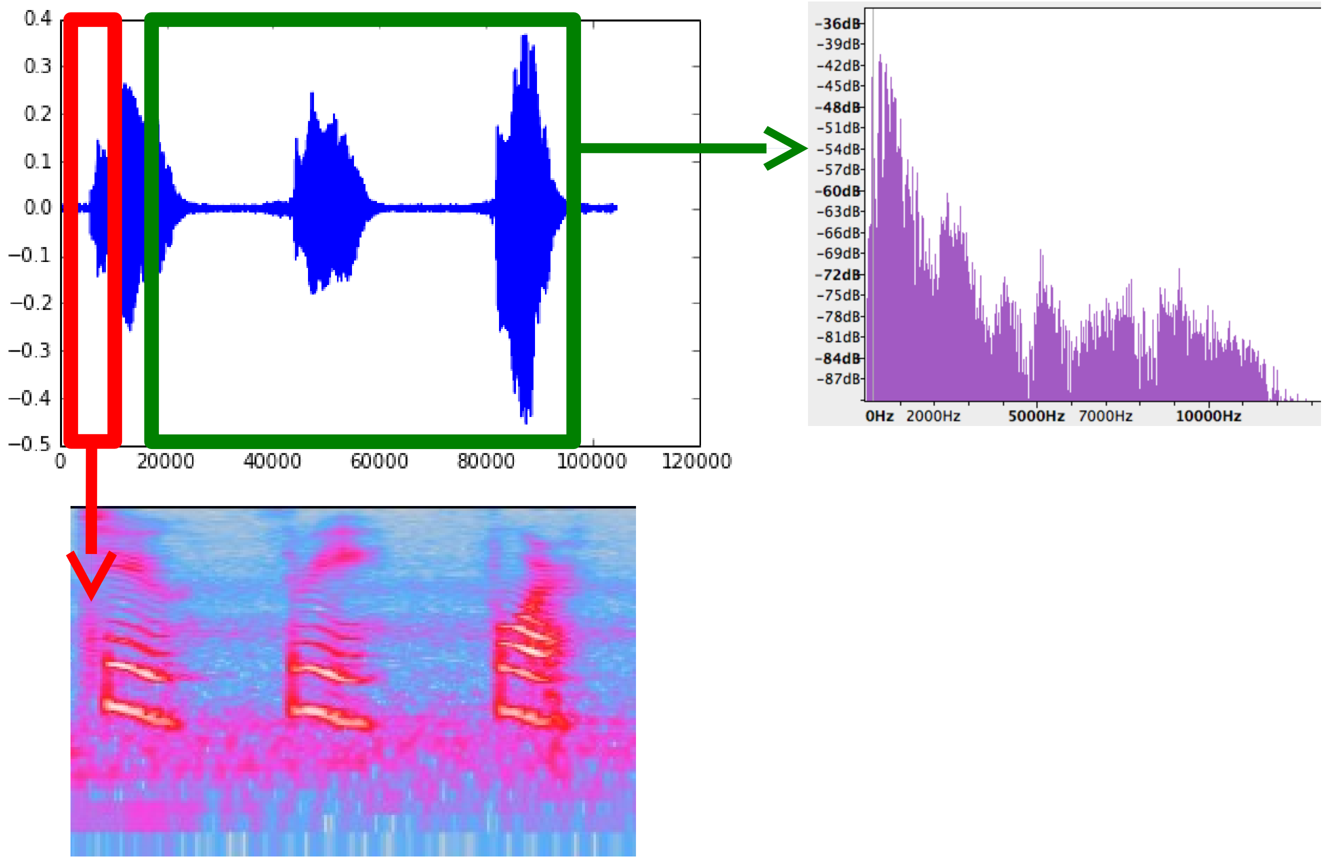
ZCR of 1000-sample window, hop size 100 samples:



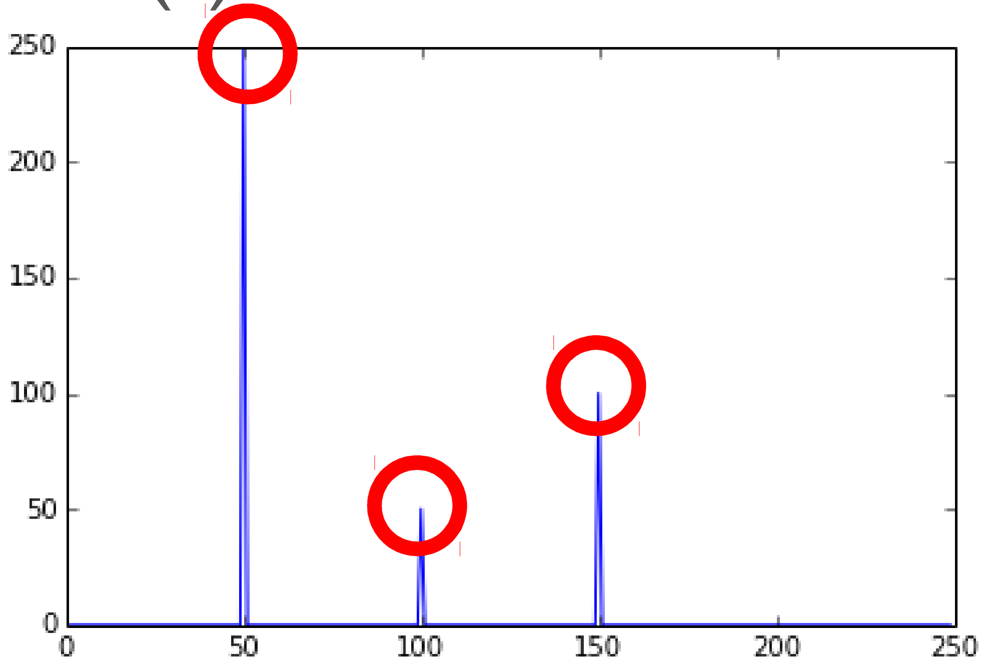
Reasonable for noisiness, silence, ~pitch

Spectral features

From the FFT or STFT of a signal:



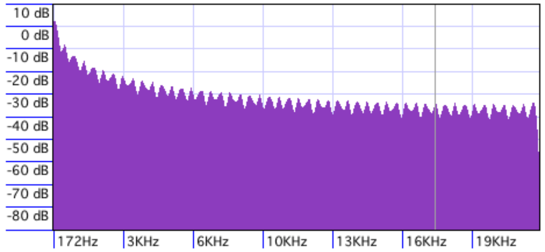
Spectral peak(s)



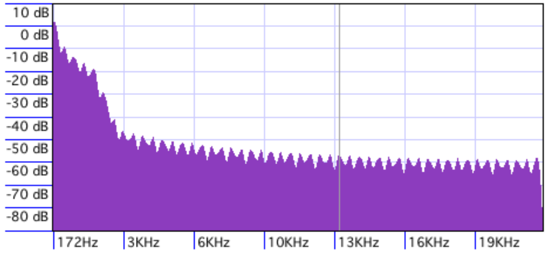
What are most prominent frequencies?
What is fundamental frequency?

Review: Timbre

- Timbre of a sound: related to several factors, one of which pertains to amount of high frequency content
  - More high frequency = "brighter", less = "darker"



"brighter"

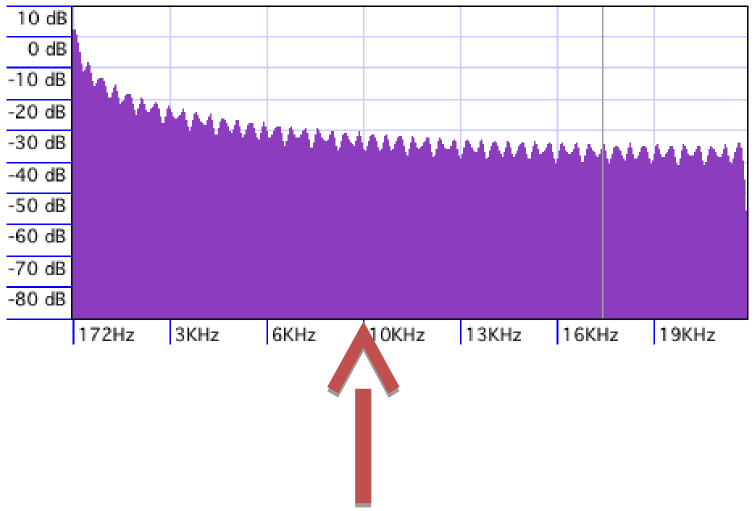


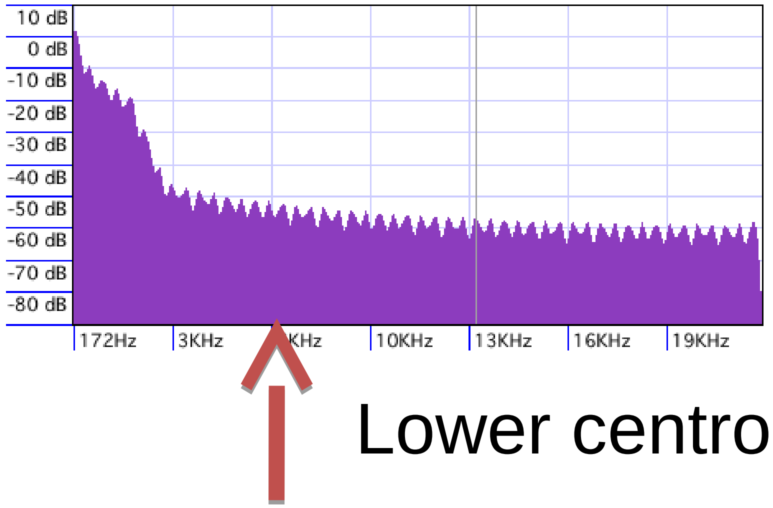
"darker"

A simple timbre feature

Spectral centroid: "center of mass" of the spectrum

$$\mathrm{SC}(m) = \frac{\sum_k f_k |X(m,k)|}{\sum_k |X(m,k)|}$$

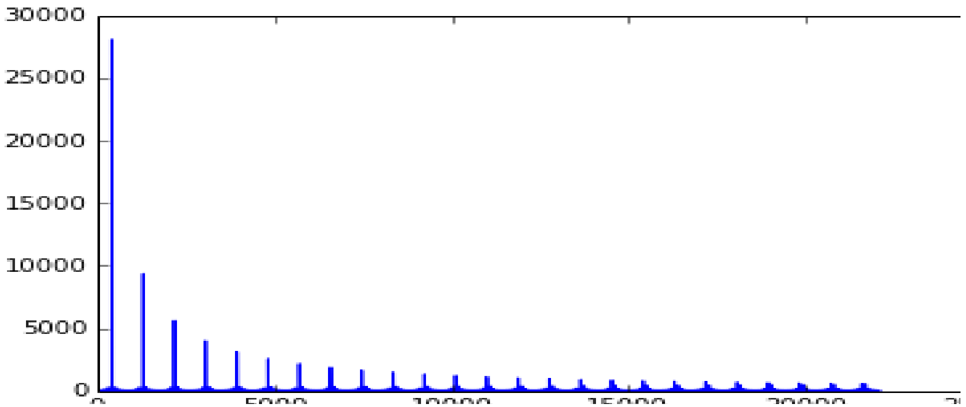


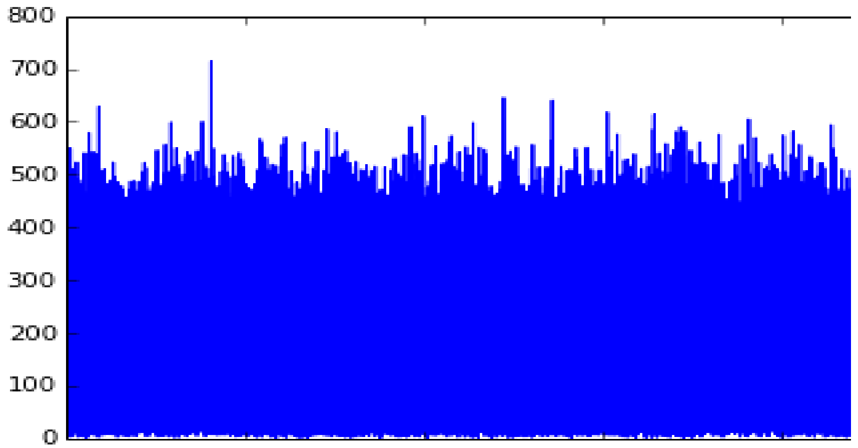


Lower centroid

Spectral flatness:

How “noise-like” vs. “tone-like”?





Tone-like

Noisy

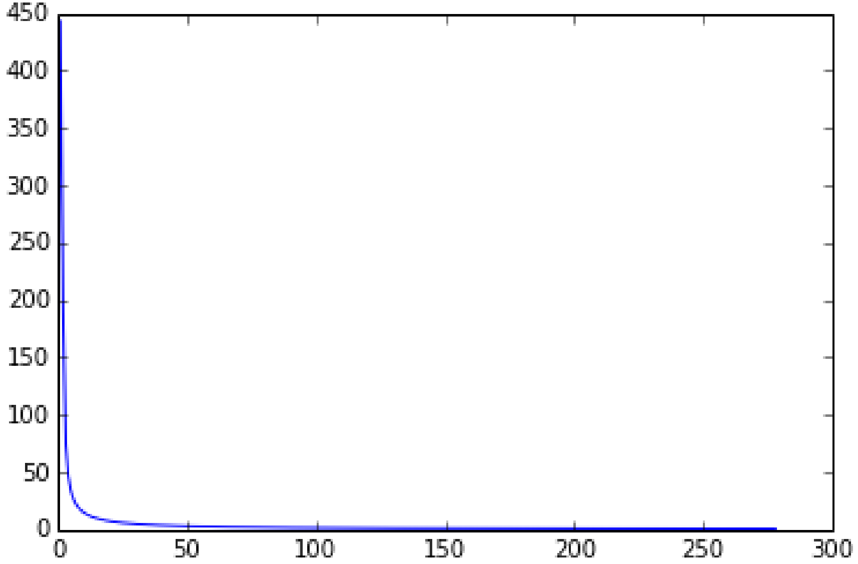
$$\text{Flatness} = \frac{\sqrt[N]{\prod_{n=0}^{N-1} x(n)}}{\frac{\sum_{n=0}^{N-1} x(n)}{N}} = \frac{\exp\left(\frac{1}{N}\sum_{n=0}^{N-1} \ln x(n)\right)}{\frac{1}{N}\sum_{n=0}^{N-1} x(n)}$$

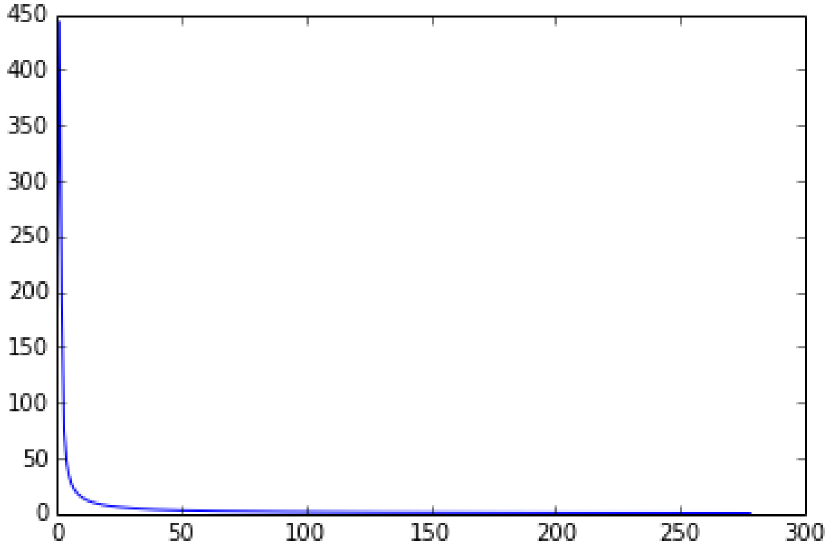
More complex audio features

Some problems with FFT/STFT

1. FFT has linear pitch scale: Perceptual differences between pitches don't match differences in FFTs

100 & 120Hz

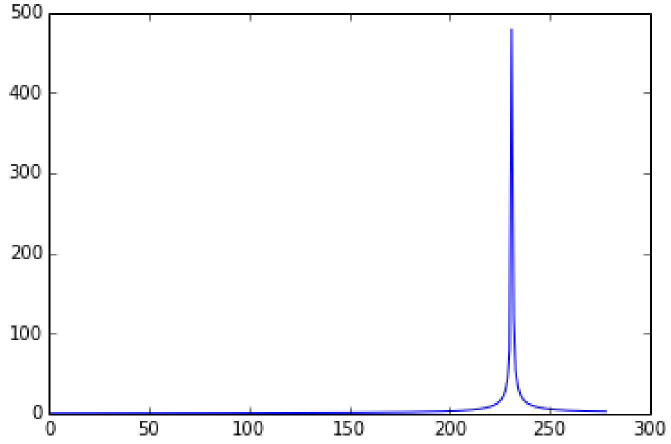


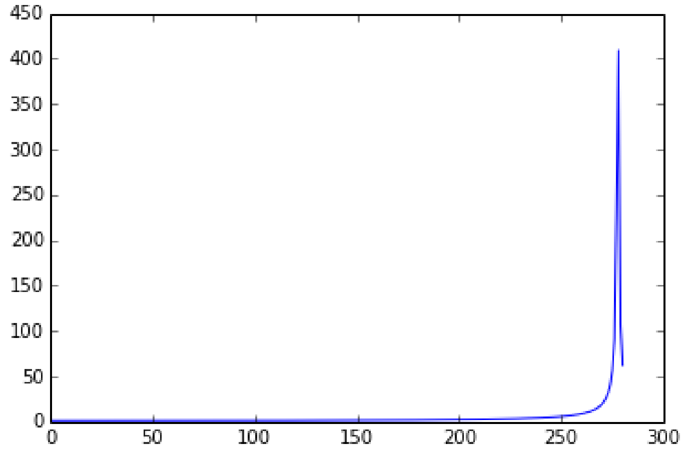


Some problems with FFT/STFT

1. FFT has linear pitch scale: Perceptual differences between pitches don't match differences in FFTs

1000 & 1200Hz





Some problems with FFT/STFT

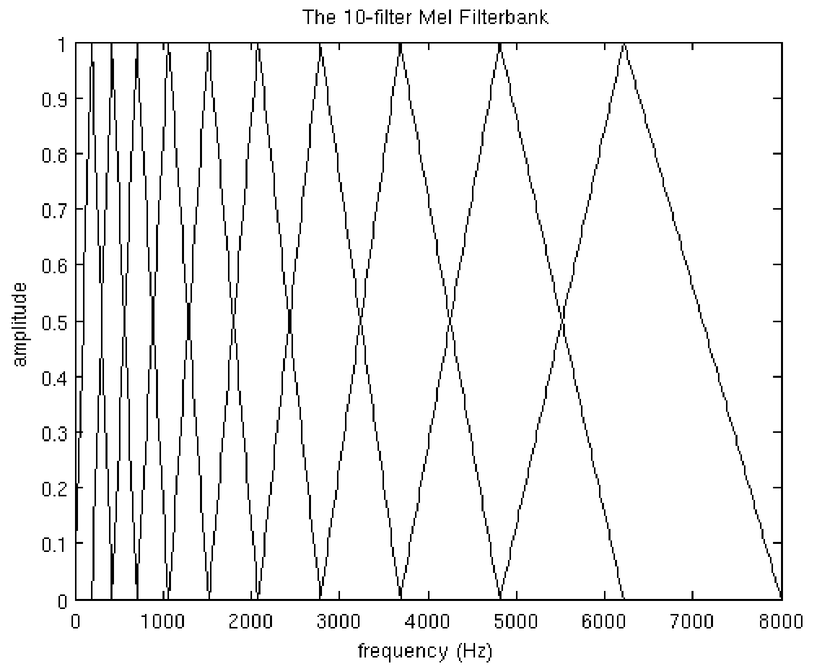
1. FFT has linear pitch scale: Perceptual differences between pitches don't match differences in FFTs

Solution: Transform FFT bins into perceptually-spaced bins (or use a slightly different transform)

2. Perceptual differences in volume are not accurately represented by differences in bin magnitudes

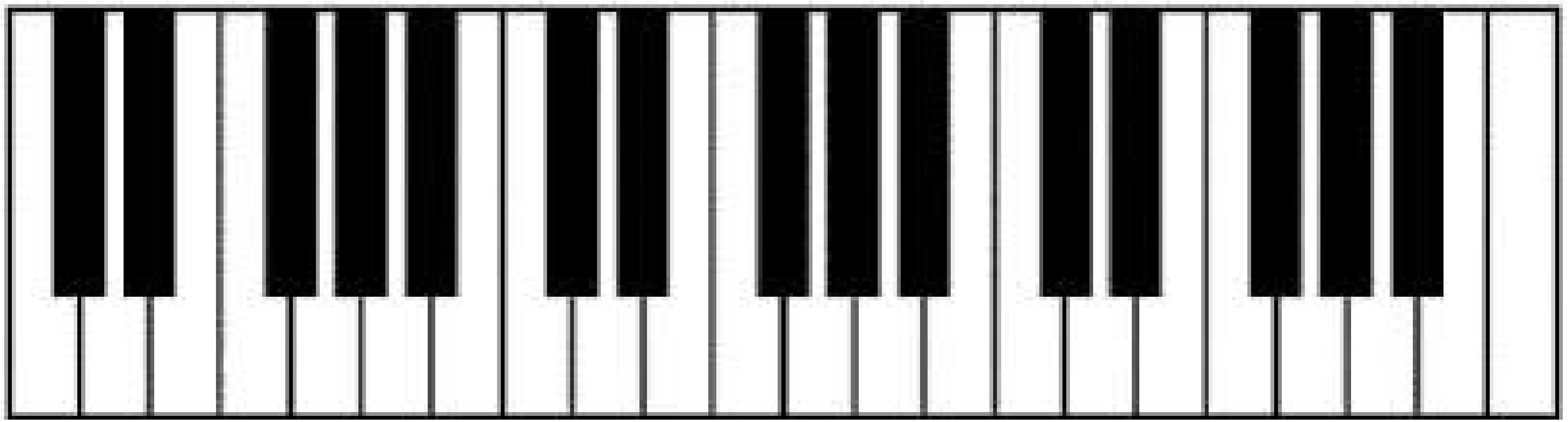
Simple solution: Take log of magnitude spectrum.
More complicated: Use bark scale coefficients

Mel scale



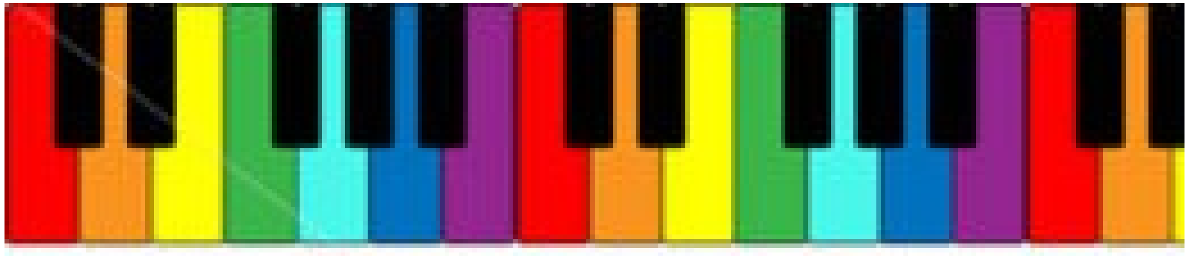
Can visualize mel coefficients instead of FFT bin magnitudes

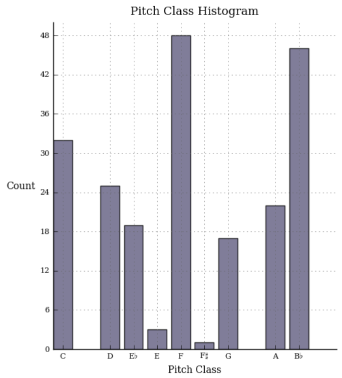
Or, combine bins to make one bigger bin per pitch:



Constant-Q with semitone bands

Or, combine all pitches of the same chroma (note name, e.g. all As, all Bb, etc.:





“Pitch histogram”

One more feature: speech & timbre (& instrumentation, genre, …)

Mel Frequency Cepstral Coefficents (MFCCS)

MFCCs

1. Take FFT
2. Convert magnitudes to mel scale
3. Take log of mel scale bins
4. Compute Discrete Cosine Transform (like FFT)

Spectrum of a spectrum: Cepstrum

Why???

Log of mel bins: because we hear loudness logarithmically

DCT: Low-quefrency bins tell us about overall shape of mel spectrum

Yes, that says quefrency.

More reading on MFCCs:

For good explanations:

- See slide 24 of http://eamusic.dartmouth.edu/~mcasey/m103/m103_8.pdf
- Also http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

Final comments on spectral features

Typically computed from STFT or equivalent (sliding analysis window)

May have many feature values per audio file/song

- This can help analyze song: e.g., find boundaries between verse/chorus, or periods of silence

- If doing analysis of a whole song, combine these in some way (e.g., take mean and standard deviation of 1st, 2nd, 3rd cepstral coefficients; this becomes new feature vector)

Example feature representations for real applications

Speech recognition

Window size 16ms-25ms, 50% overlap

Compute MFCCs

Use only first 13 MFCCs per window

Musical genre classification

- 100ms windows, 50% overlap
- Constant-Q spectrum with 12 bins per octave
- Add powers (ignoring subtle volume effects)
- Compute cepstrum, keeping first 20 cepstral coefficients

Cover song detection

- 500ms windows, 40% overlap
- Constant-Q spectrum with 12 bins per octave
- Octave folding (by adding powers)

How do I choose features?

- Depends on the context!
- Often, many features might work
- Use your judgment and knowledge of perception & task
- Experiment!

For more information

http://www.nyu.edu/classes/bello/MIR_files/timbre.pdf

http://earsketch.gatech.edu/learning/teaching-computers-to-listen-2-analysis-features

Audio feature extraction & visualization tools

Sonic Visualizer: Plug-ins for these audio features and MANY MORE

jAudio: free and cross-platform toolkit for extracting features from audio recordings

Example datasets

- Million song dataset:
  - Features already extracted (includes MFCC-derived features and meta-data features)
  - http://labrosa.ee.columbia.edu/millionsong/pages/example-track-descriptiona
- Audio & symbolic datasets commonly used in music IR:
  - http://grh.mur.at/sites/default/files/mir_datasets_0.html