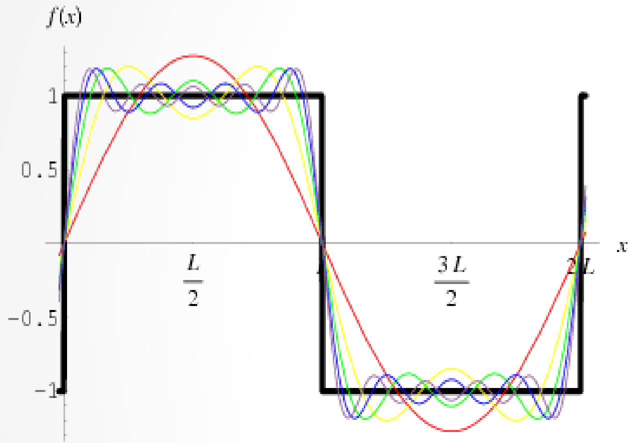
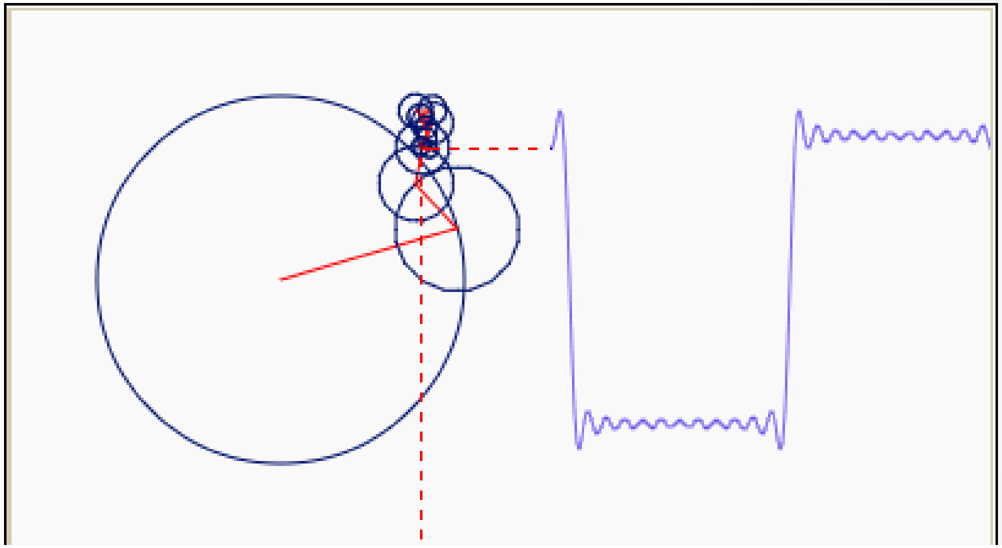
Lecture 2: sampling, filters, convolution, digital formats

luke@sjulsonlab.org

Signals can be decomposed into sinusoids





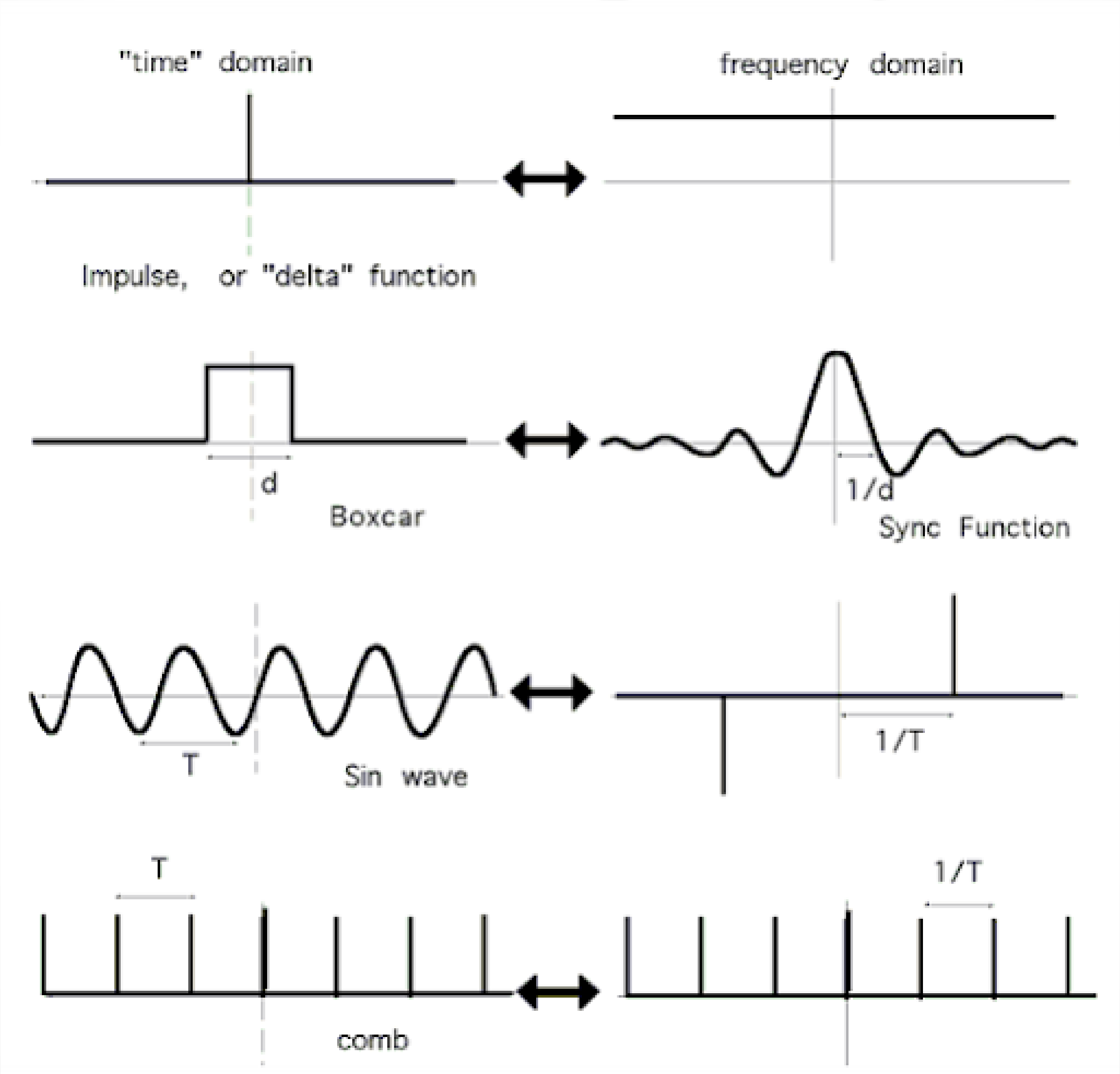
The Fourier transform converts from time to frequency domain

Sometimes called Dirac delta function, δ(t)

Note that the delta function contains all frequencies

T = wavelength

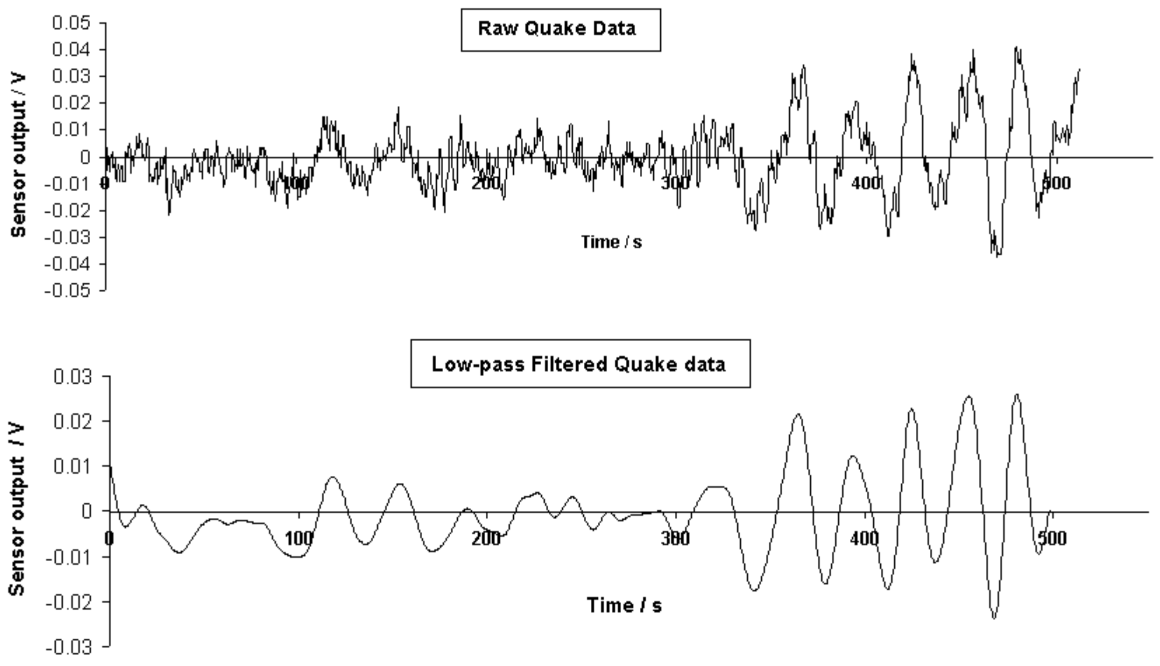
1/T = frequency



Filters and transfer functions

- Filters (mostly) block certain frequencies and let others pass through
- Draw transfer functions by hand…

Filtering examples



- Draw example of high-pass filtering

Convolution: the time domain version of filtering

Convolution Theorem

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

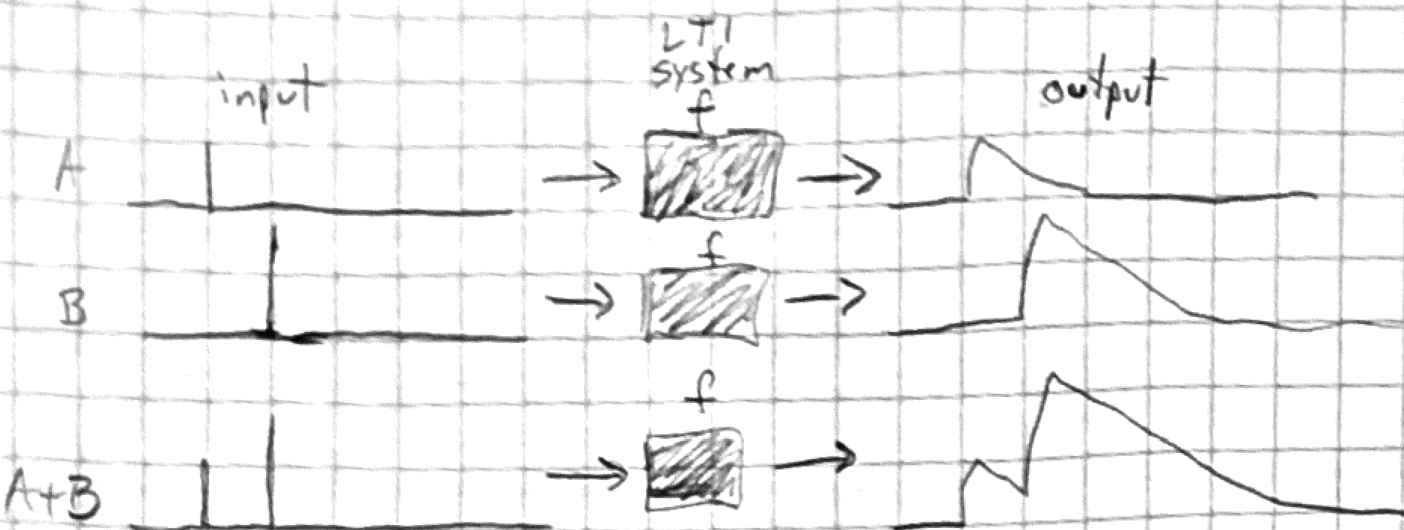
$$f \otimes g \leftrightarrow F \times G$$

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

$$f \times g \leftrightarrow F \otimes G$$

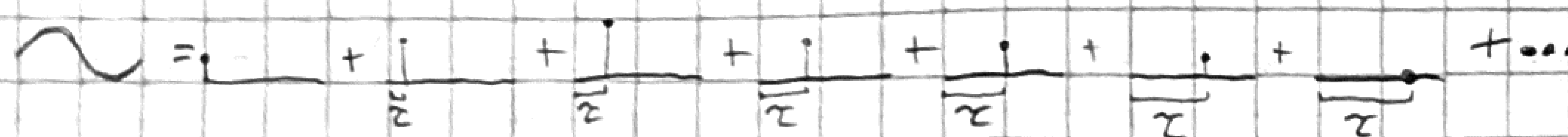
The output of Linear Time Invariant (LTI) systems is the convolution of the input with the impulse response function

LTI systems



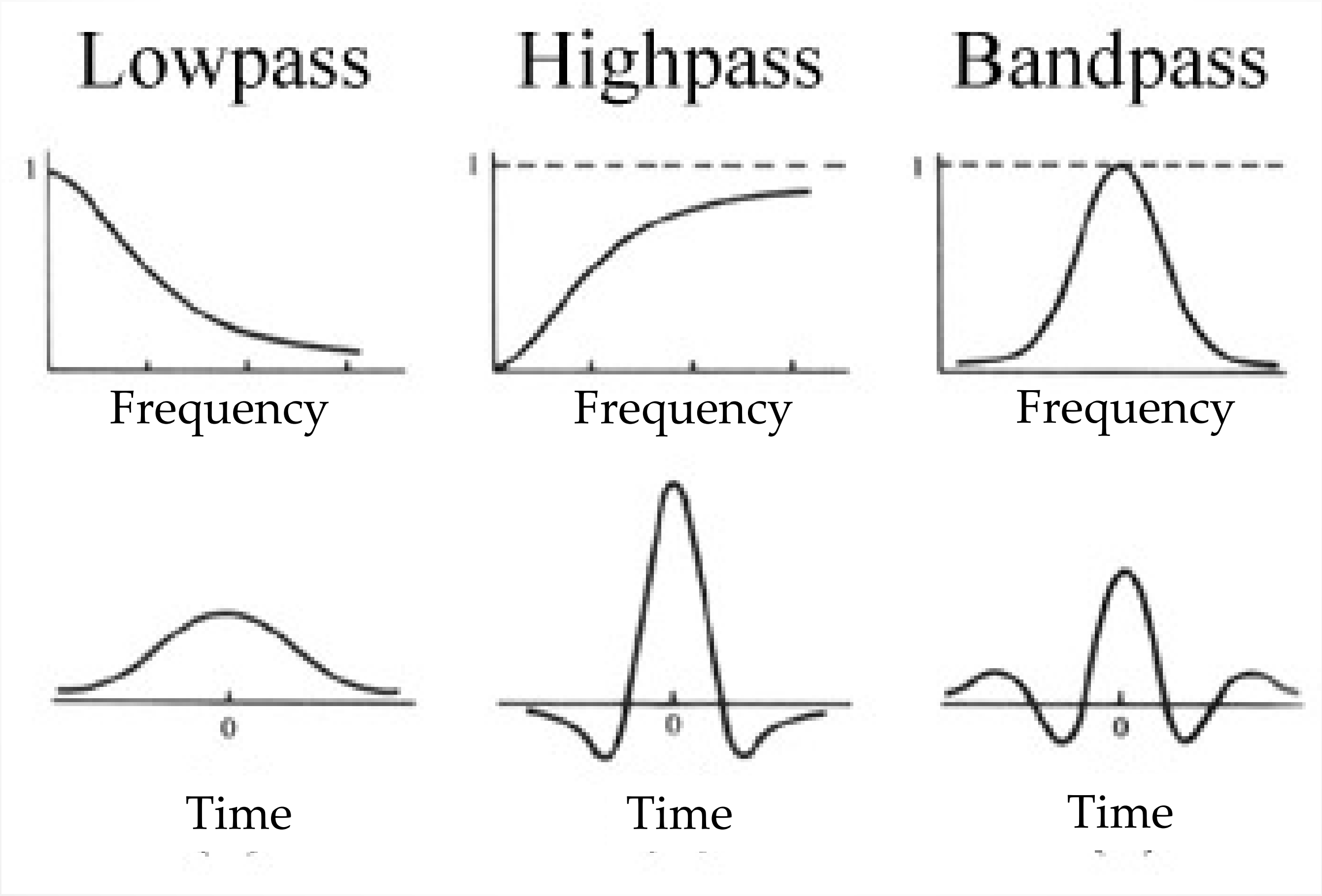
A and B don't interact $\qquad f(A)+f(B)=f(A+B)$

This means any input can be broken into parts. If we know the impulse response function, we know what the output for any input will be.



An LTI system CONVOLVES its input with its impulse response function.

Filter transfer functions (freq domain) are Fourier transforms of their convolution kernels (time domain)



Harmonics and notch filters

- Draw by hand

Filtering: important points

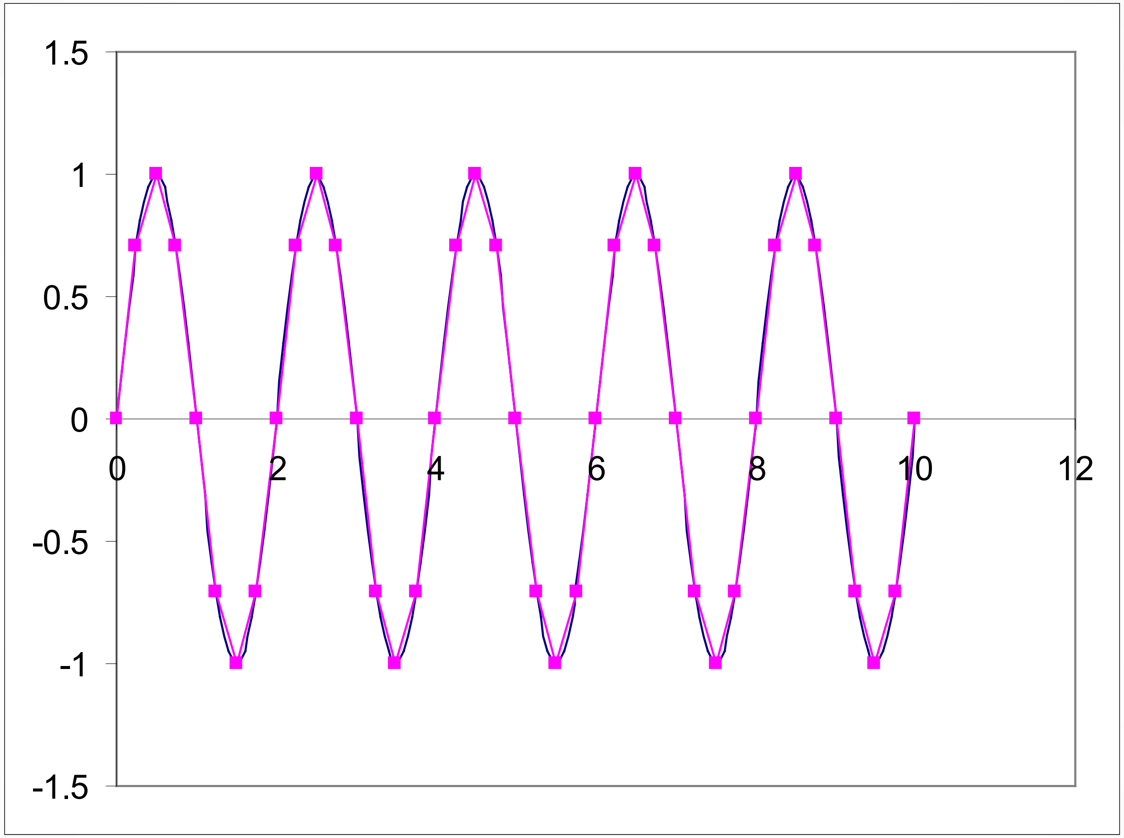
- Filtering removes certain frequencies
- Multiplication in the frequency domain (e.g. filtering) equals convolution in the time domain and vice versa
- Filtering (in freq domain) corresponds to convolution (in time domain) with a filter "kernel"
- Conversely, convolution (time domain) can be conceptualized as filtering (freq domain)
- Distortions of an underlying signal by instrumentation, ChR2, or GCaMP are convolutions*
- ChR2 and GCaMP act as low-pass filters

* if they are operating in a linear range – see lecture 5

Sampling

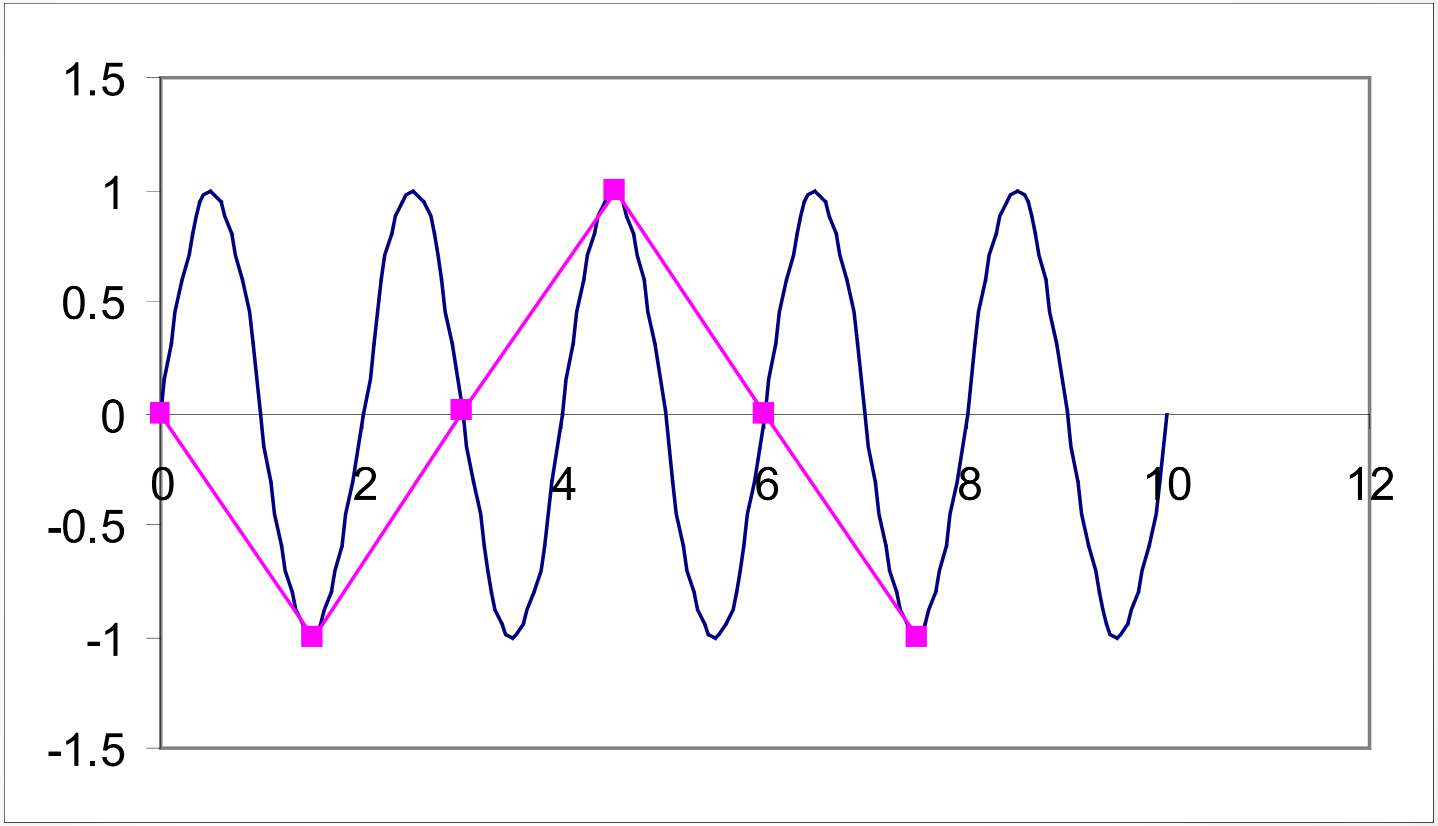
- How do you configure your filter settings and sampling rate correctly?

Good Sampling



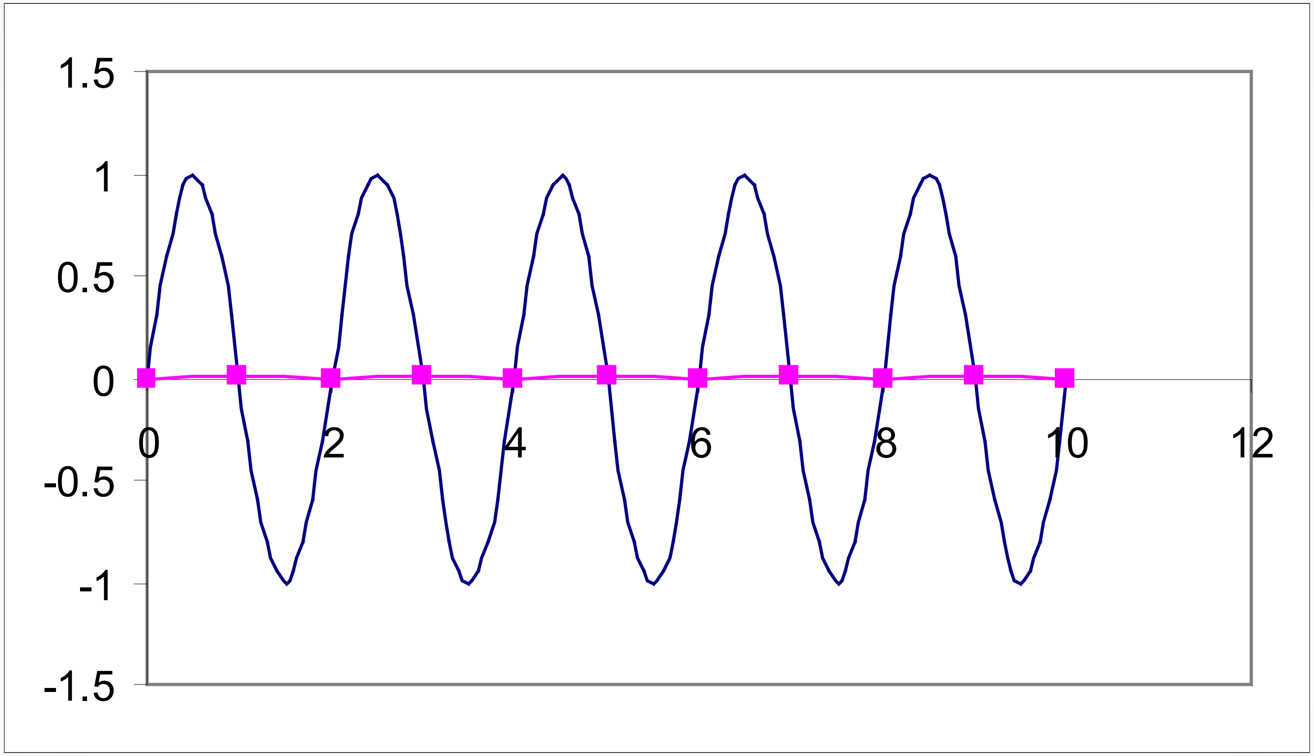
Sampling rate = 8X frequency

Bad sampling



Sampling rate = 1.5x frequency

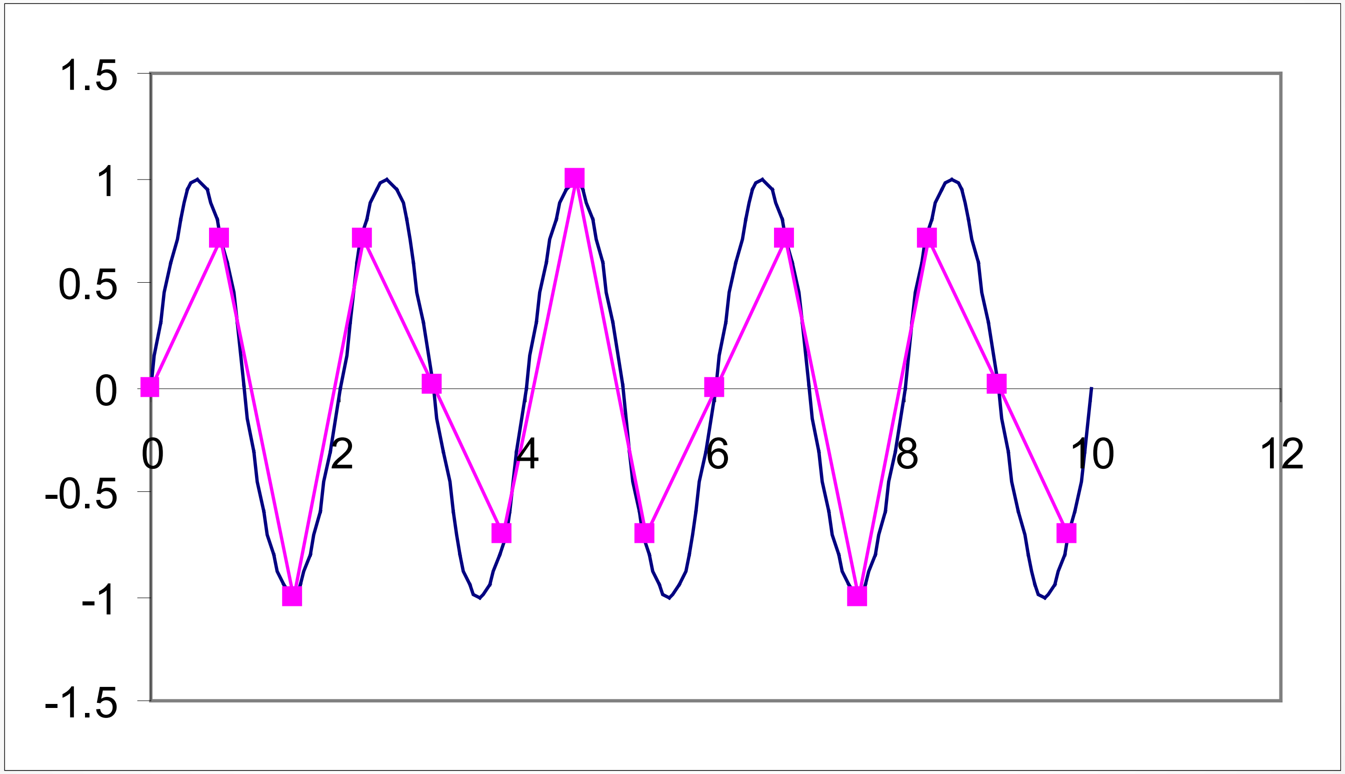
Still bad



Sampling rate = 2X frequency

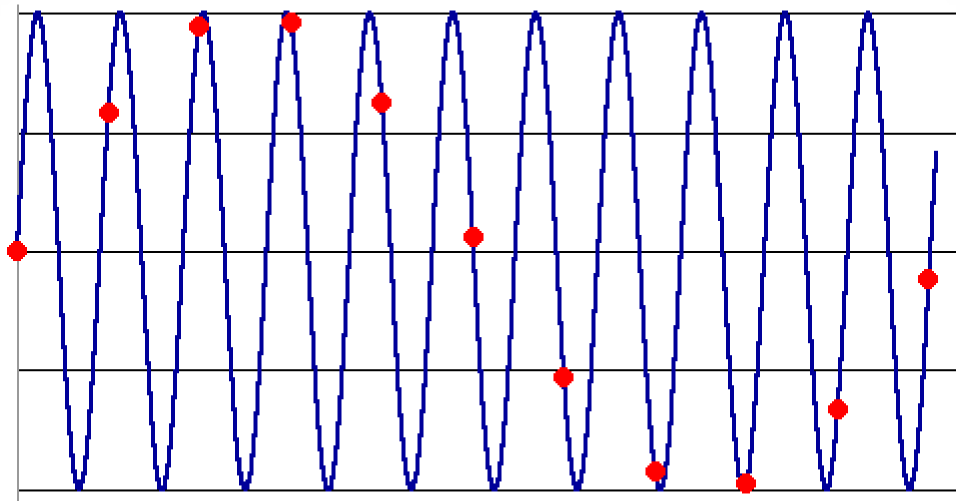
Nyquist's theorem: to capture a frequency accurately, you must sample more than 2X that frequency

Adequate sampling



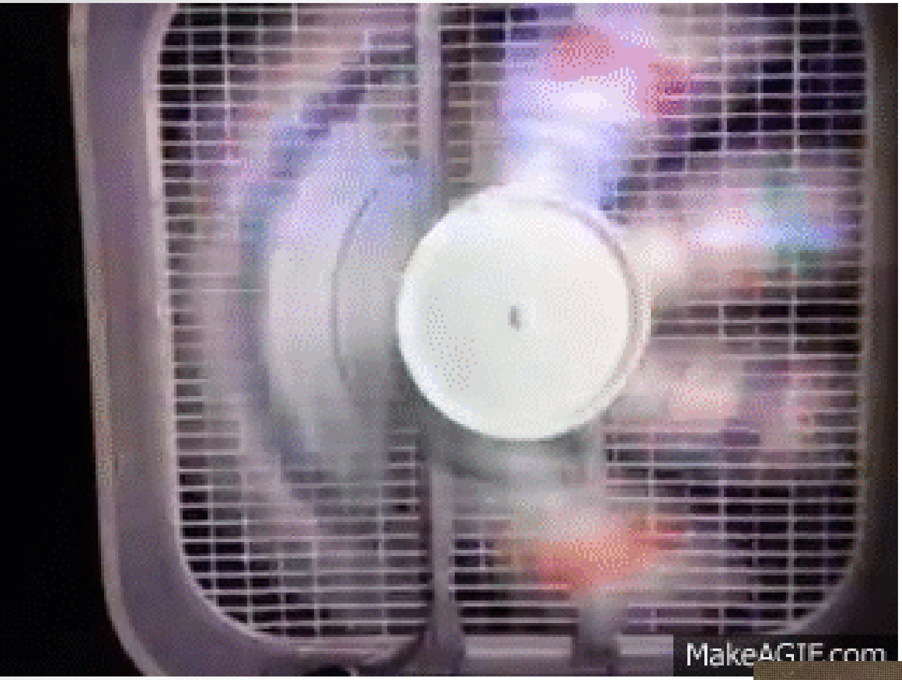
Sampling rate = 3X frequency

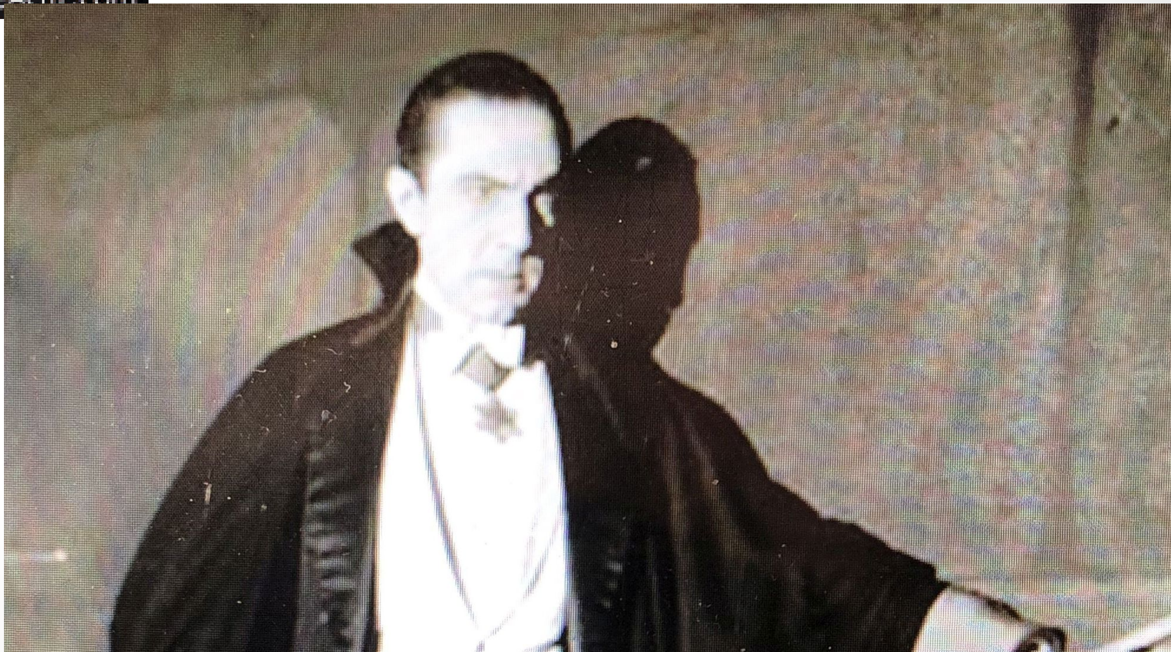
Why is undersampling bad? Aliasing



- The blue is the true underlying signal
- The red dots are the samples, taken at too low of a sampling rate
- This "aliases" the blue signal, which has a frequency more than 2X the sampling rate, into an artifactual signal with a frequency less than 2X the sampling rate

More aliasing





How to avoid aliasing

- Low-pass filter (in hardware) prior to sampling (A to D conversion)
- Action potentials have frequency content up to ~8 kHz. What frequency do we need to sample at?
- Most of what is above 8 kHz is random thermal noise, so you aren't so much aliasing a specific sine wave at a high frequency to one at a low frequency as you are adding broadband noise to your signal
- A notable exception to this is periodic noise from switching mode power supplies (e.g. an iPhone charger), which sometimes switch at 10-30 kHz

Sampling: important points

- Physiological signals only have frequency content in certain ranges
- We should use low-pass filtering to remove everything above that range because it is noise
- To prevent aliasing, we must sample at AT LEAST 2x higher frequency than the filter cutoff. In practice, use 3x.
- We can use notch filtering to remove 60 Hz line noise and its harmonics, but that is problematic if our signal has power in those frequencies (e.g. local field potentials or EEGs)

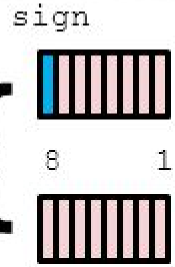
Variable types and digital formats

- What does your data look like once it's inside the computer?
    - Good question.

Integer Number Representations

conversion functions `intmin, intmax`

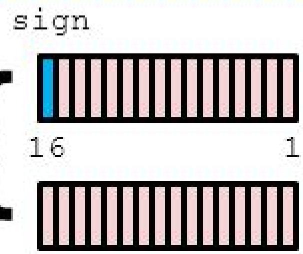
int8
8-Bit Integer
uint8



$[-2^7, +2^7-1] = [-128, +127]$

$[0, +2^8-1] = [0, +255]$

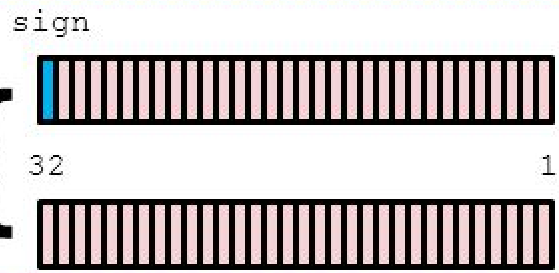
int16
16-Bit Integer
uint16



[-32,768 +32,767]

[0 65,535]

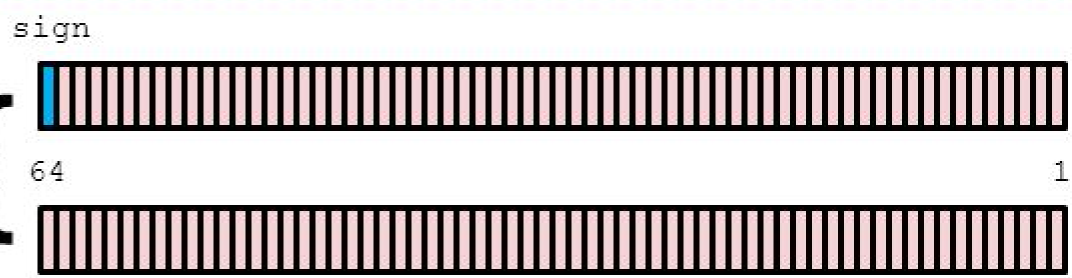
int32
32-Bit Integer
uint32



$[-2^{31}, +2^{31}-1]$

$[0, +2^{32}-1]$

int64
64-Bit Integer
uint64



Fixed-point numbers

- Good:
  - Simple, exact representation

- Bad:
  - Range is too small!
  - Only integers

Integer Issues

This is false – matlab will either convert it to Inf or implicitly convert it to type double

- **Overflow,** expression tries to create an integer value larger than allowed valid range `[min,max]`
  - `x = int8( 127 ) + 1`
  - ***Saturate Arithmetic* (MATLAB)**
    - value clamped to `min, max` range (`x = 127`)
  - ***Wrapping Arithmetic* (Most languages)**
    - wraps back around to other end of range (`x = -128`)
- **Truncation**, fractions not supported
  - `int16(1)/int16(4) = 0` **not** `0.25`
  - Rounds result to nearest whole number

Floating-point numbers

- Like scientific notation for binary

  twenty-five = $2.5 * 10^1$

  twenty-five = 11001_2

  $= 1.1001_2 * 2^4$

- In general:
  - n = sign * mantissa * $2^{exponent}$

- Good:
  - Can represent non-integral numbers
    - $-2.5 = -1 * 1.25 * 2^1$
  - And very large numbers
    - $10^{100} = 1 * 1.142987... * 2^{332}$

The data is typically acquired as ints. However, any math beyond addition and subtraction requires the data to be in a float format

CPUs do both integer and floating-point math. GPUs are specialized to do only certain types of floating-point operations, and they do it much faster.

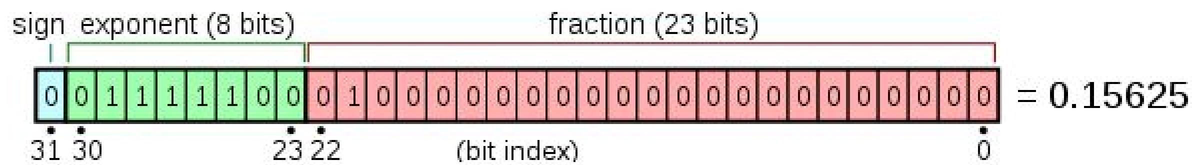
Real Number Representations

IEEE 754 Floating point standard

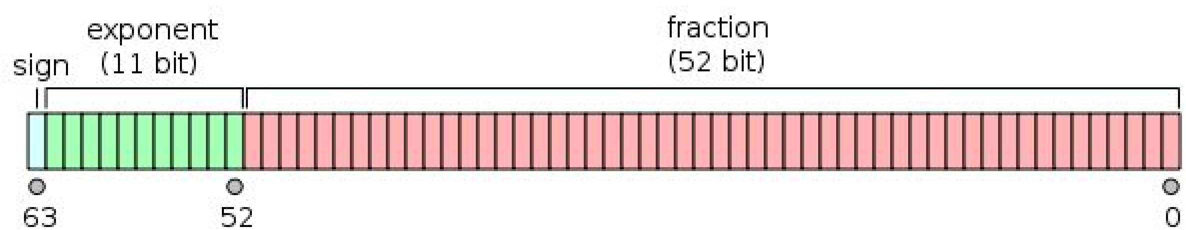
- **Reals**

Pascal is the only language I've encountered that calls them reals instead of floats

  - Sign bit *(1 bit)*
  - Exponent *(8 or 11 bits)*
  - Mantissa (fraction) *(23 bits or 52 bits)*
  - **Single**

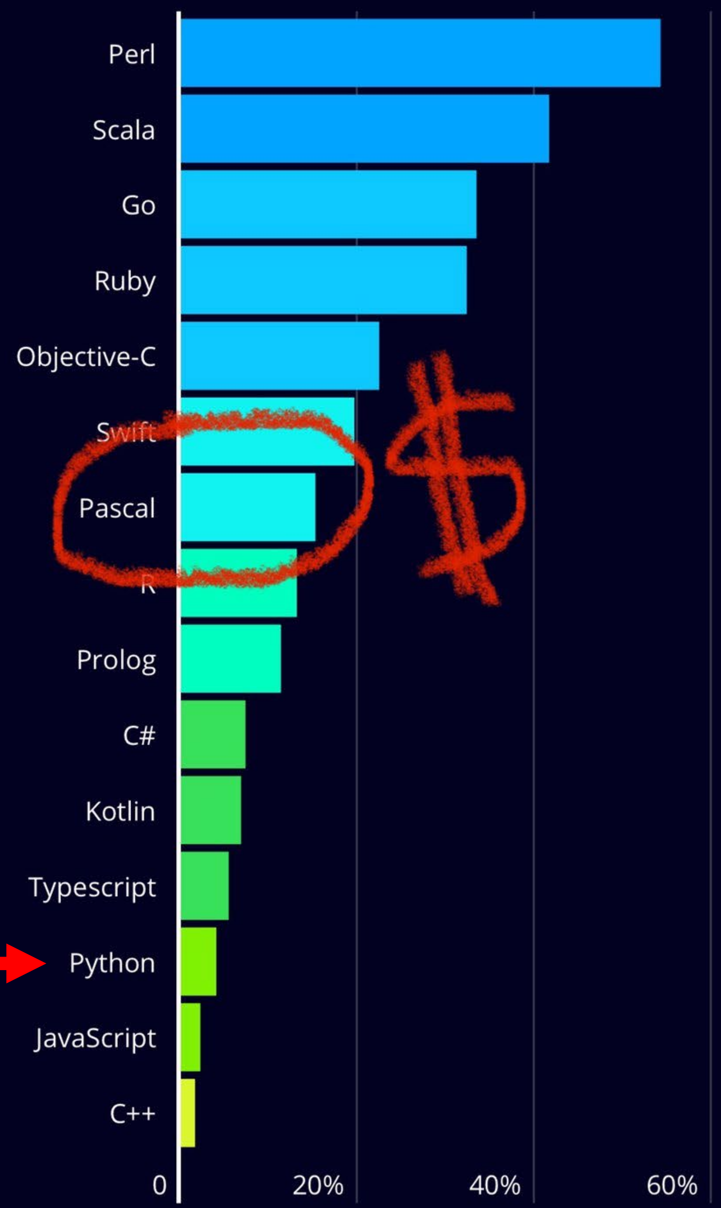


  - **Double**



However:

Salary increase based on languages known



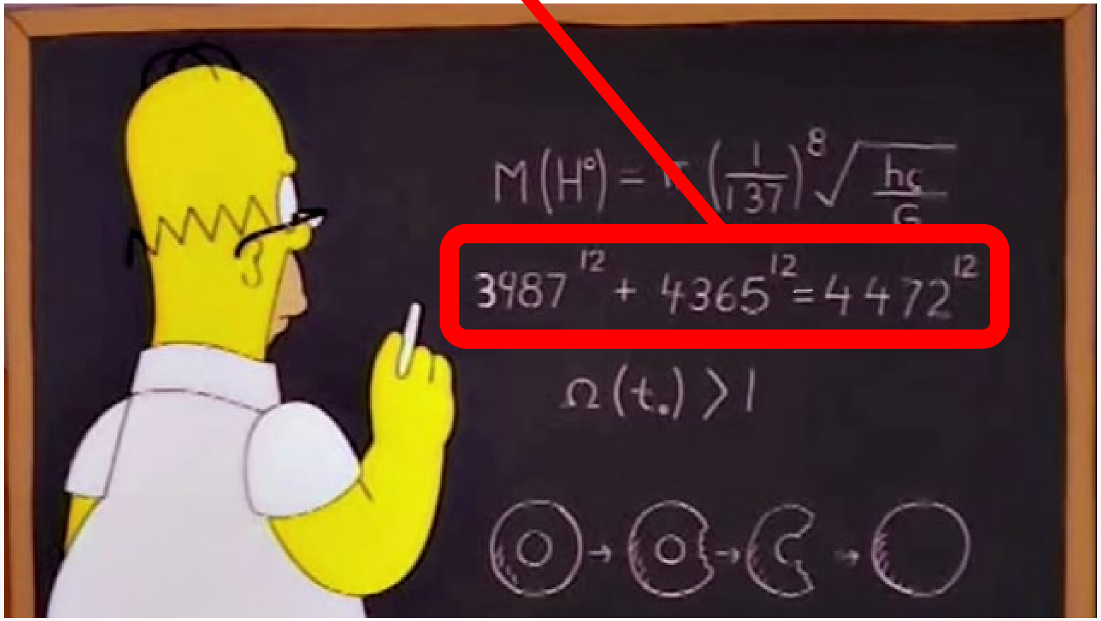
Float Issues (single, double)

- **Precision Error** $Error = |actual - representation|$
  - Most numbers don't get represented exactly
  - Finite precision of IEEE floating point
  - Represented by nearest real (floating point) number

- **Numeric Stability** (does error overwhelm?) $Error = |true\_answer - computed|$
  - Truncation Errors
  - Accumulated error from repeated calculations

Numerical precision: a real-world example

- Fermat's Last Theorem: there are no positive integer solutions to $X^n + Y^n = Z^n$ for $n > 2$
- **However:**
  - **There is a Simpsons episode where Homer discovers that a crayon he inserted in his nose as a child has migrated into his frontal lobe, suppressing his cognitive capacity. The doctor removes the crayon, and Homer's true intellect is revealed.**



Did Homer Simpson disprove Fermat's Last Theorem?

- Does 3987^12 + 4365^12 = 4472^12 ???
- On a scientific calculator, 3987^12 + 4365^12 – 4472^12 gives you an answer of…

- Feeling alone in a world where no one understands his genius, Homer reinserts the crayon into his brain and returns to his normal life.
- **But what happens if we use double-precision floats?**

Look at these matlab scripts

- Just look at them.

But how much memory do these variable types take?

- Good question.

How is memory/storage organized in a computer?

- How big should I expect a data file to be?
- How is the output of a 12-bit DAC stored?
- How should I configure storage on computers in the lab?

Bonus 1: complex exponentials and further explanation of Fourier transforms

Bonus 2: linear vs. switching-mode power regulators