CS 598PS Machine Learning for Signal Processing

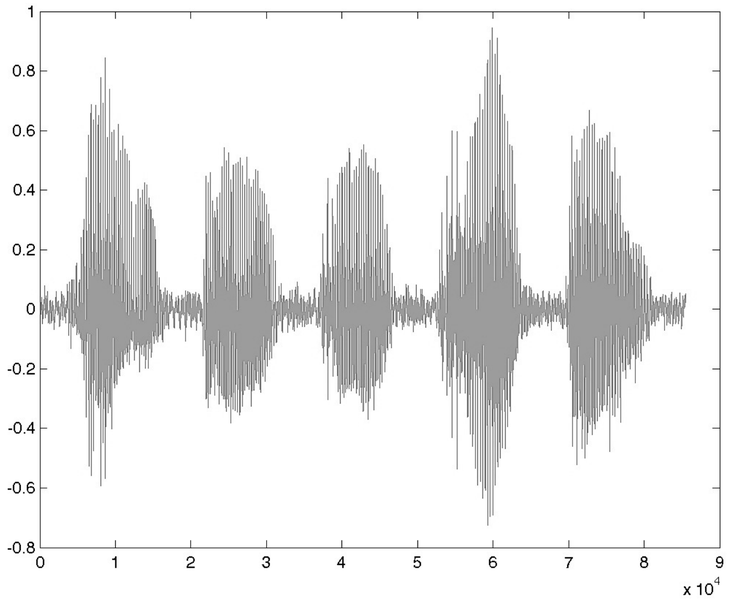
Time Series and Signal Processing Primer

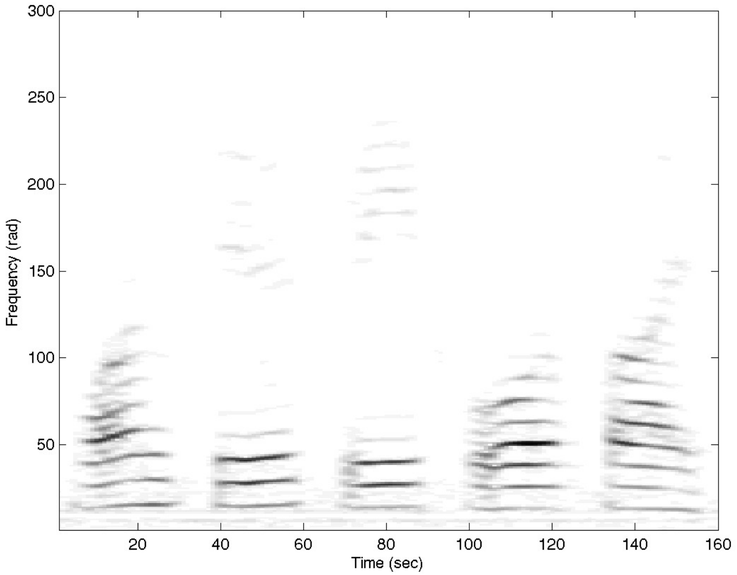
1 September 2020

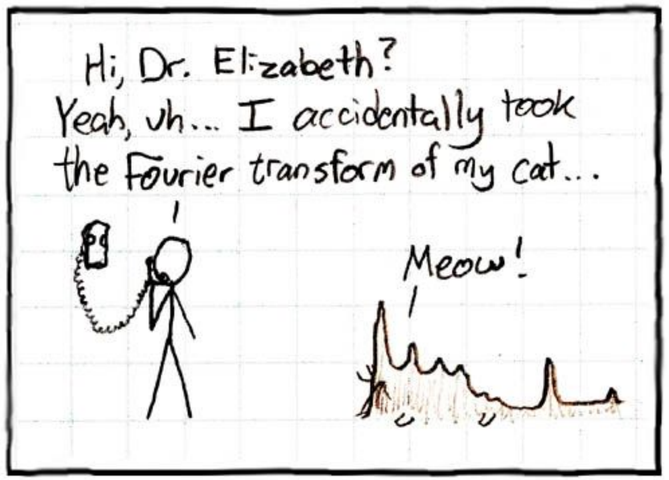
** waiting for the late logins ...*

Overview

- Basics of time series
- Signal representations
  - Time vs. frequency vs. time/frequency
  - Sampling, quantization
- The Fourier transform
- Filtering
  - Convolution
  - Fast convolution
- Resampling



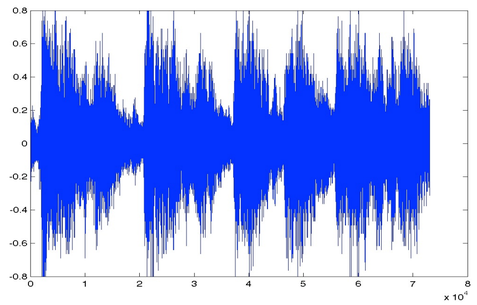




Time series

- An ordered collection of numbers
  - E.g. Temperatures throughout the year
  - Height/weight throughout a lifetime
  - Stock prices across time
  - Sound or video
- If order is important, then what you have is a time series
  - Doesn't have to be throughout time!
  - Images are seen as time-series because left-to-right and up-down order matters

Average Temperature											
Jan	Feb	Mar	Apr	May	June	July	Aug	Sept	Oct	Nov	Dec
28.8	29.4	37.1	47.2	57.9	67.2	72.7	71	64.1	54.0	43.7	32.8

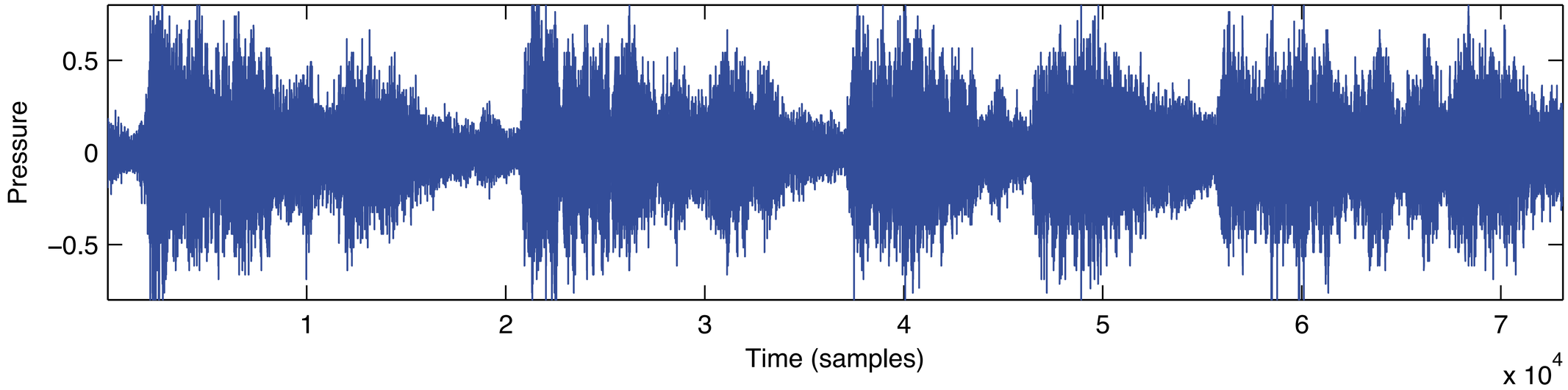


Representing signals

- There are multiple ways to represent time series
  - Some are good, some are bad, some are convenient
- A useful representation is the frequency domain
  - Provides an alternative view that reveals information
  - Is very mathematically convenient!

Sound as a time series

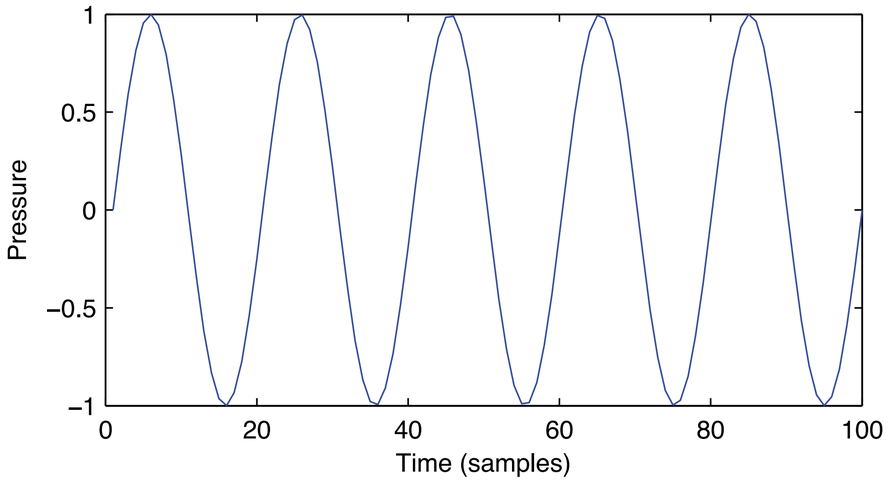
- Sounds are a typical case of time series

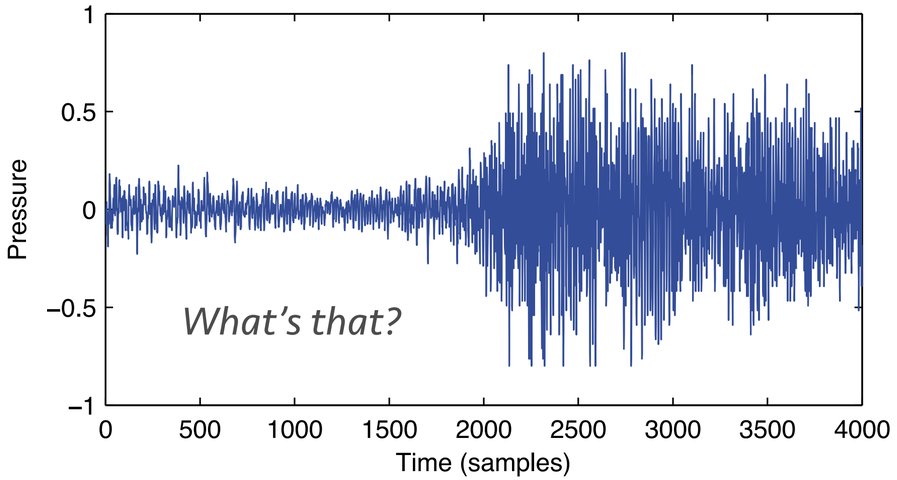


- We'll start with sounds as an illustrative example
  - But everything here works for all 1-D series

The time domain

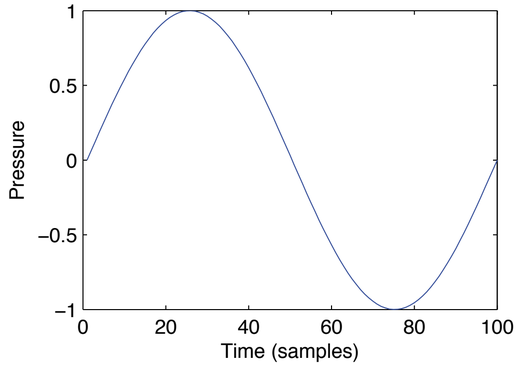
- Sound is one-dimensional
  - "Waveform" format
  - Series of numbers denoting instantaneous air pressure
- Not very intuitive for complex sounds like speech and music
  - Can display some information about the energy and the length of the signal
  - But not what it sounds like!



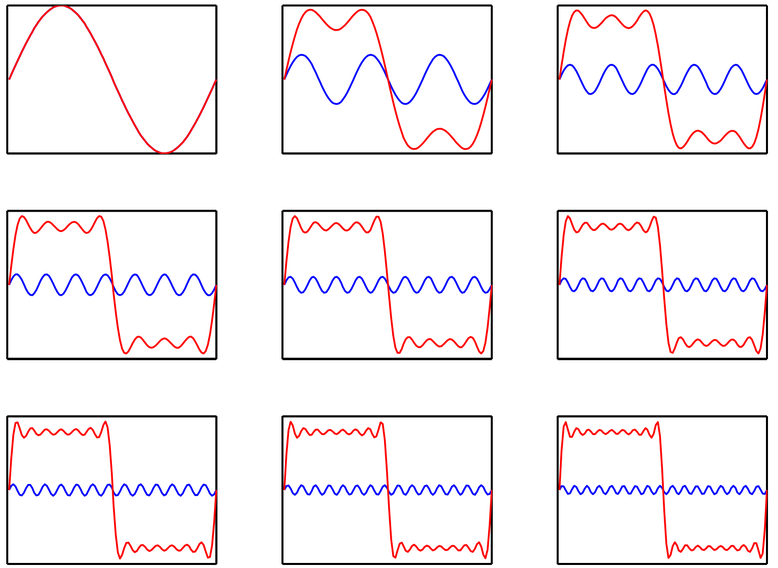


Building blocks for sound

- Sinusoids are special
  - Simplest waveform – a single frequency
    - More on that later …
- A sinusoid has three parameters
  - Frequency, amplitude & phase
    - $s(t) = a \sin(f t + \varphi)$
  - Its simplicity makes sinusoids excellent building blocks for many time series

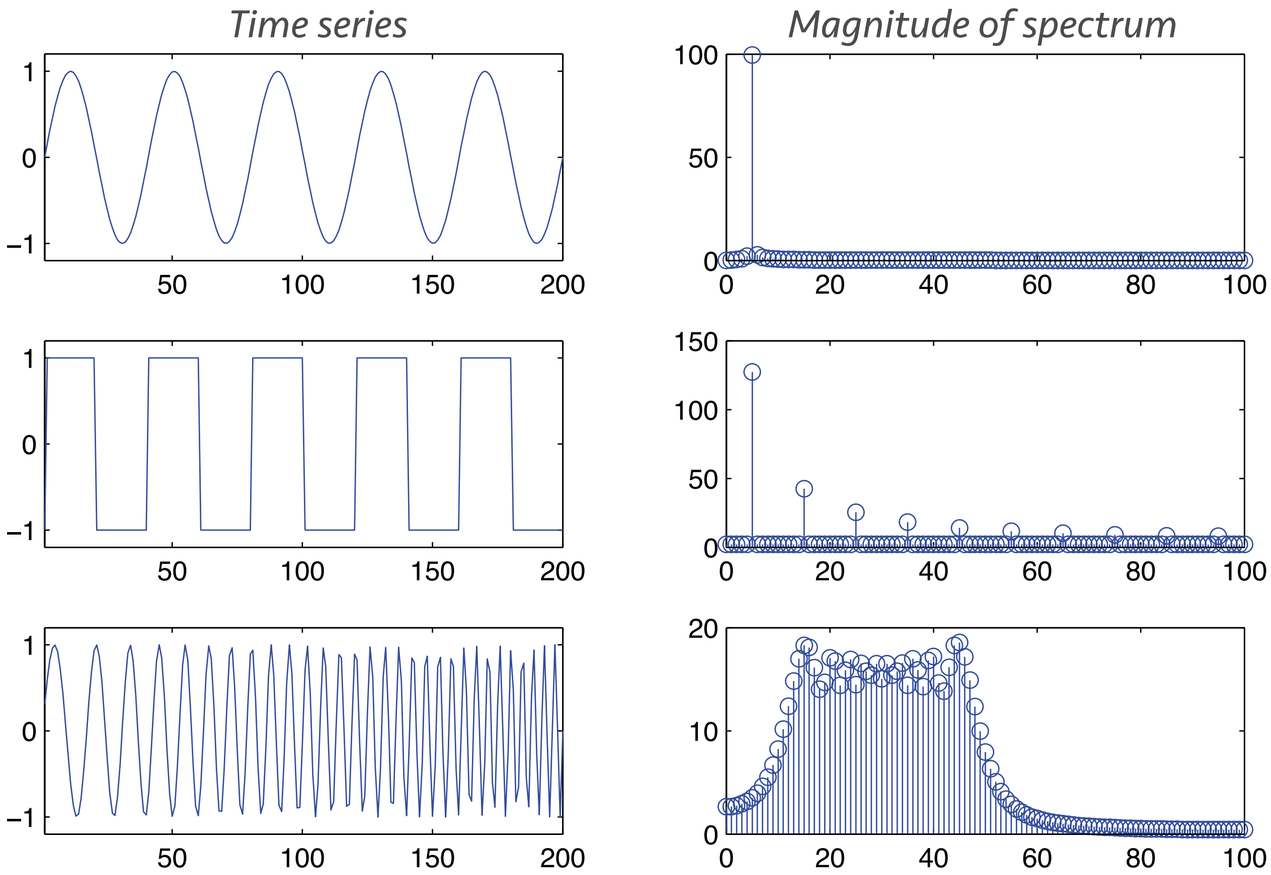


Making a square wave with sines



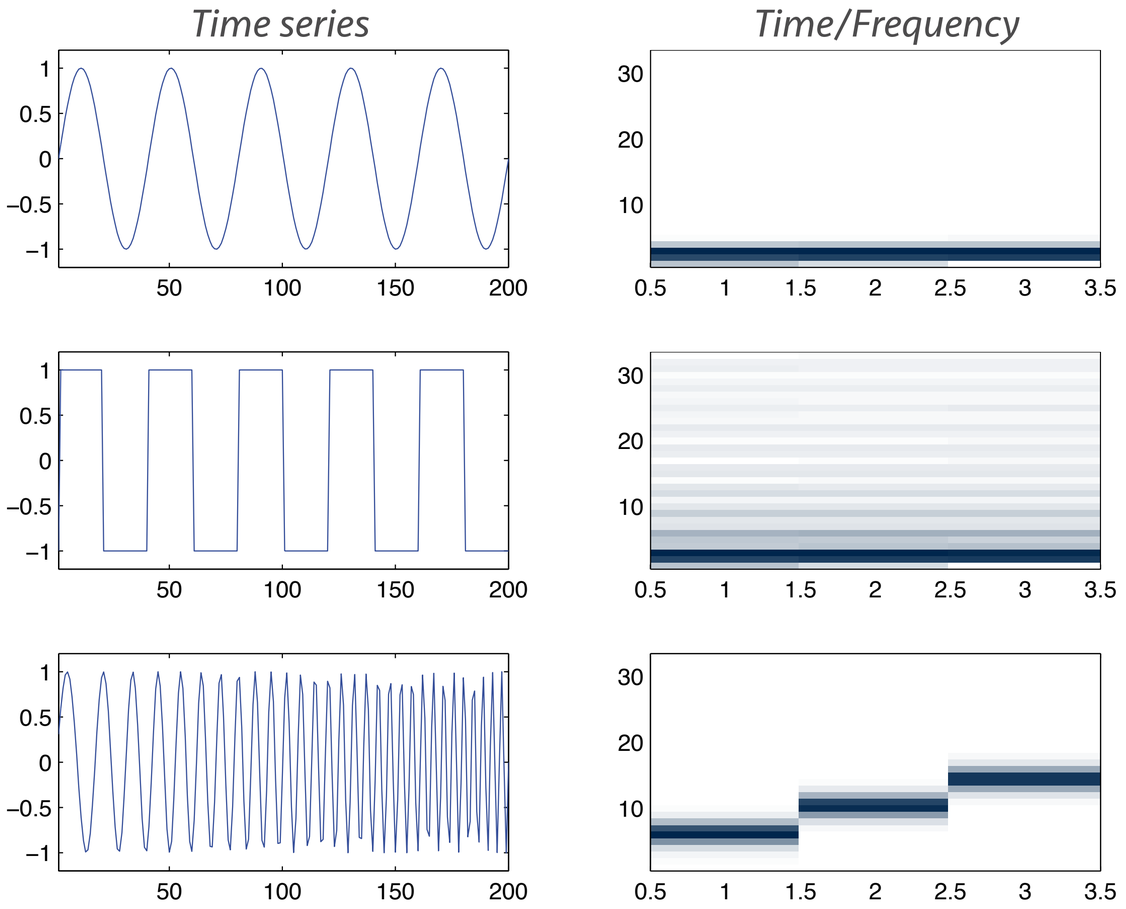
Frequency domain representation

- Time series can be decomposed in terms of sinusoidal building blocks
  - That is called the *spectrum*
- There is no temporal information in the spectrum, only frequencies
- Not that great of a representation for dynamically changing sounds



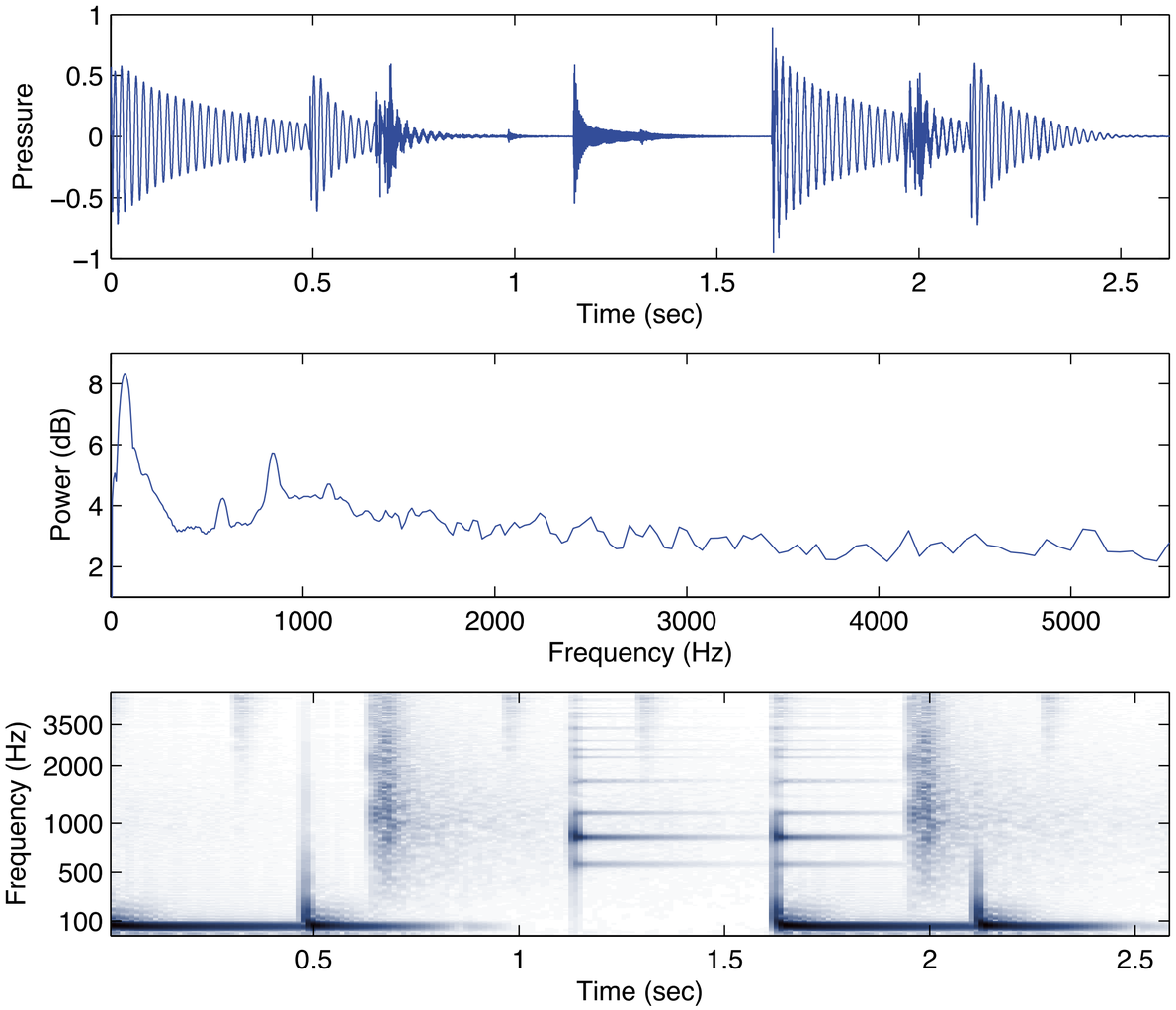
Time/frequency representation

- Many names/varieties
  - Spectrogram, sonogram, periodogram, …
- A time-ordered sequence of frequency compositions
  - Can help show how things change in both time and frequency
- Most useful representation so far!
  - Reveals information about both the time and frequency content without loss



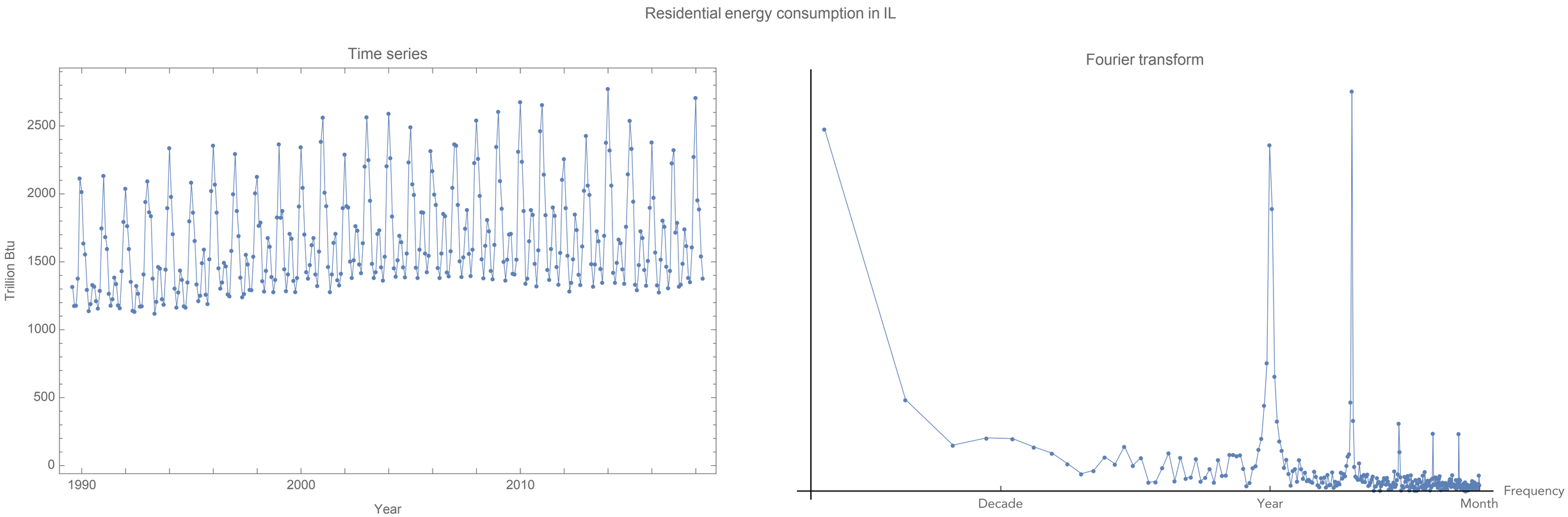
A real example

- Time domain
  - We see the events
    - How do they sound like though?
- Frequency domain
  - We see bass and mids
    - Where are they though?
- Spectrogram
  - We "see" all the sounds
    - And have a sense of how they sound



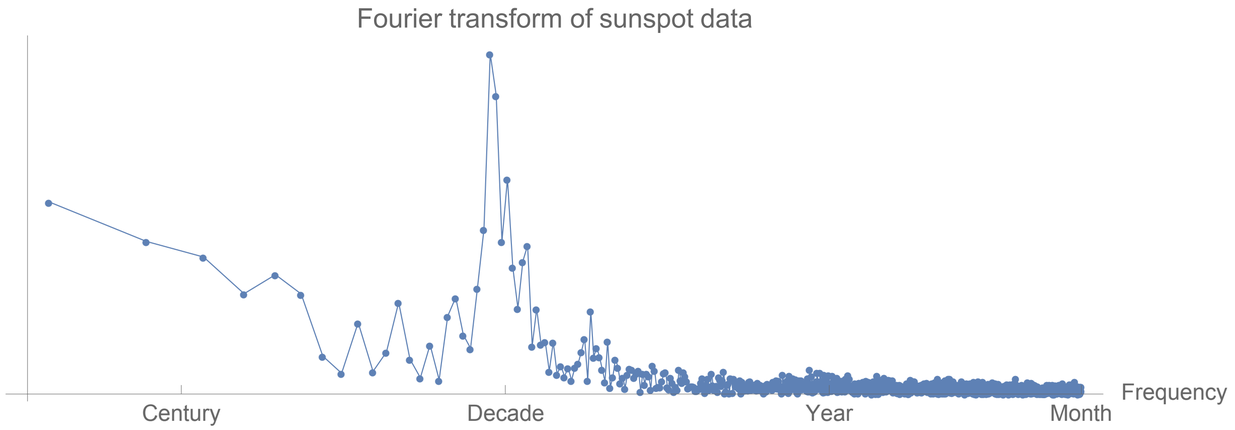
What about other kinds of data?

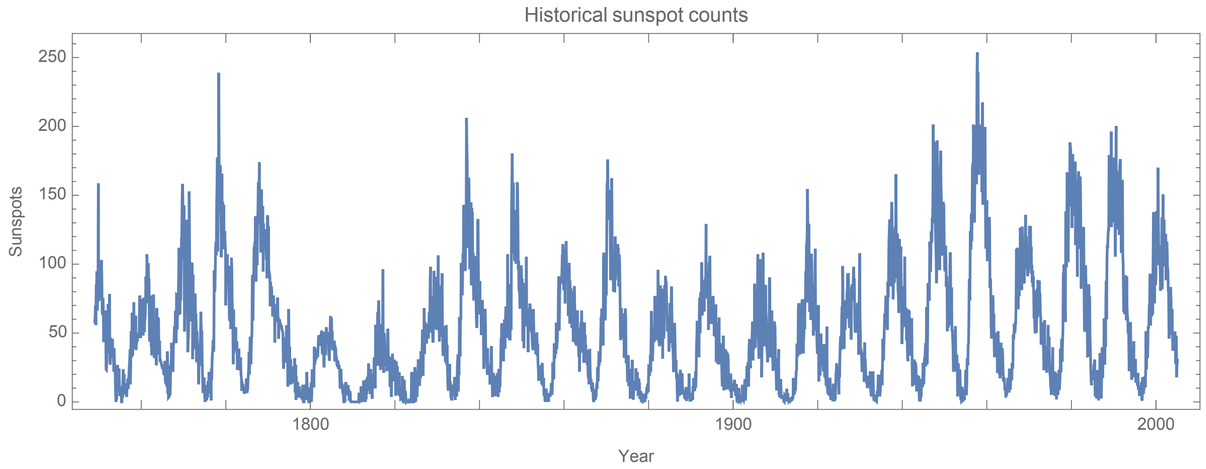
- Works with all types of time series
  - E.g. Energy consumption

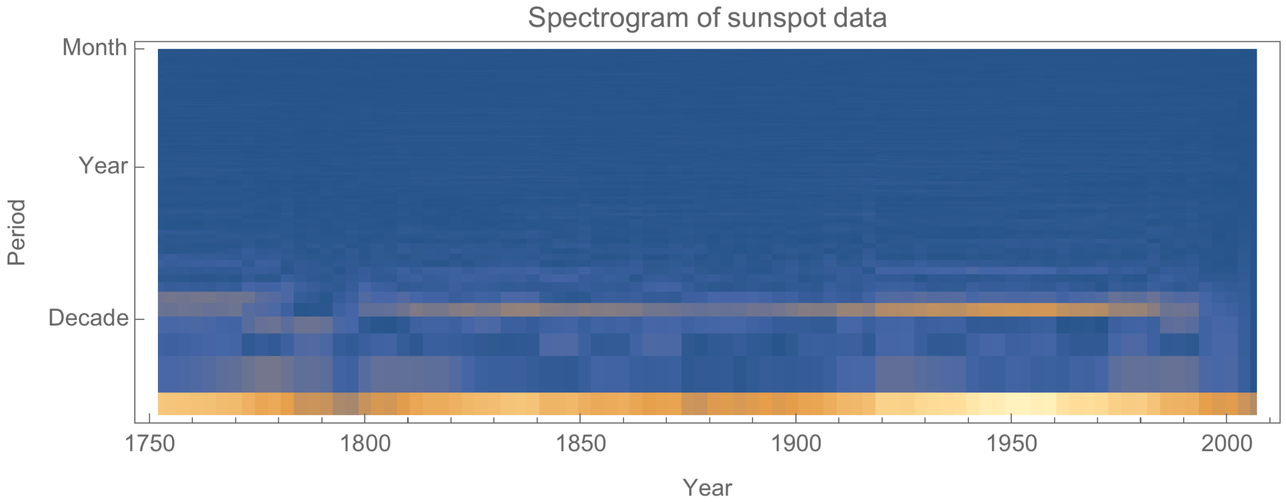


How about astronomy?

- Sunspot count data
  - Fourier reveals sunspot period
  - Spectrogram reveals more info





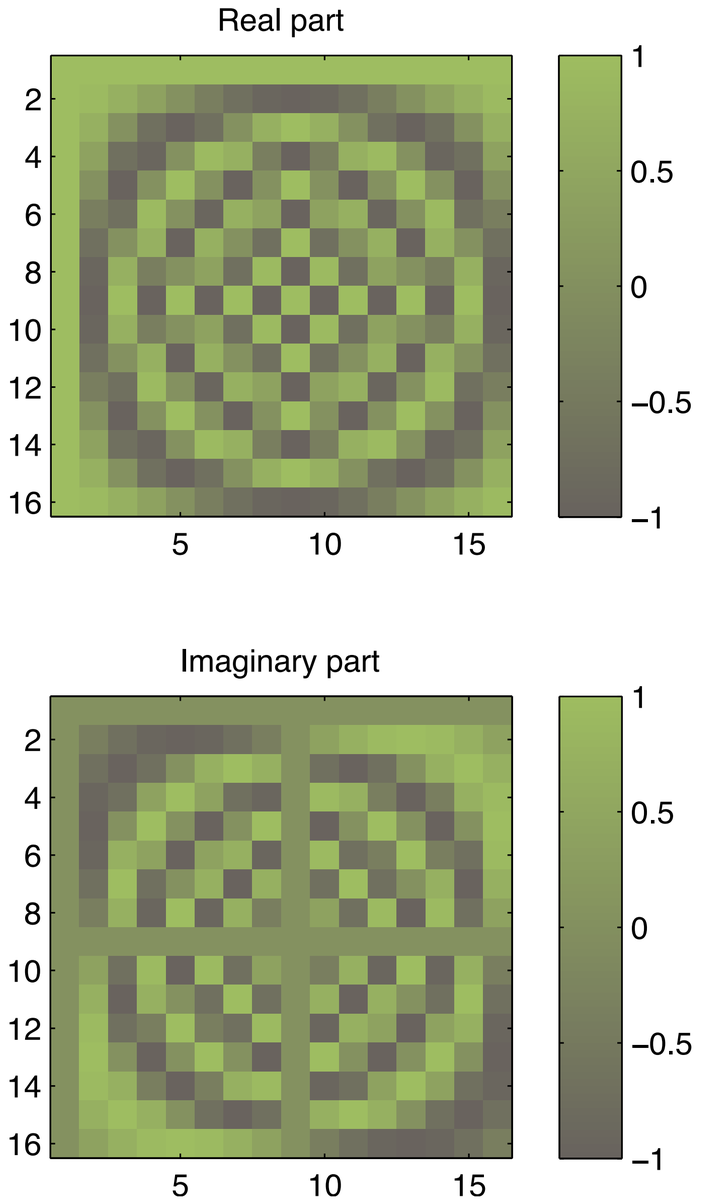


The Discrete Fourier Transform (DFT)

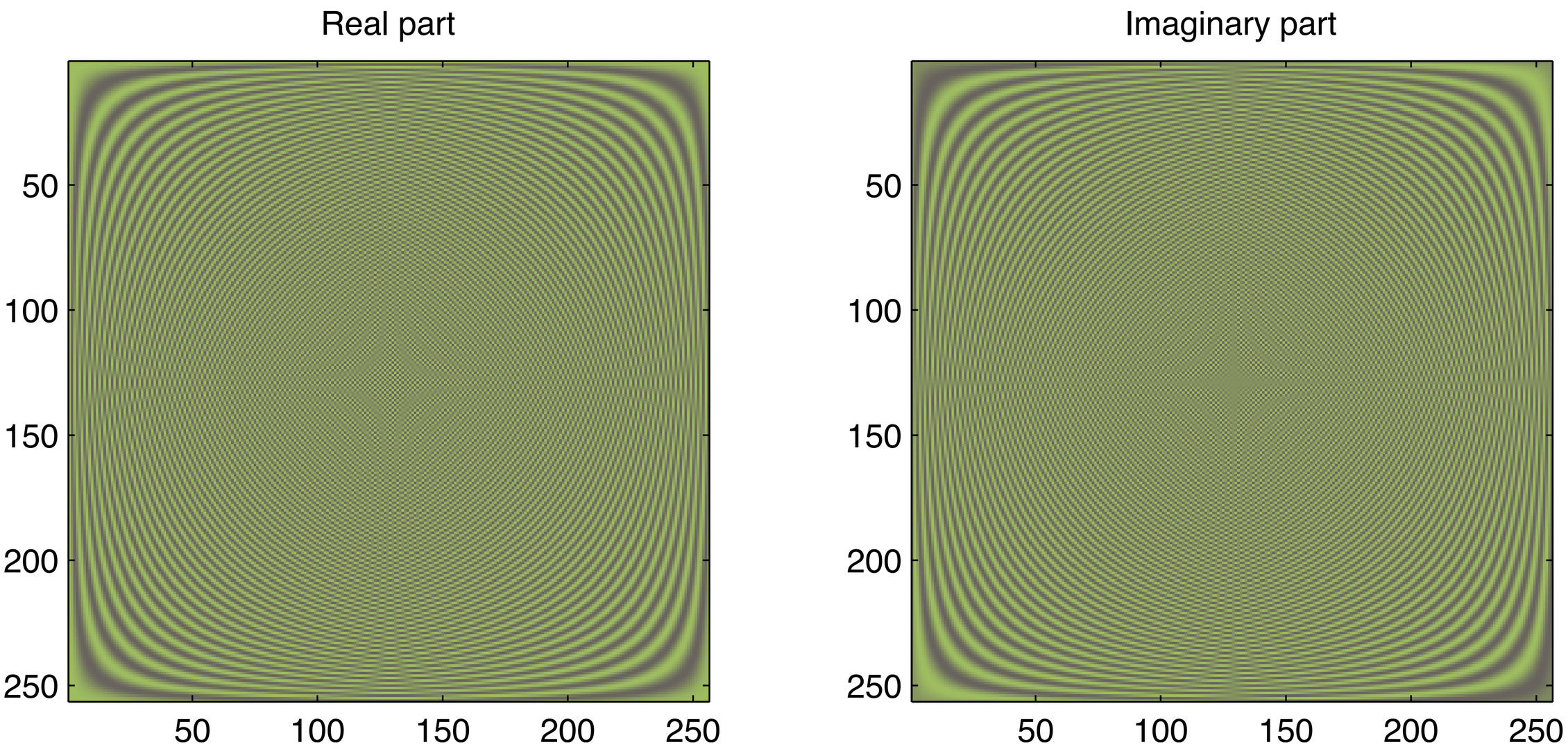
- So how do we get from the time domain to the frequency domain?
  - It is a linear transform → a matrix multiplication
- The Fourier matrix is square, orthogonal and has complex-valued elements

$$F_{j,k} = \frac{1}{\sqrt{N}} e^{ijk\frac{2\pi}{N}} = \frac{1}{\sqrt{N}}\left(\cos\frac{jk2\pi}{N} + i\sin\frac{jk2\pi}{N}\right)$$

  - Just multiply a vectorized time-series with the Fourier matrix and voila!

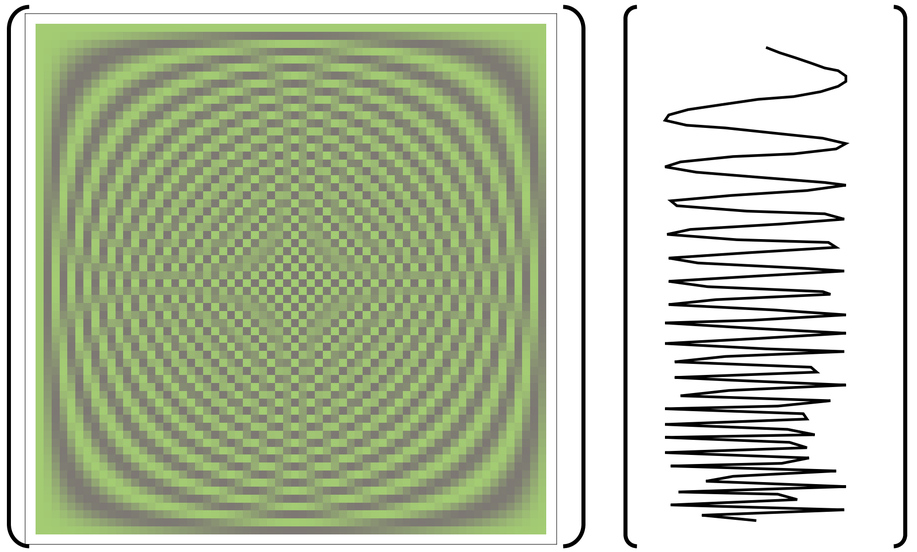


And for bigger sizes



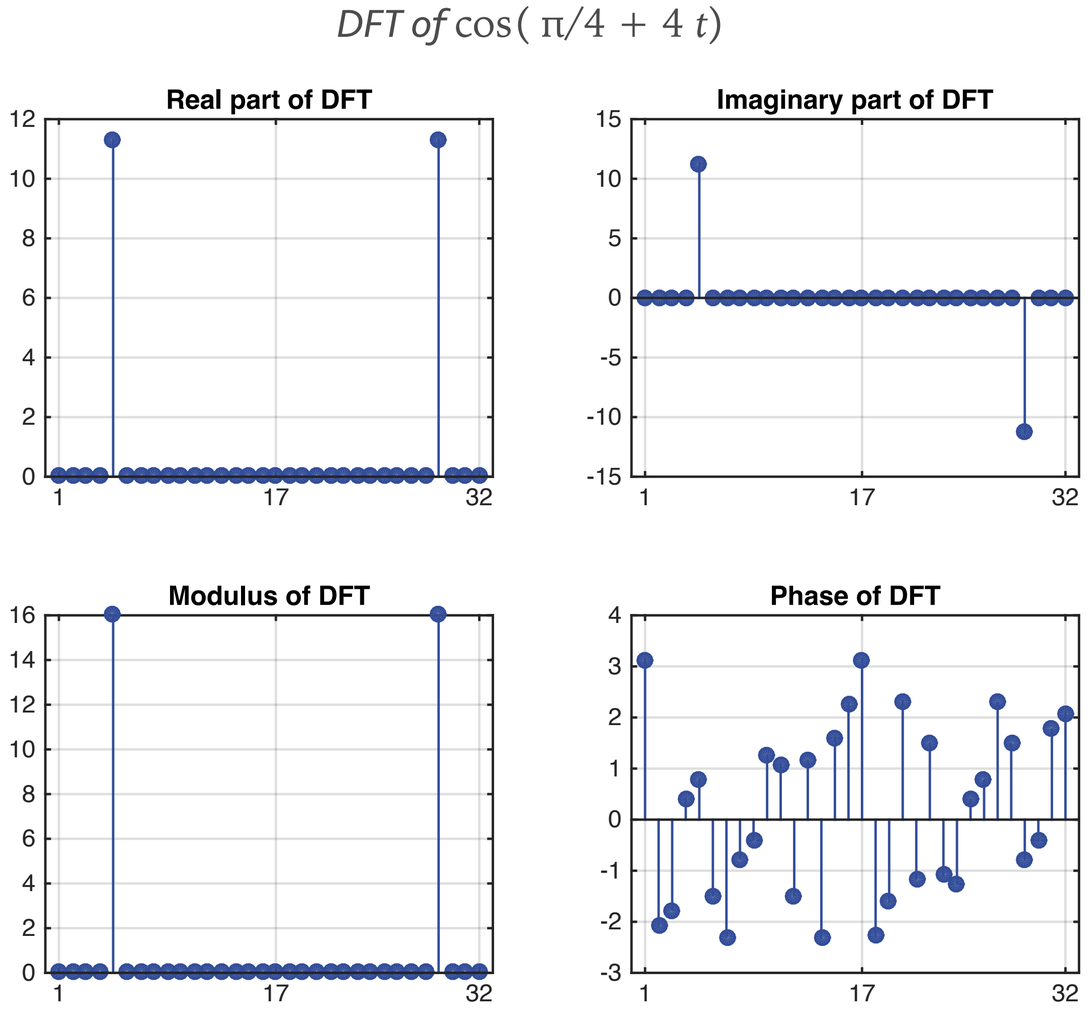
How does the DFT work?

- Multiplying with the Fourier matrix
  - Dot-product each Fourier matrix row with input
- Each Fourier row focuses on a single frequency from the signal
  - Since all the Fourier sinusoids are orthogonal there is no overlap
- The resulting vector describes how much of each sinusoid the original vector has



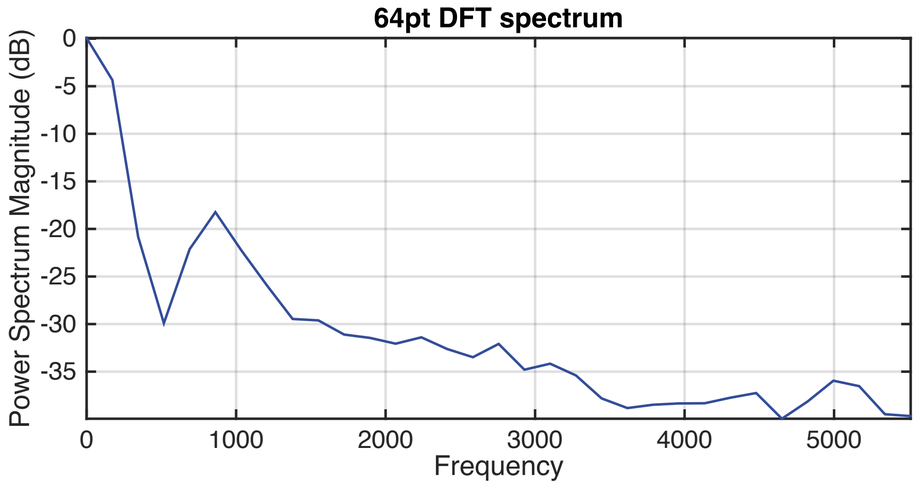
The DFT in a little more detail

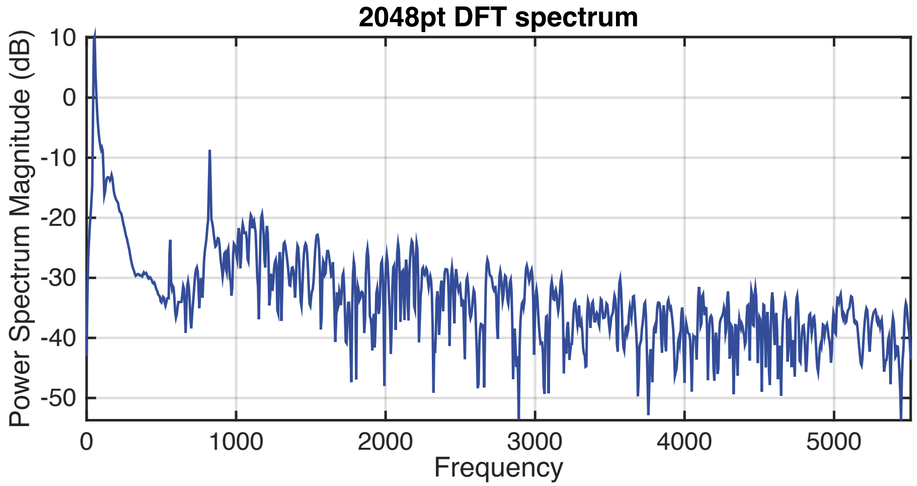
- The DFT returns complex numbers
- For real signals DFT is conjugate symmetric
  - The middle value represents the highest frequency
  - The two halves are mirrored complex conjugates
- The interesting parts of the DFT are the magnitude and the phase
  - $\text{Abs}(\text{F}) = \|\text{F}\|$
  - $\text{Arg}(\text{F}) = \measuredangle \text{F}$
- To go back to the time domain we apply the DFT again (with some scaling)



Size of a DFT

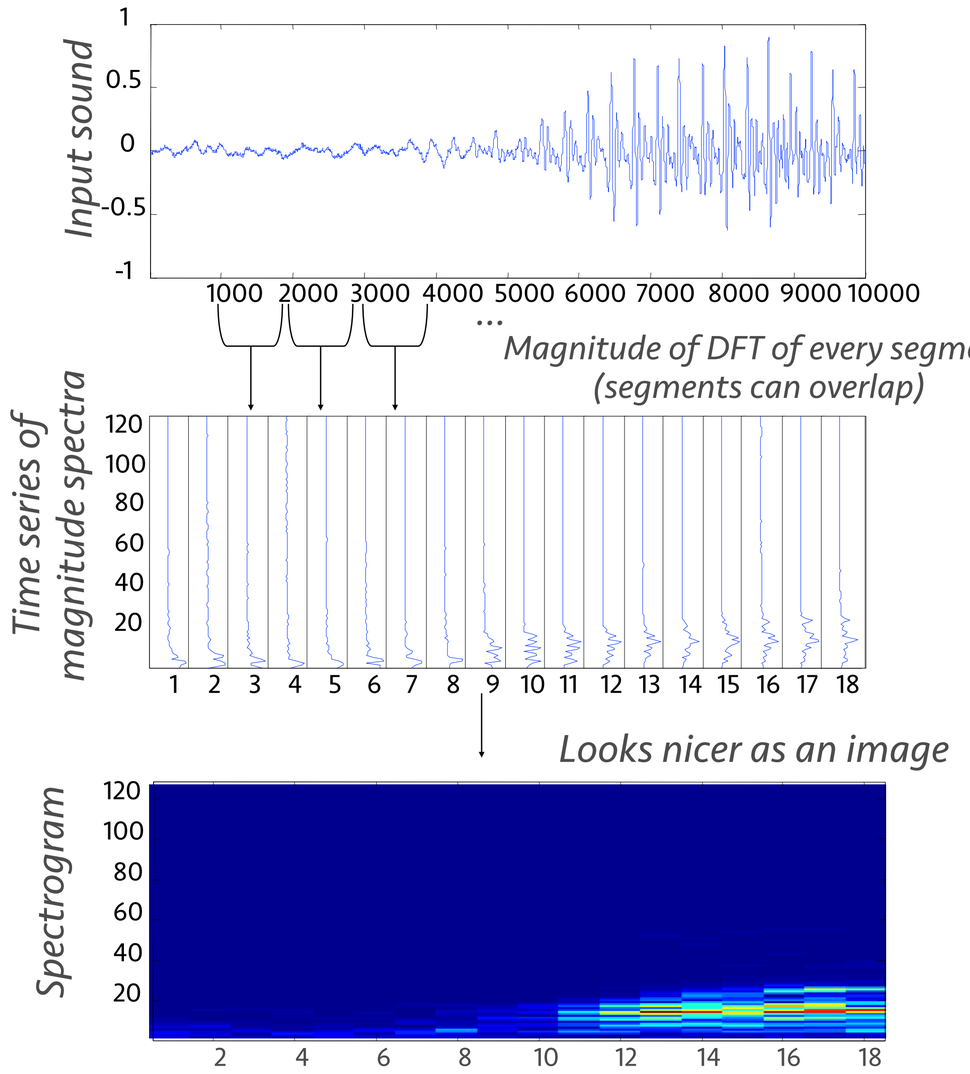
- The bigger the DFT input the more frequency resolution
  - But the more data we need!
- Zero padding
  - Append a lot of zeros at the end of the input to make up for small inputs
    - But we don't really infuse any more information we just make prettier plots





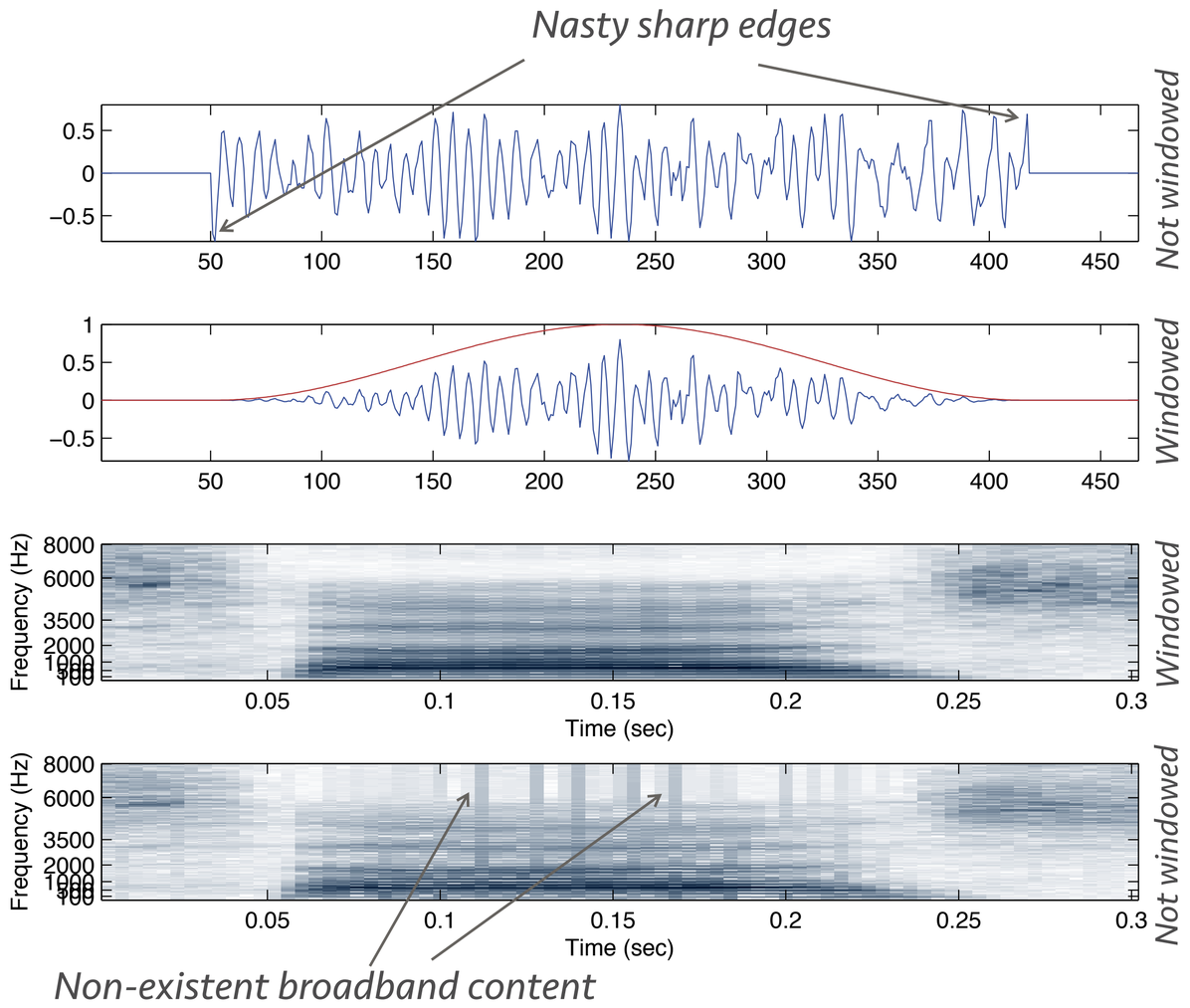
From the DFT to a time/frequency

- The spectrogram is a series of DFTs
  - They are applied on successive input segments
    - Which can overlap if desired
- We most often only show the magnitude of the result
  - But we also need the phase for reconstruction
- The parameters to use are
  - The DFT size, the overlap, and the *window*



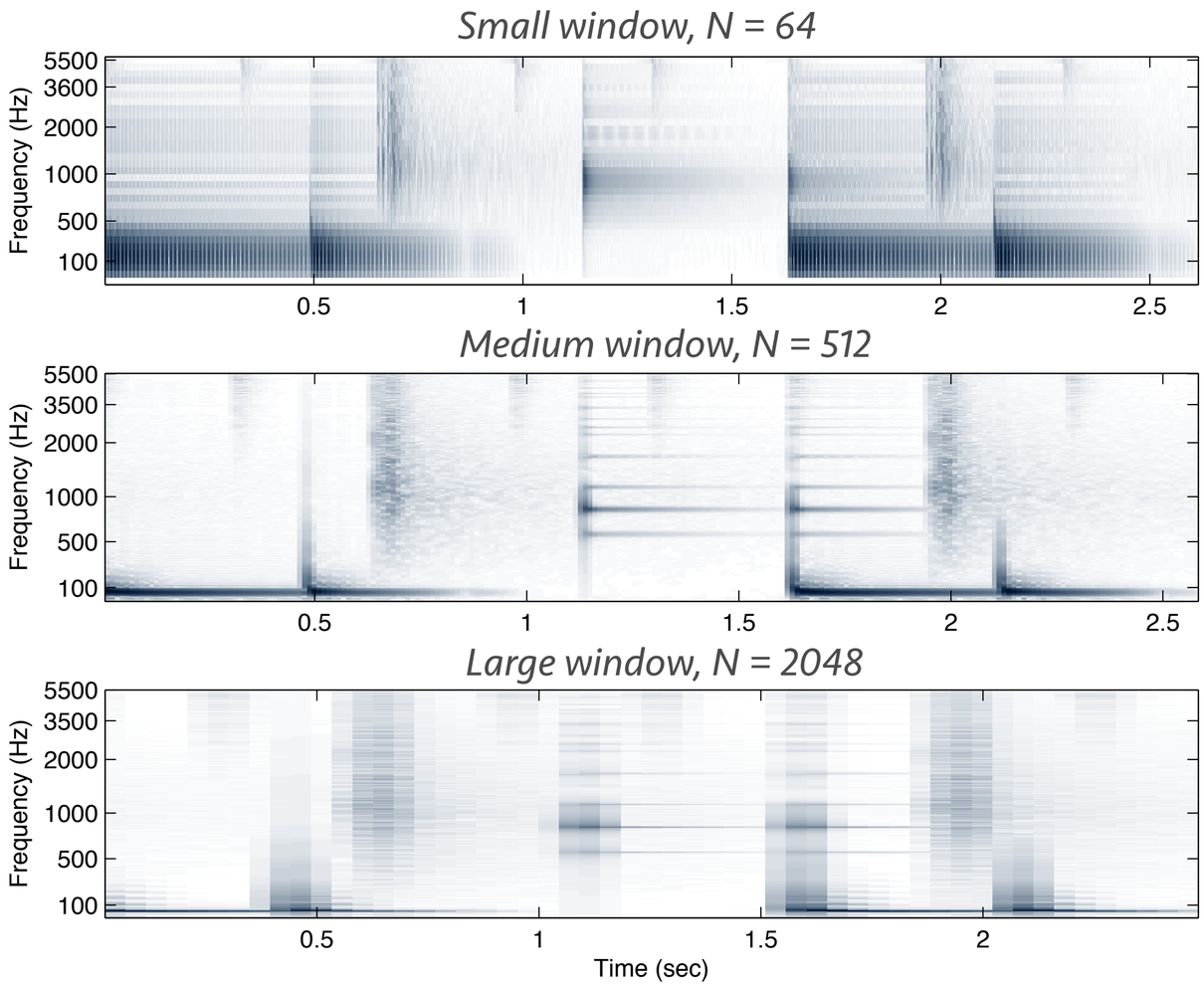
Why window?

- The DFT is periodic
  - The Fourier sinusoids extend infinitely
  - If ends do not match there are discontinuities
  - These imply more sinusoids to explain them
- *Windowing*
  - Tapers the sharp edges of the input and removes the unwanted discontinuities
  - At the expense of changing the input though!
- Overlap
  - We need to take overlapping segments to make up for the attenuated parts of the input



Time/Frequency tradeoff

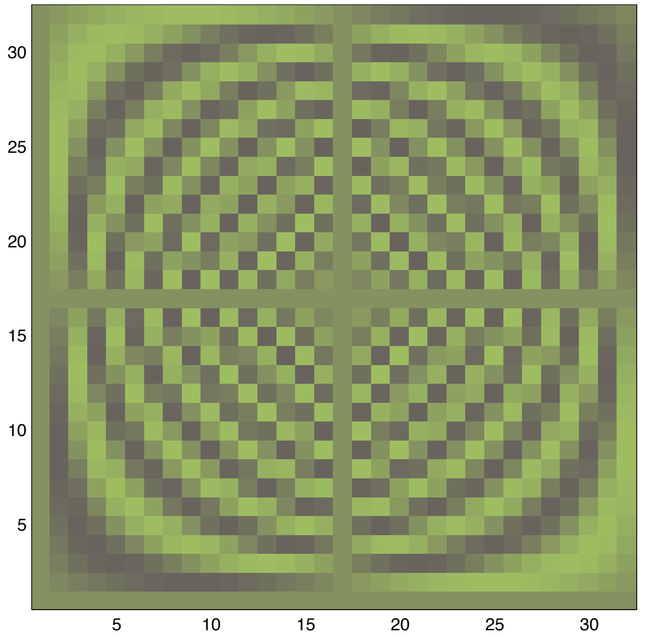
- Heisenberg's uncertainty principle
  - We can't accurately know both the frequency and time location of a wave
- Spectrogram problems
  - Big DFTs sacrifice temporal resolution
  - Small DFTs have bad frequency resolution
- We can use a denser overlap to compensate for time resolution
  - Ok solution, not great

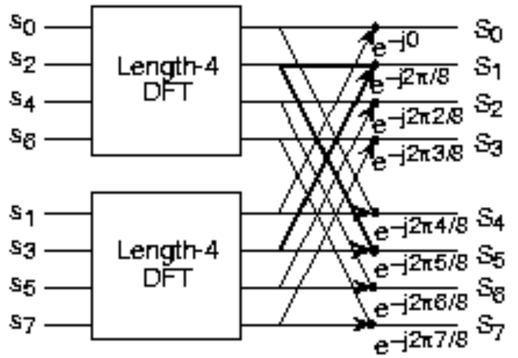


The Fast Fourier Transform (FFT)

- The Fourier matrix is special
  - Unique repeating structure with strong symmetries
- We can decompose a Fourier transform to a function of two Fourier transforms of half the size
  - Two smaller Fourier transforms are faster than one big one
  - We keep decomposing it until we have lots of 2-point DFT
  - Power-of-2 sizes work best
- If you can do something with an FFT, do it
  - It will always result in a computational gain

The Fourier matrix, N = 32

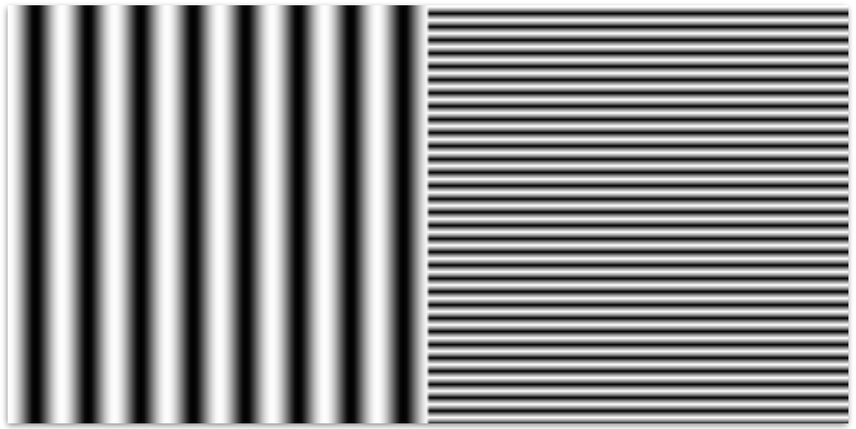




Example FFT, N = 8

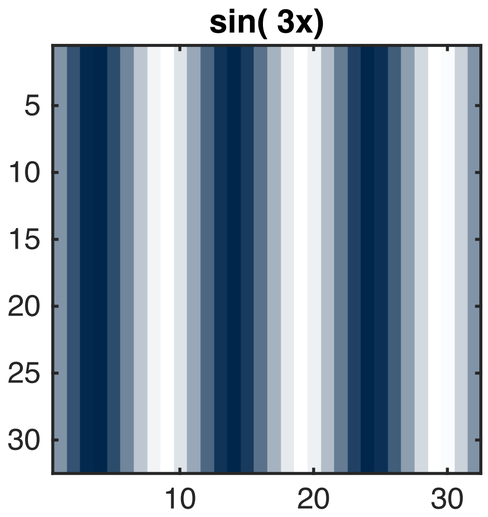
Image DFTs

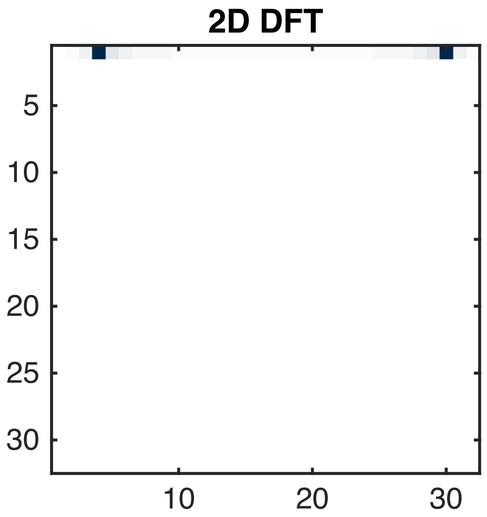
- Images are 2d so we need to generalize
- Fourier bases are vertical and horizontal 2d sinusoids
  - Sinusoidal on one axis, constant on the other
  - e.g:

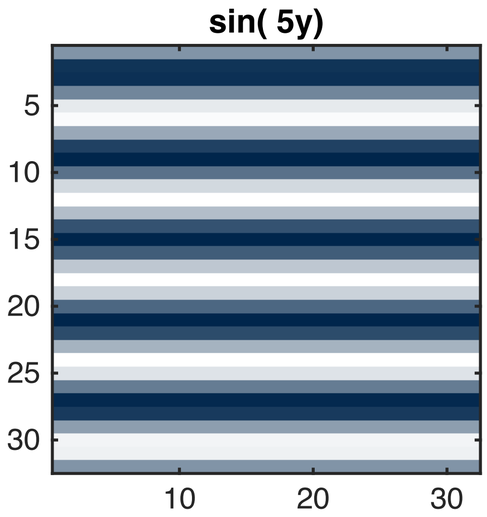


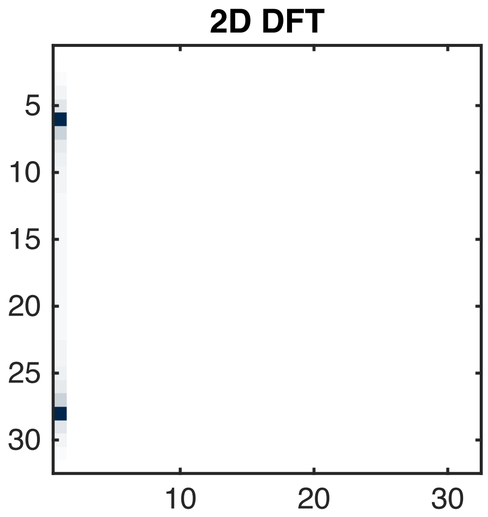
  - Images get decomposed using this basis set

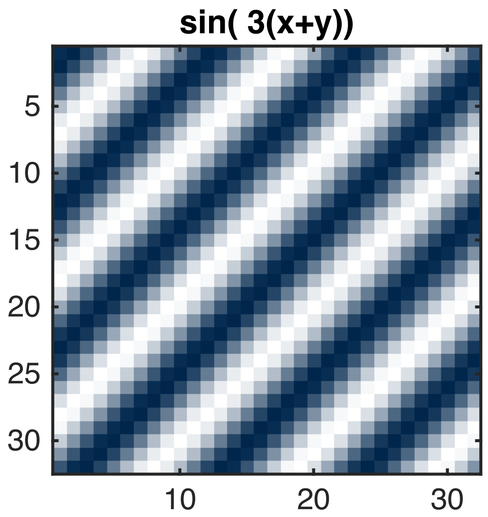
Some example DFTs

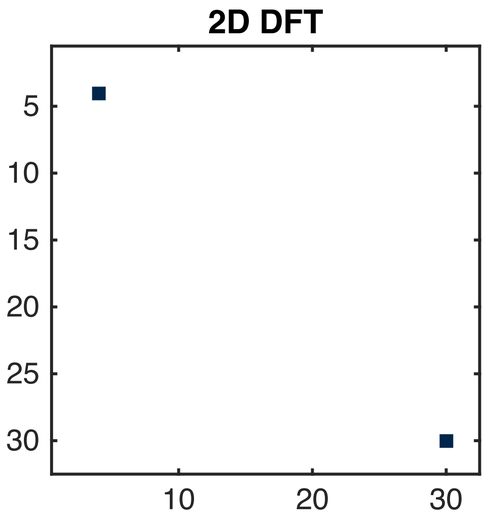


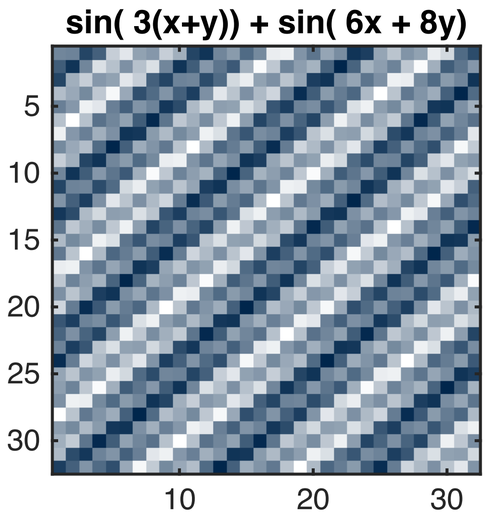


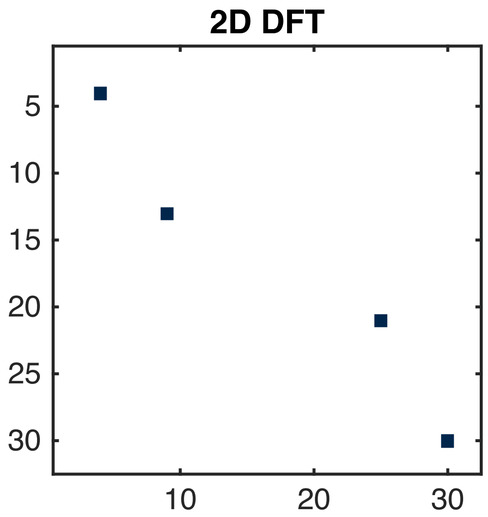






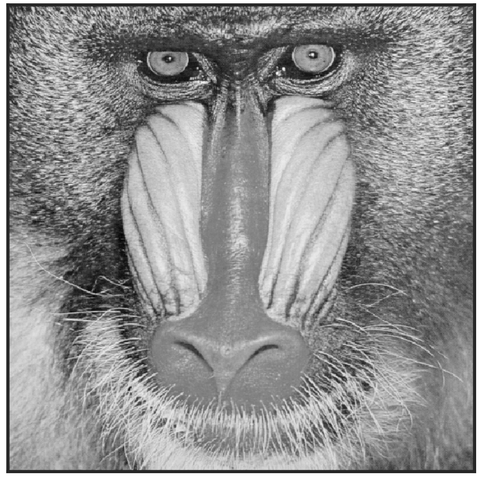




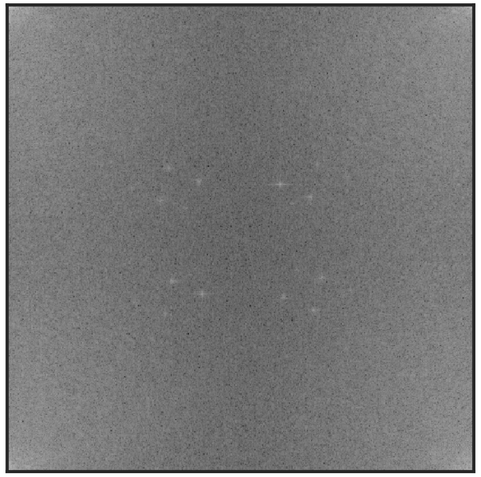


And some real image examples

Input

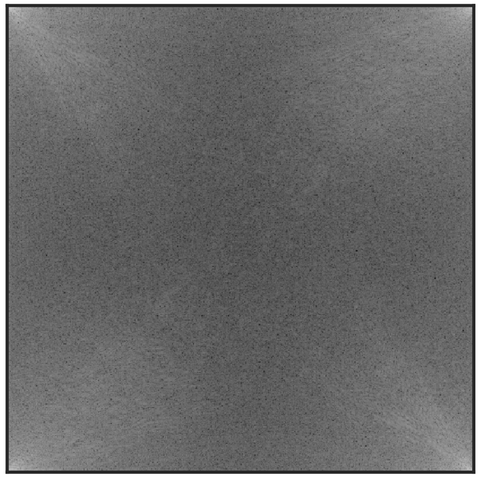


2D DFT



Input

2D DFT

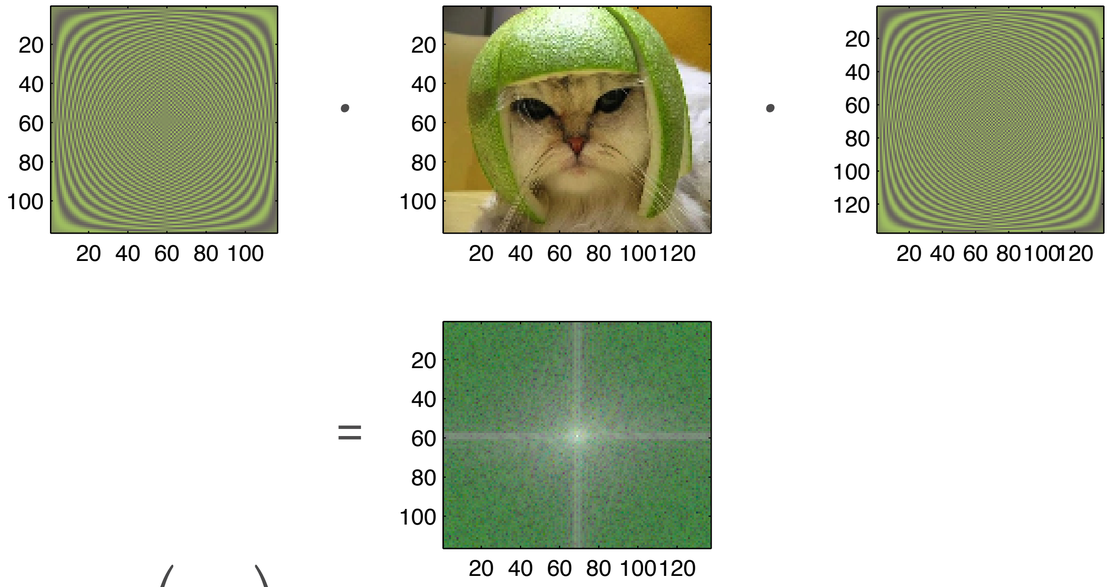


How do we compute that?

- DFT both sides of the data matrix

$$\mathbf{Y} = \mathbf{F} \cdot \mathbf{X} \cdot \mathbf{F}$$

$$\mathbf{X} \in \mathbb{C}^{N,N}, \mathbf{F}_{j,k} = \frac{1}{\sqrt{N}} e^{ijk\frac{2\pi}{N}}$$

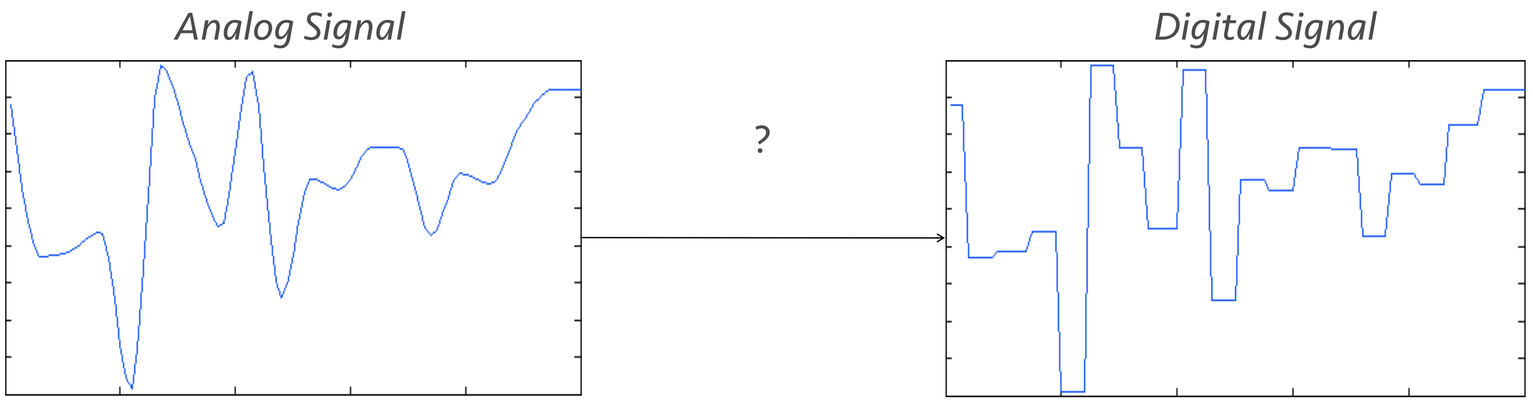


- What about tensors?
  - Same thing …

$$\text{vec}\left(\mathbf{Y}\right) = \left(\mathbf{F} \otimes \mathbf{F} \otimes \cdots \otimes \mathbf{F}\right) \cdot \text{vec}\left(\mathbf{X}\right)$$

Sampling Signals

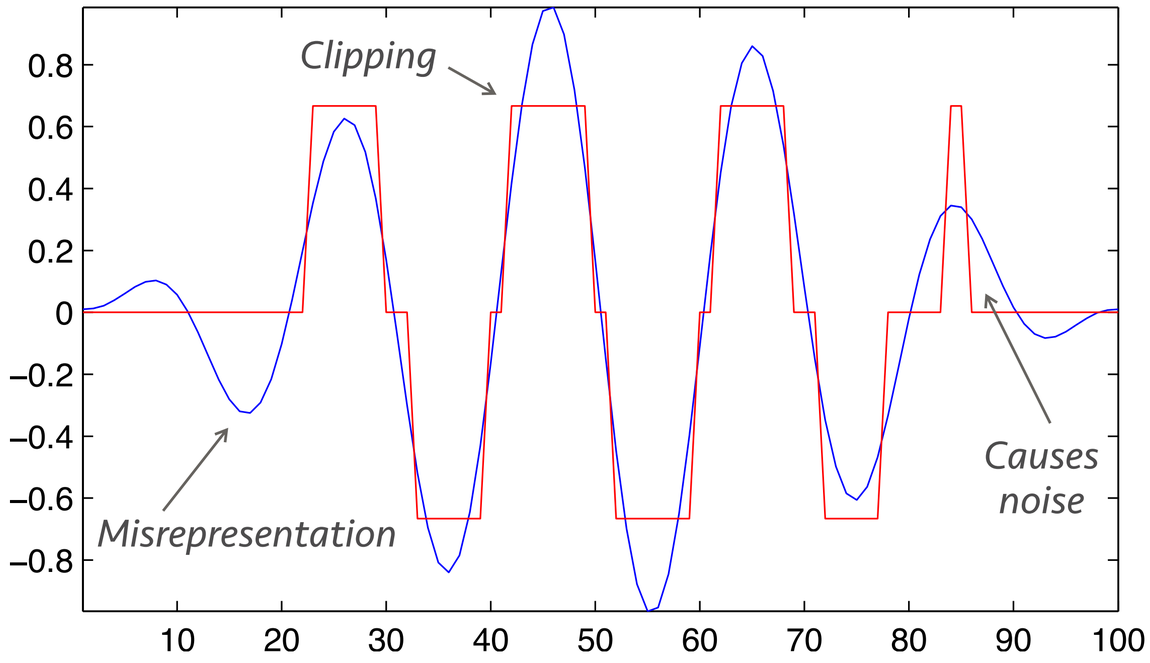
- How do we obtain a signal?
  - Almost always we convert from analog



  - Not a straightforward operation!

Signal representation - Quantization

- Converting a real input to a digital number requires *quantization*
- We need adequate resolution to represent the original measurements
  - Precision is measured in bits
  - Possible noise with small inputs
  - Possible distortion with large inputs
- Bit precision (headroom)
  - 8-bit = 48dB, poor
  - 12-bits = 72dB, maybe ok
  - 16-bits = 96dB, good
  - 24-bits = 144dB, maybe little too much
  - Our ears get about 110dB, eyes 90dB



Quantization in practice


12 bits


8 bits


4 bits


3 bits

Why dynamic range matters

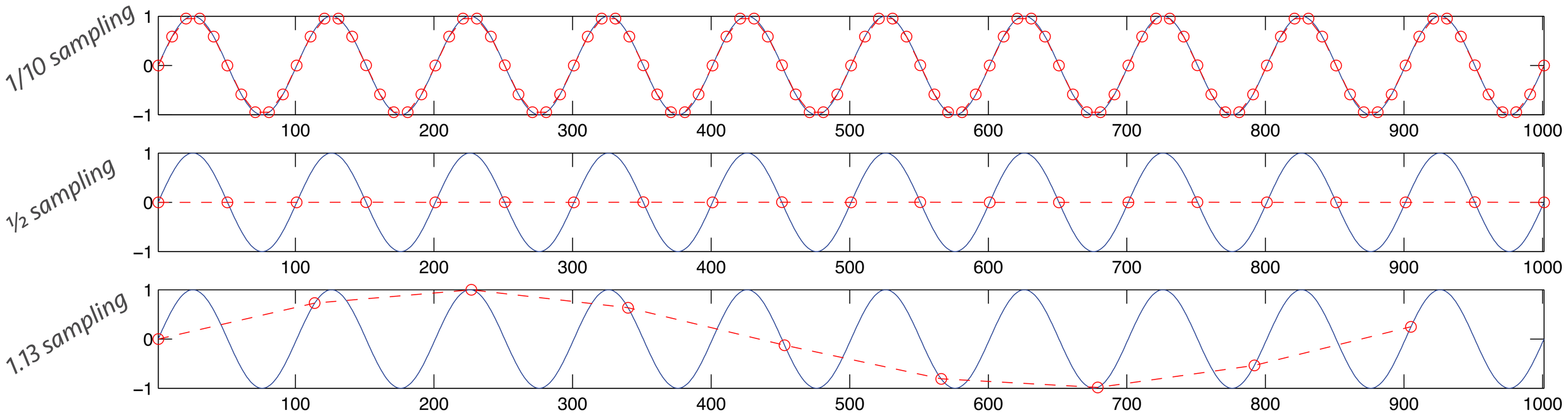
https://www.youtube.com/watch?v=gp_D8r-2hwk

Sampling

- How often we sample measurements is also important!!
  - 1d time-series sampling is usually measured in Hz
- Must sample at least at 2× the highest frequency we want to represent
  - Because you need at least two samples to represent a period
- Perceptual limits
  - Hearing: we hear up to 20kHz, hence need a sample rate of 40kHz and up
  - Seeing: we cognitively perceive up to 60Hz, suggesting a sample rate of 120Hz and up
- Common sample rates
  - Speech 8kHz to 16kHz, Music: 32kHz to 44.1kHz, Pro-audio: 96kHz
  - Movies: 24fps (recently 48fps), TV: 30fps, HDTV: 60fps (high end up to 600fps!)

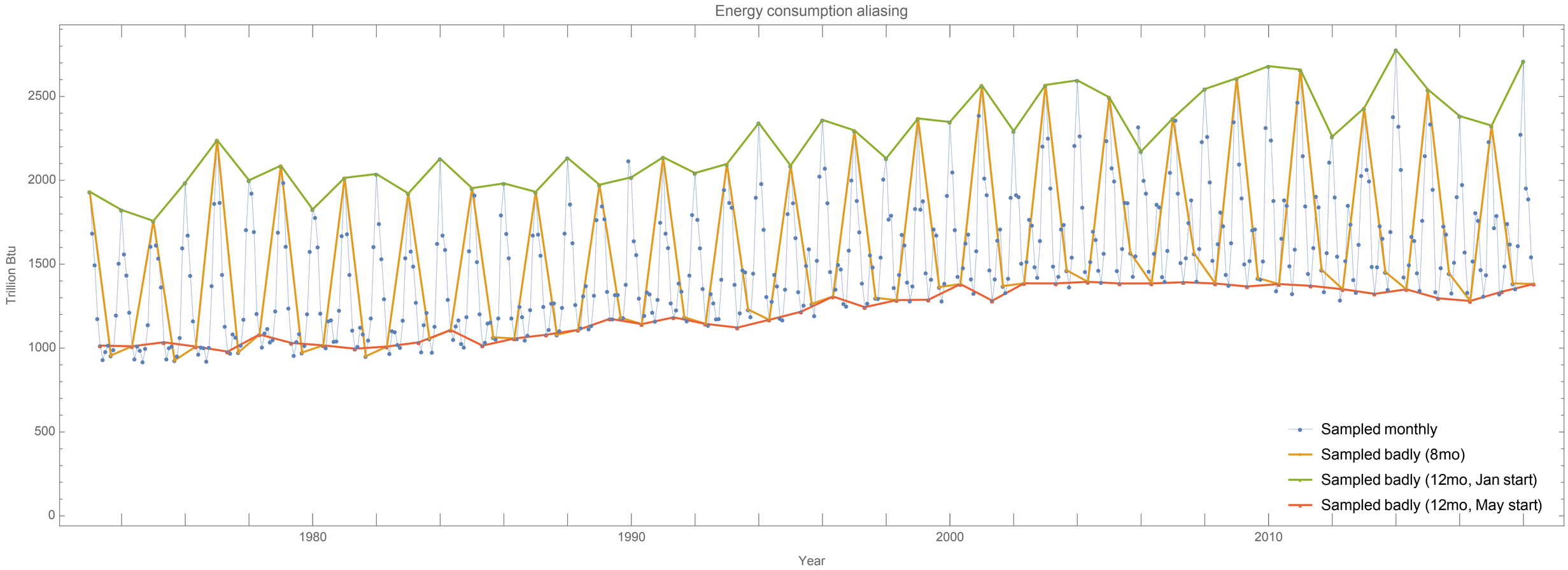
Aliasing

- Low sample rates result in *aliasing*
  - Frequencies above half of the sampling rate are misrepresented



In real-life

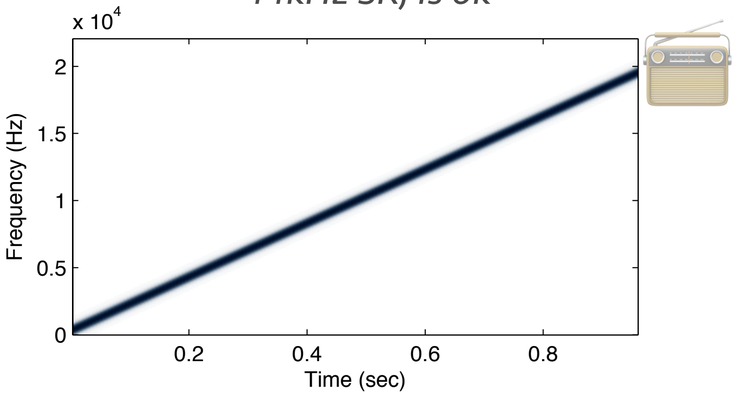
- Electricity consumption sampled poorly
  - Different sampling makes for different conclusions



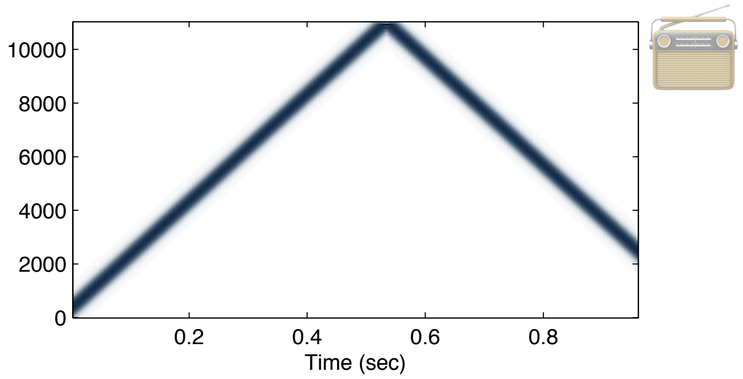
Aliasing examples

Sinusoid sweeping from 0Hz to 20kHz

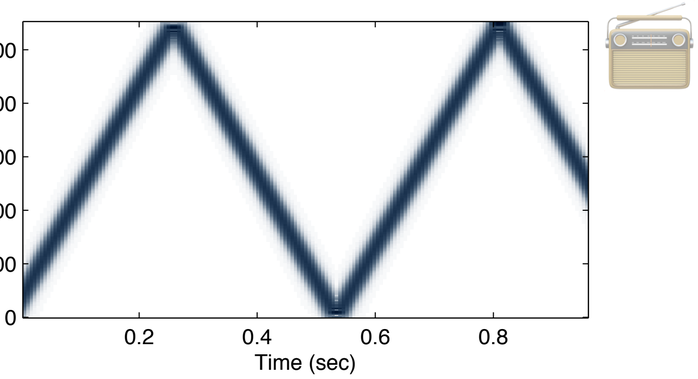
44kHz SR, is ok



22kHz SR, aliasing!



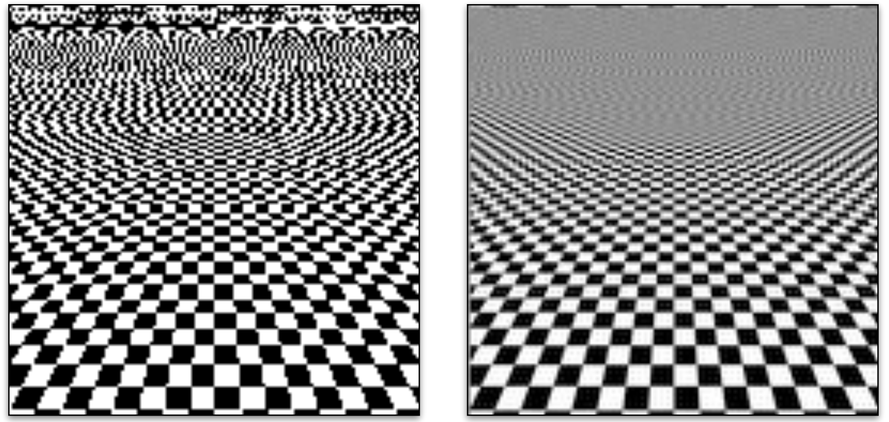
11kHz SR, double aliasing!



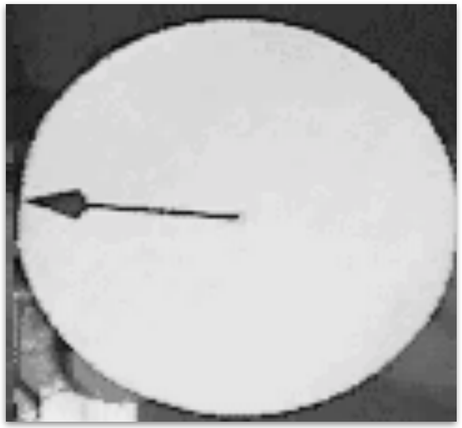
Music example

Original *11 kHz* *5 kHz* *4 kHz* *3 kHz*

In images



In video



Extreme video aliasing!

https://www.youtube.com/watch?v=R-IVw8OKjvQ

Convolution

- Operation between two time series
  - Shift and scale one of the time series according to the other one
- Formal definition:

$$z = y * x$$

$$z(t) = y(0) \cdot x(t) + y(1) \cdot x(t-1) + \cdots$$

$$z(t) = \sum_i y(i) \cdot x(t-i)$$

- Commutative operation
  - $x*y = y*x$
- Resulting output is sum of lengths minus one

Convolution Examples

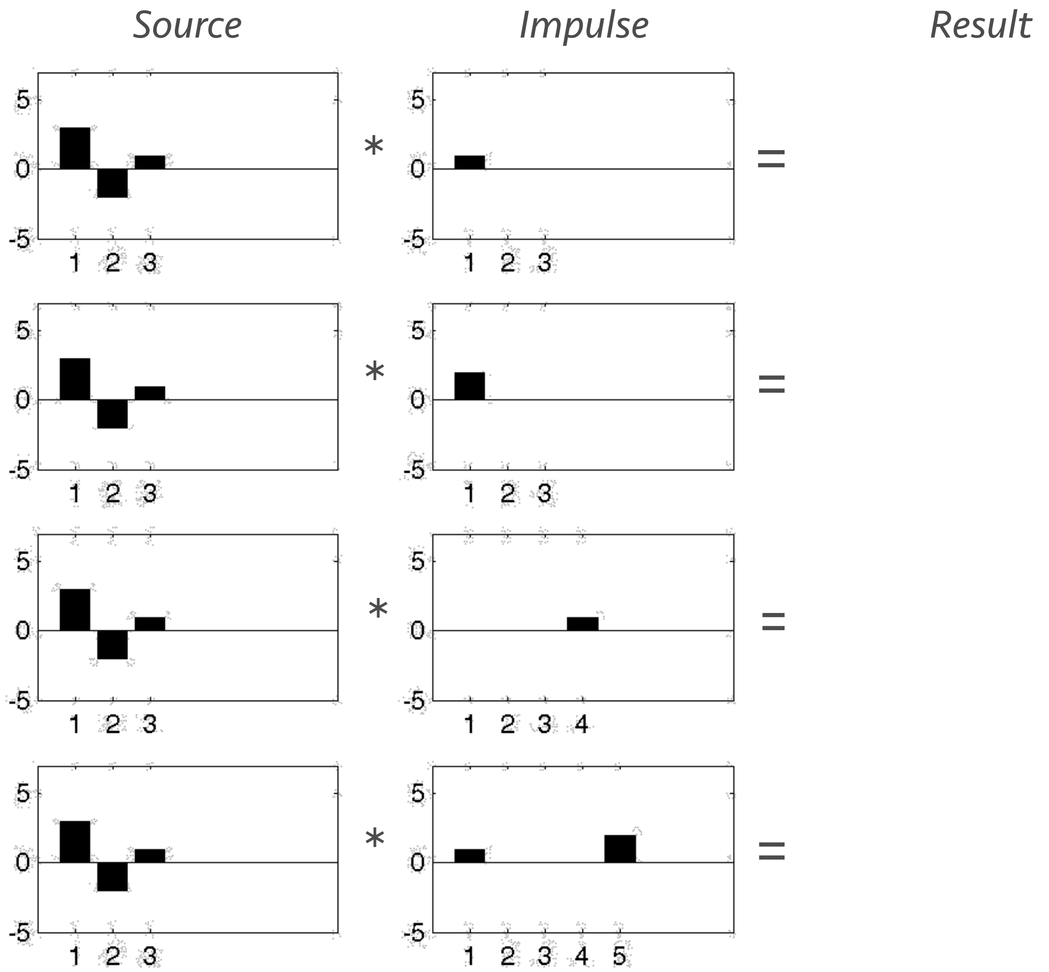
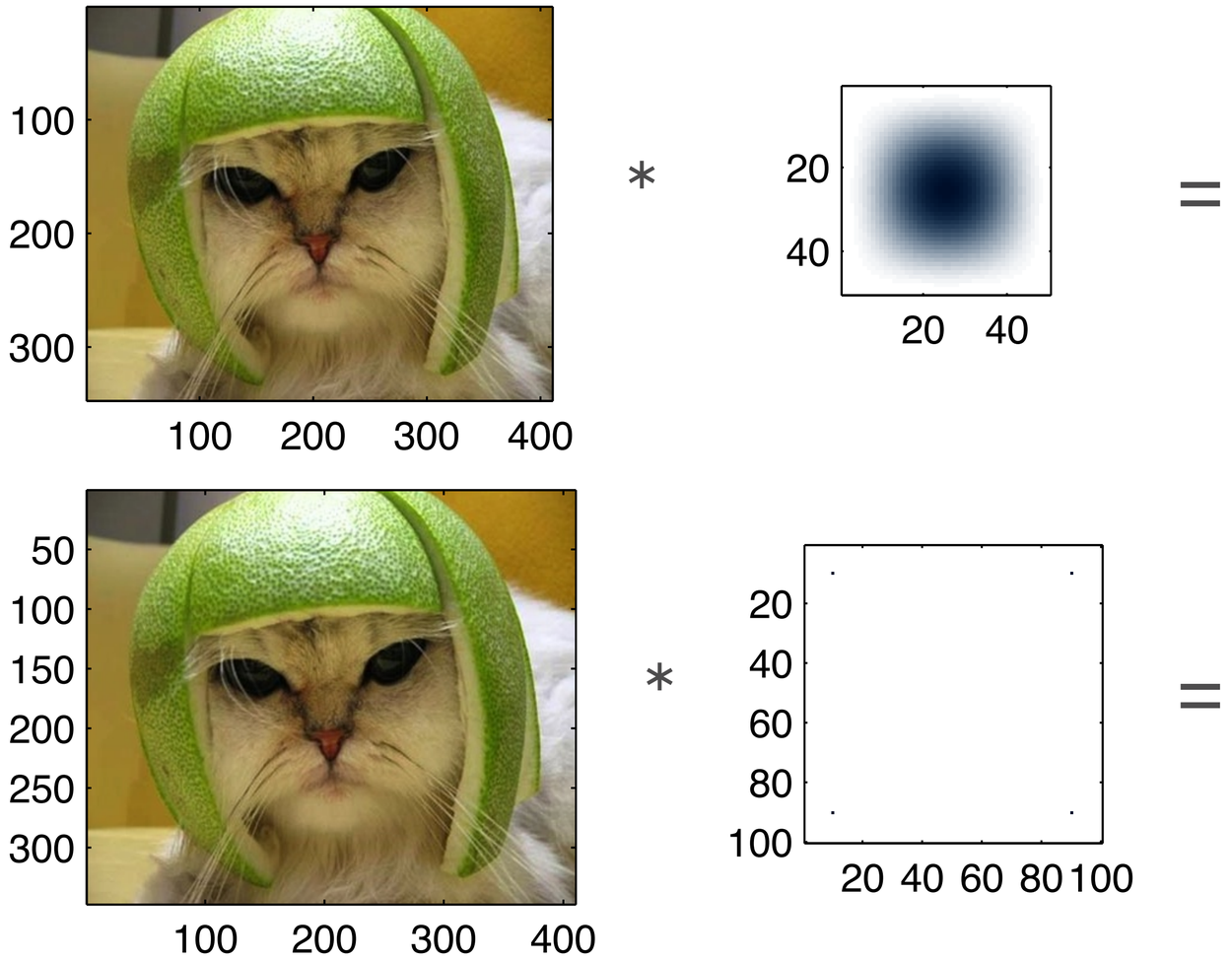


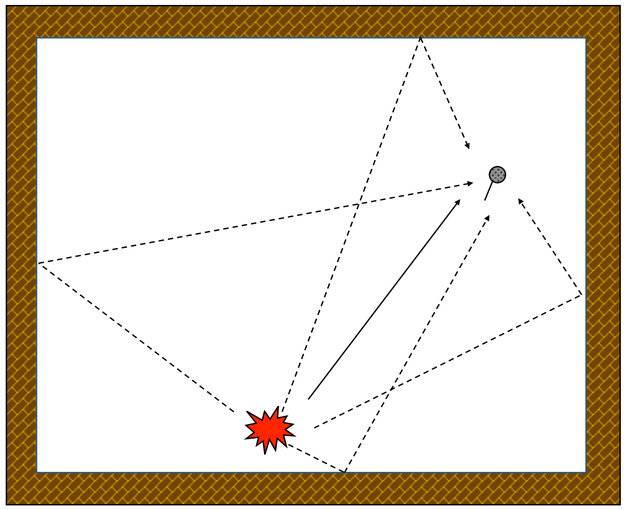
Image convolution

Input *Convolved with* *Results in*

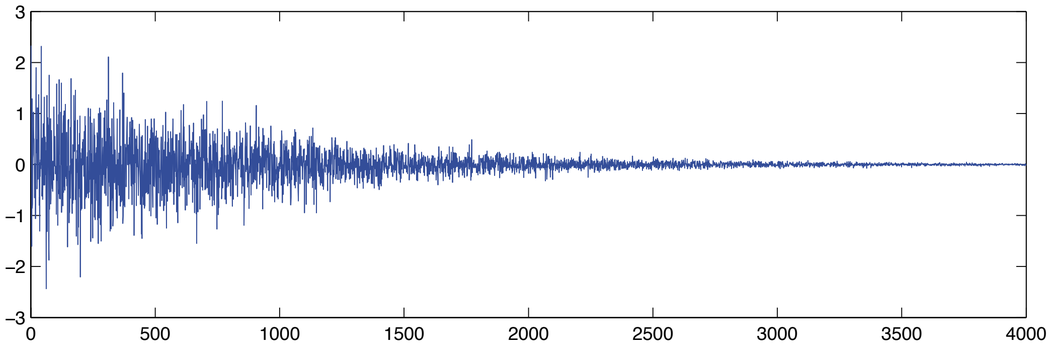


Fun with convolution: Rooms

- Sound bounces on walls
  - Each bounce creates an echo which repeats the same sound delayed
  - Effectively this is a filter!
- We can emulate that with convolution!
  - Make an impulse response that has lots of random echoes at varying levels
  - Also make it decay a bit over time to make more realistic
- Cheap way to make "cathedral emulations"!

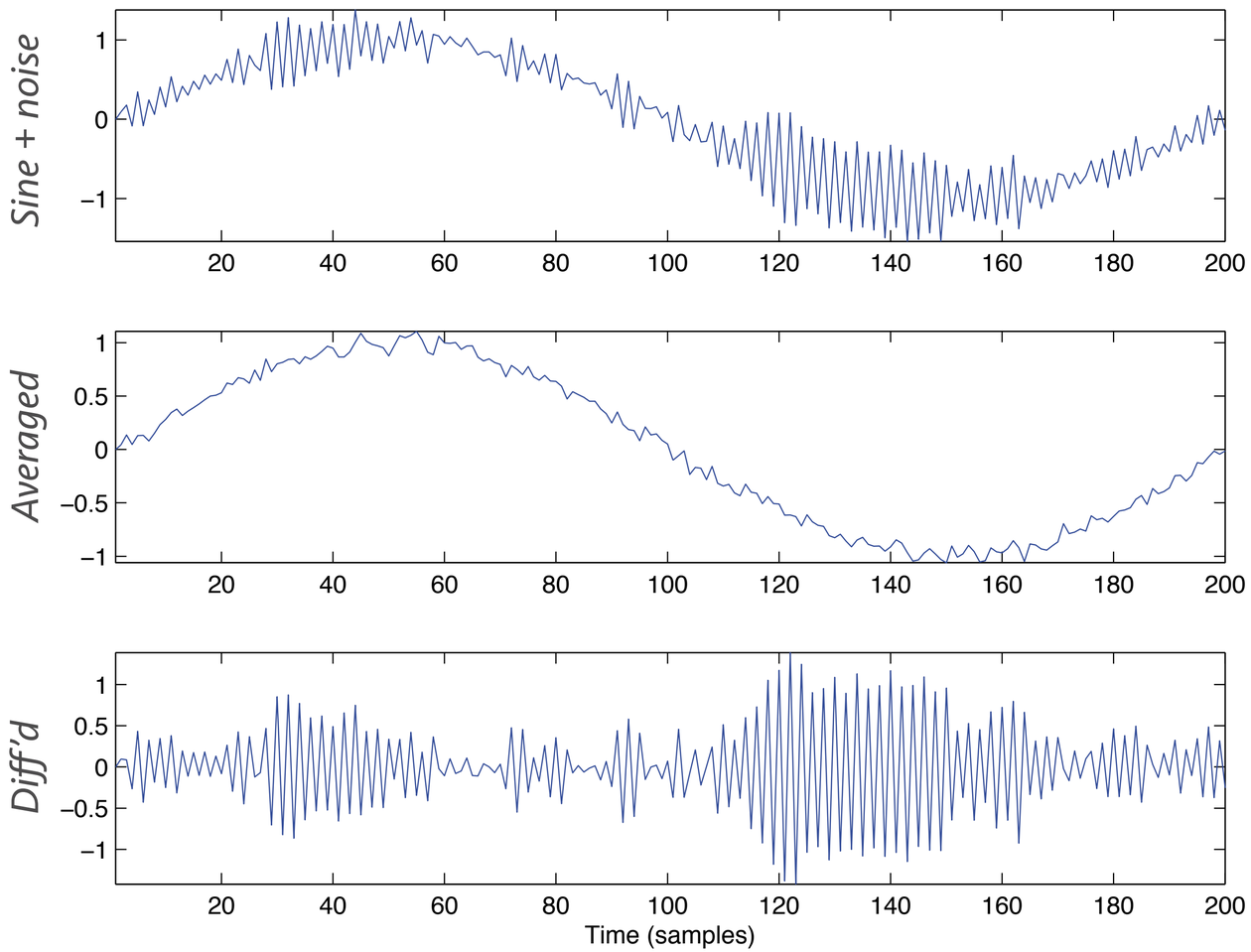


Noise masquerading as a room impulse response



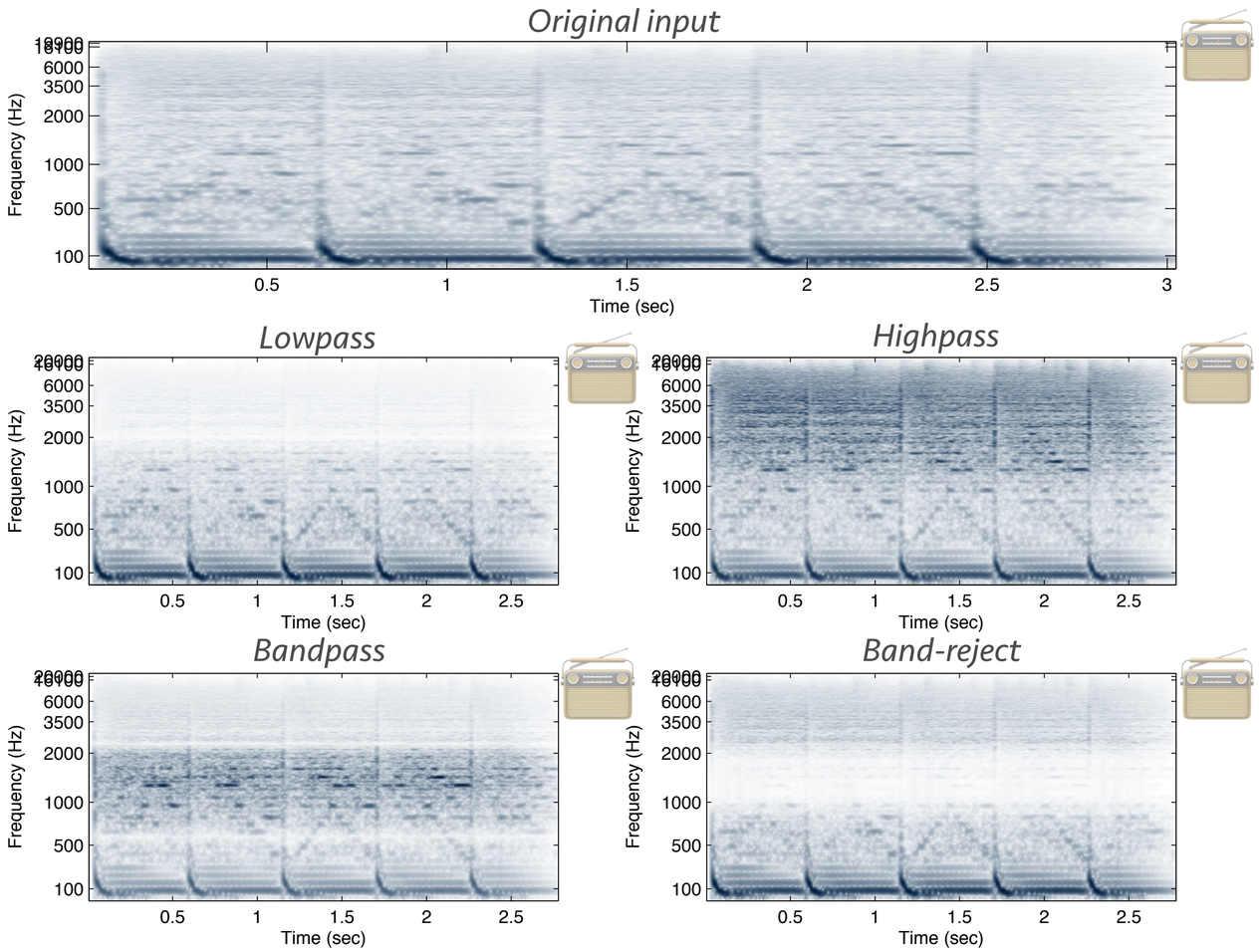
One more convolution thing

- Example contains a sine and noise
  - The sine is low frequency, the noise is high
- Averaging
  - Convolving with [½, ½]
  - Smoothes input, removes high frequencies
- Differentiating
  - Convolving with [1, –1]
  - Exaggerates noise, removes low frequencies
- Can we generalize?
  - Indeed we can, using *filters*

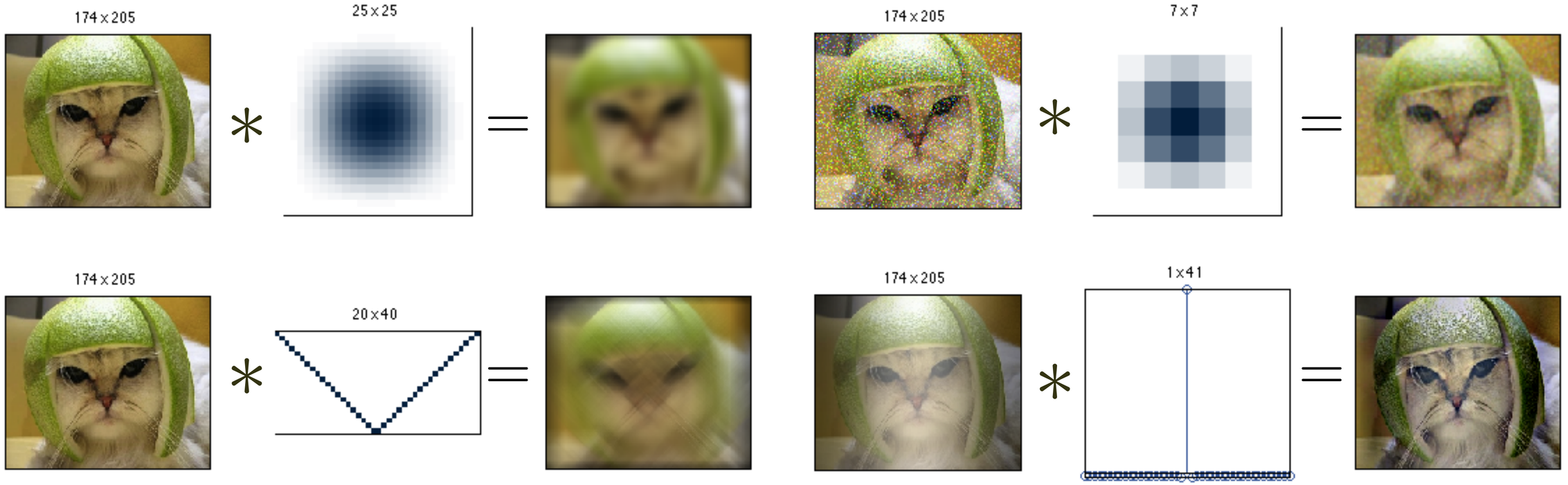


Filter types by function

- Lowpass
  - Allows only low frequencies through
- Highpass
  - Allows only high frequencies through
- Bandpass
  - Allows only a band of frequencies through
- Band-reject/stop
  - Allows only band of frequencies
- Custom
  - Arbitrary boosting/suppression

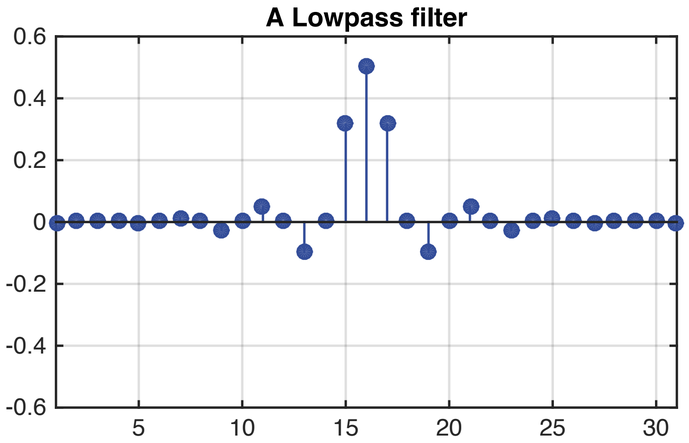


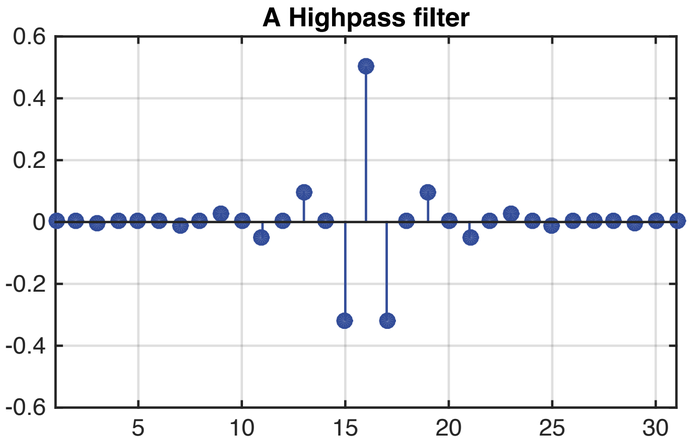
Filtering images

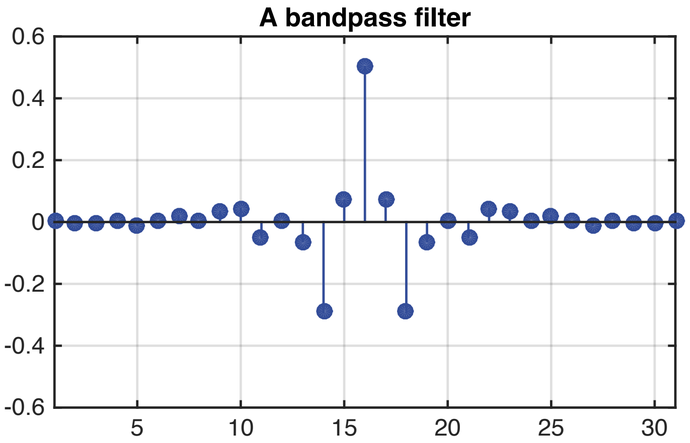


How do filters look like?

- Simple time series with specific structure



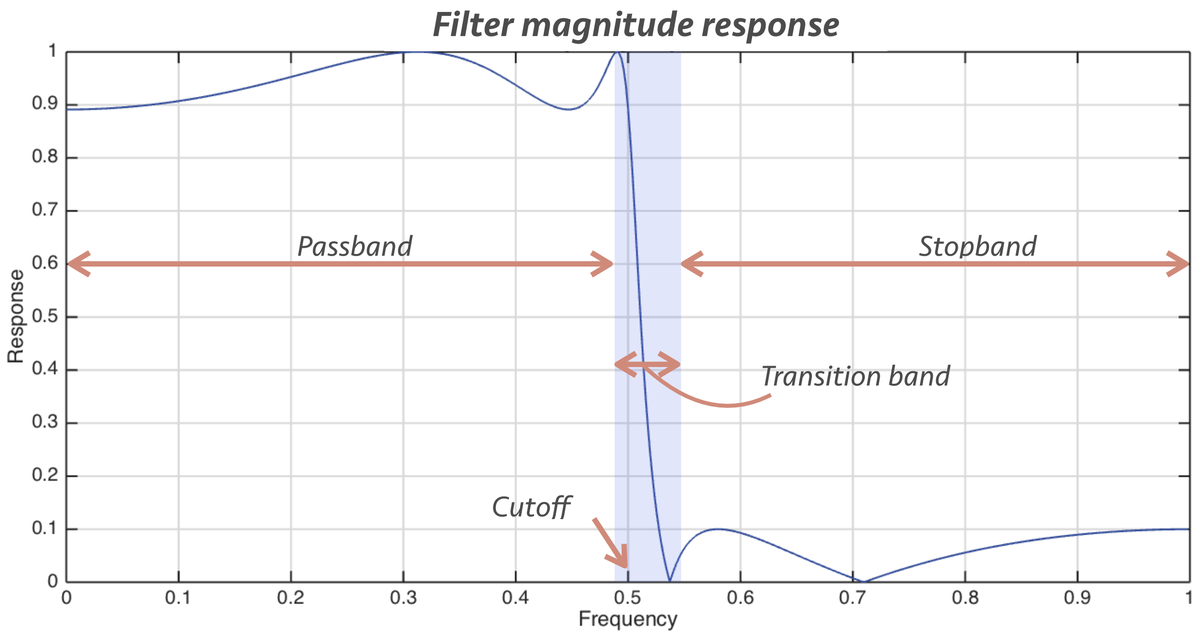




- How do we design them?
  - Many, many ways
    - Go take a DSP course!

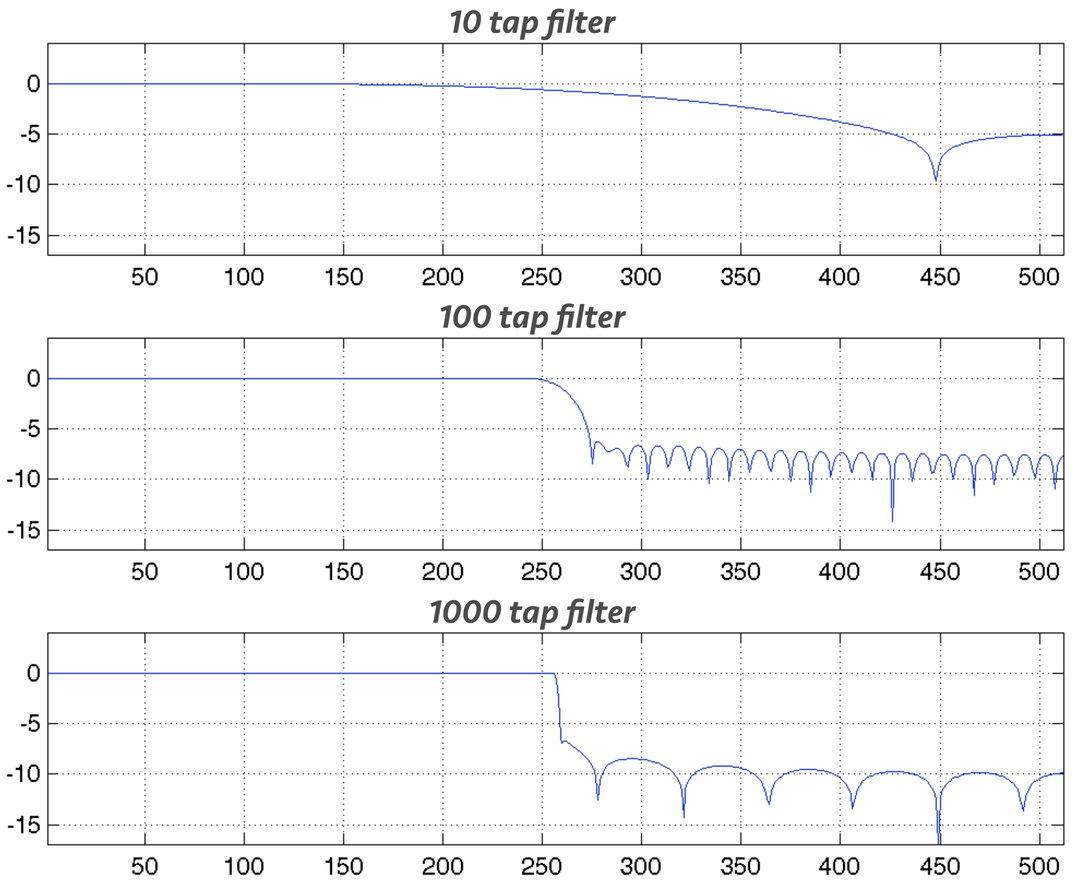
Filter nomenclature

- Filters alter frequency content
  - We can measure this effect by taking a filter's Fourier transform!
- This is the *filter frequency response*
  - Allows us to judge the quality and effectiveness of a filter we convolve with
- For simple filters we have
  - The *passband* – what goes through
  - The *stopband* – what is filtered out
  - The *transition band* – the in-between



Filtering trade-offs, Length

- Length vs. Strength
  - The longer a filter is the better it can approximate the desired frequency response
- Length vs. Efficiency
  - The longer the filter is the more time it takes to compute it



Fast convolution

- Convolution can be quite slow
  - FFT to the rescue!
- Fourier domain multiplication is time domain convolution
  - And vice versa:

$$F(x * y) = F(x) \odot F(y)$$

$$F(x \odot y) = F(x) * F(y)$$

$$x * y = F^{-1}\Big(F(x) \odot F(y)\Big)$$

- Using the FFT we get immense speedups
  - The catch is that the latter half of the inputs has to be all zeros

Convolving with matrices

- Convolutions are linear transforms
  - We can implement convolution as a matrix product

$$\mathbf{a} * \mathbf{x} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \mathbf{C} \cdot \mathbf{x} = \begin{bmatrix} a_2 & a_1 & 0 & 0 & 0 \\ 0 & a_2 & a_1 & 0 & 0 \\ 0 & 0 & a_2 & a_1 & 0 \\ 0 & 0 & 0 & a_2 & a_1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} a_2 x_1 + a_1 x_2 \\ a_2 x_2 + a_1 x_3 \\ a_2 x_3 + a_1 x_4 \\ a_2 x_4 + a_1 x_5 \end{bmatrix} \Rightarrow x_i = \sum_k x_{i-k} a_k$$

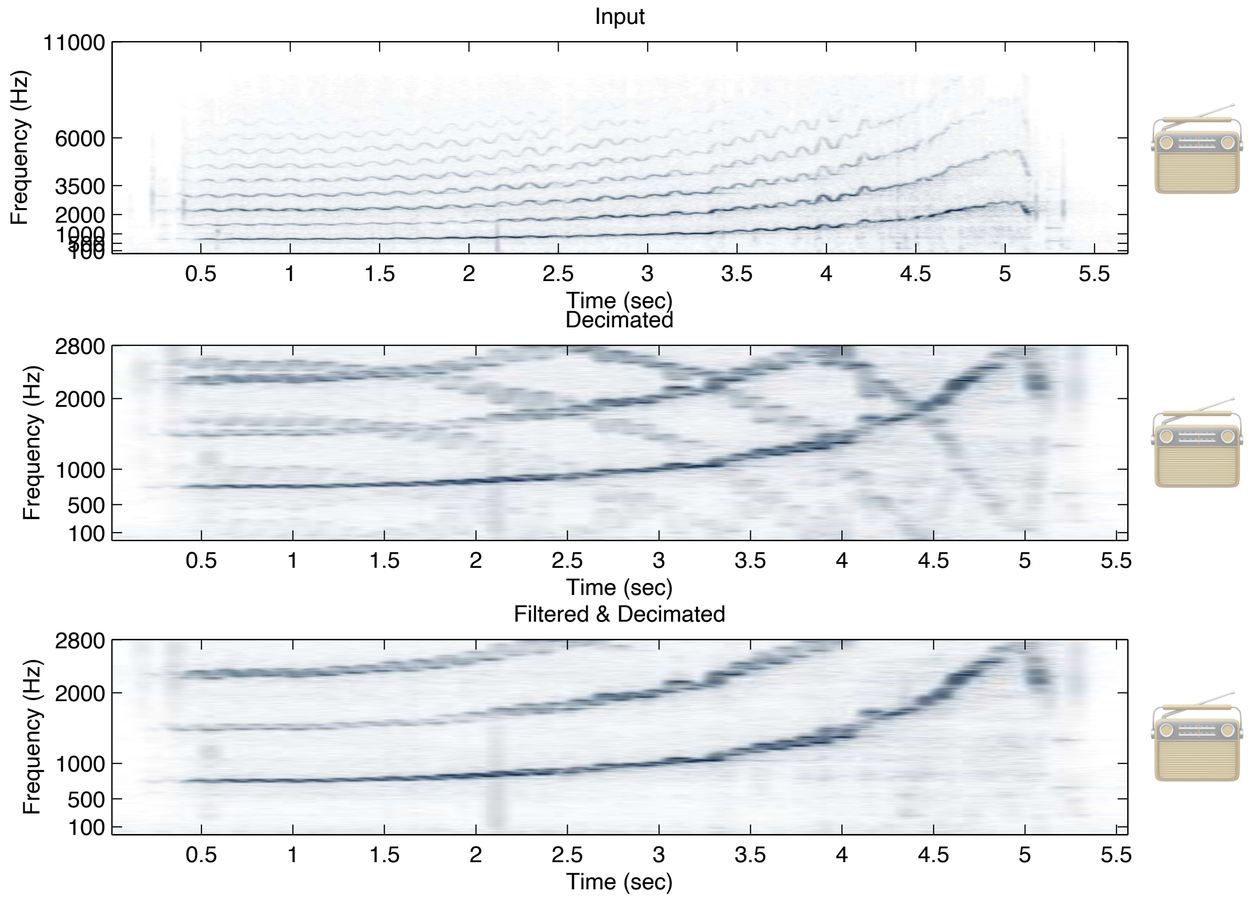
  - We can adjust $\mathbf{C}$ to deal with the necessary zero padding

Resampling

- Changing a signal's sampling rate
  - Usually to accommodate different playback hardware
    - E.g. CD vs. DAT audio, or 4k vs. 720p videos
  - Also useful to reduce large data sets
- Multirate processing
  - Upsampling, downsampling
- Proper resampling bypasses aliasing
  - Remember that we need to sample correctly!

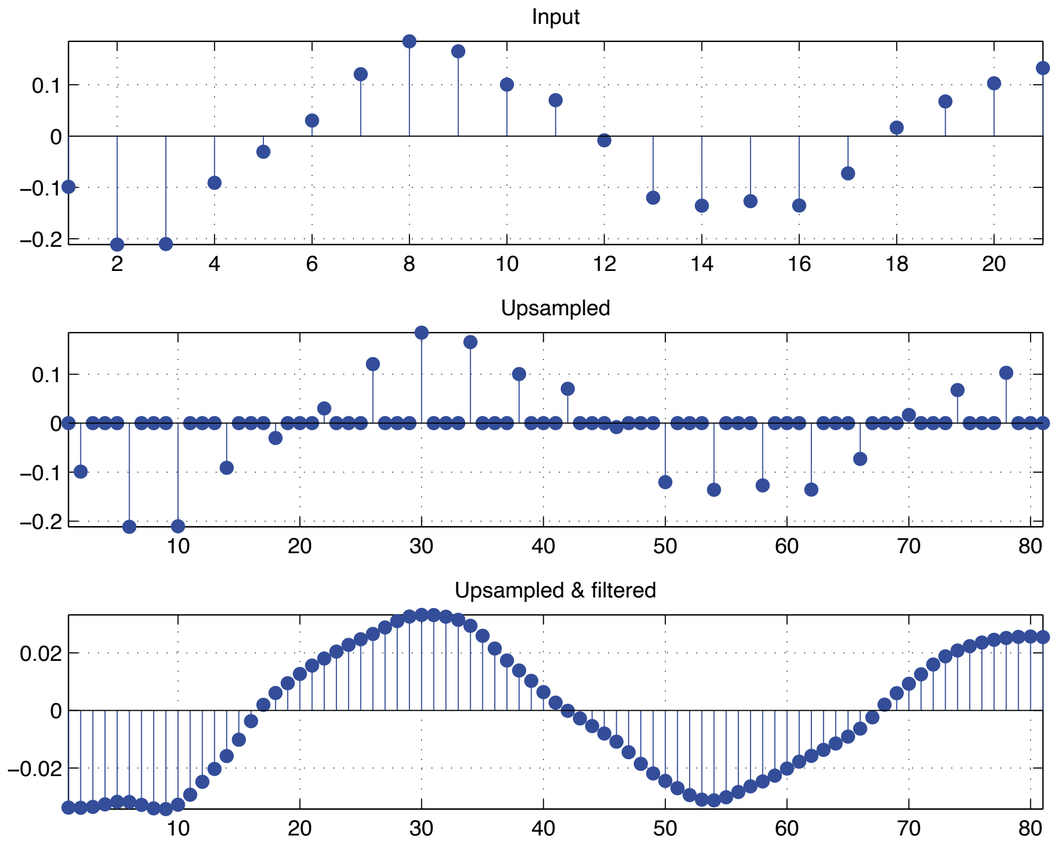
Downsampling

- Lowering the sampling rate
- Downsampling by 2
  - Picking every other sample will alias!
  - We must first remove the high frequencies first and then pick samples
- For downsampling by M
  - Lowpass filter to $1/M$
  - Pick one out of M samples
    - Only works for integer M
- High frequencies will be lost!

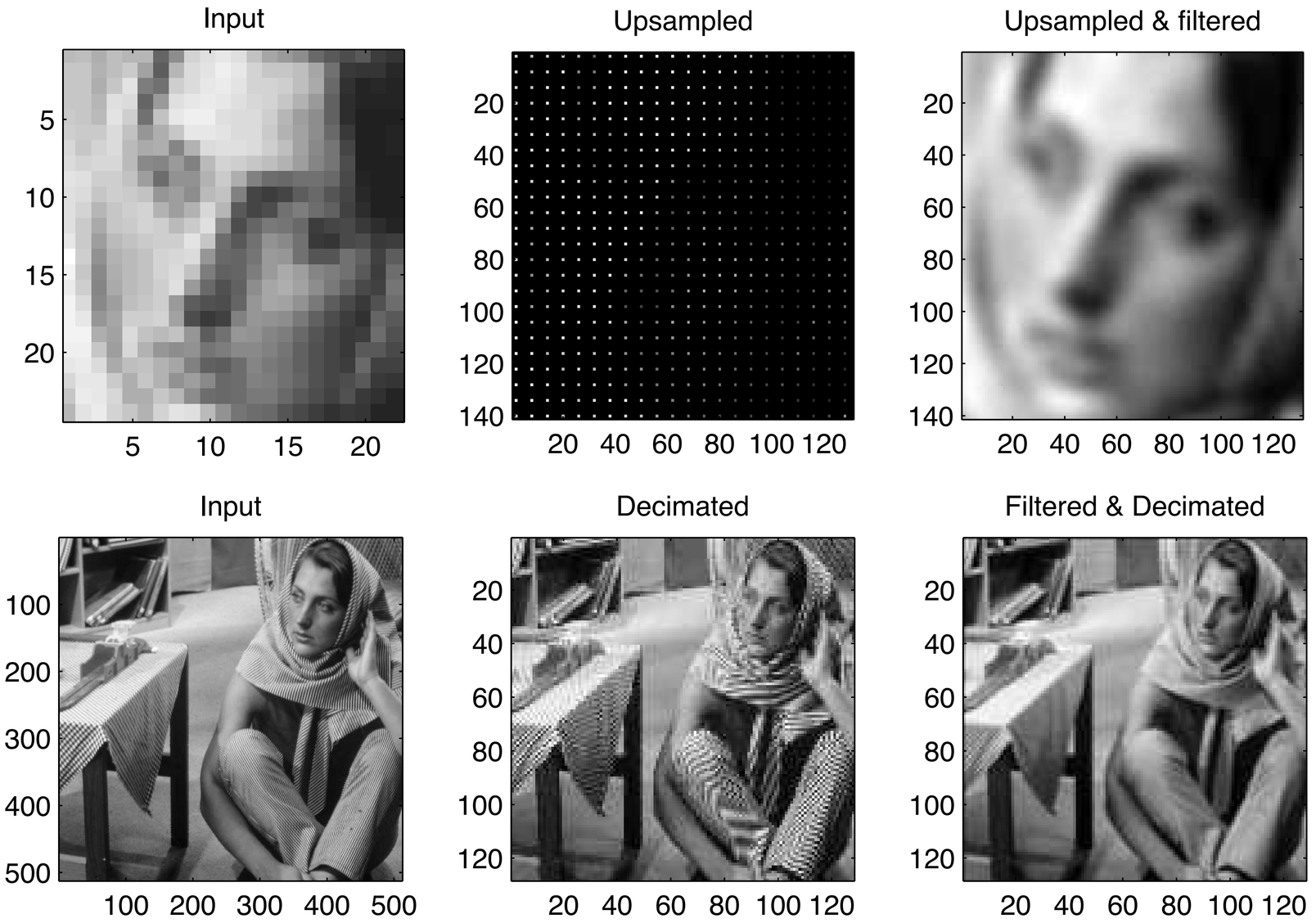


Upsampling

- Increasing the sample rate
- For upsampling by L
  - Put $L–1$ zeros between each sample
  - Lowpass filter to $1/L$
  - Only works for integer L
- Without filtering we get noisy clicks
- No loss of information
- Upsampling does not recover high frequencies!

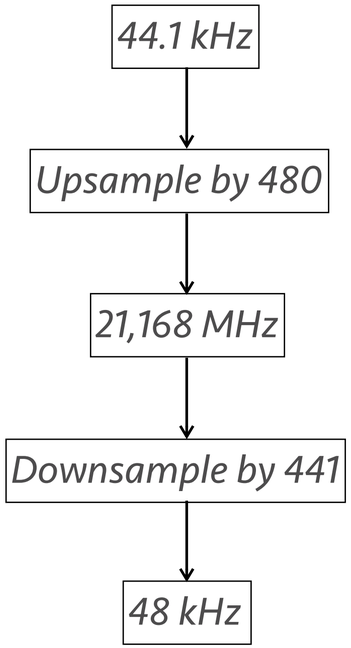


Ditto for images



What if the ratio is not integer?

- Describe it using two integer resampling operations
  - E.g. to go from 44.1kHz to 48kHz
    - Need to upsample by 1.0884
  - This is 480/441
  - We can upsample by 480 and downsample by 441
- Efficient algorithms exist
  - There are ways to avoid explicitly upsampling by 480
    - Reduces storage requirements
- That's why CDs are 44.1kHz and DATs were 48kHz, to make resampling a pain



Recap

- Time-series
- Time/frequency domain
  - Good for visualizing periodic information
- Convolution
  - Various uses
- Filters
  - low/high/band-pass/stop
- Sampling
  - Aliasing, up/down-sampling, resampling

Further reading

- Wikipedia.org
  - Search any of the terms used so far to get a nice and easy to understand overview
- Mathworld.com
  - Search for cross-correlation/convolution to find out more
- Dspguru.com
  - Nice resource, code, books, tutorials
- http://www.mathworks.com/access/helpdesk/help/toolbox/signal/
  - MATLAB's signal processing toolbox manual page
  - Lots of detailed information about both SP and MATLAB
- http://www.dspguide.com/pdfbook.htm
  - Nice and free overview book
  - Very detailed

Next Lecture

- No more background material!
- Fixed features for signals
  - How we perceive signals
  - How that helps us make machines that do so