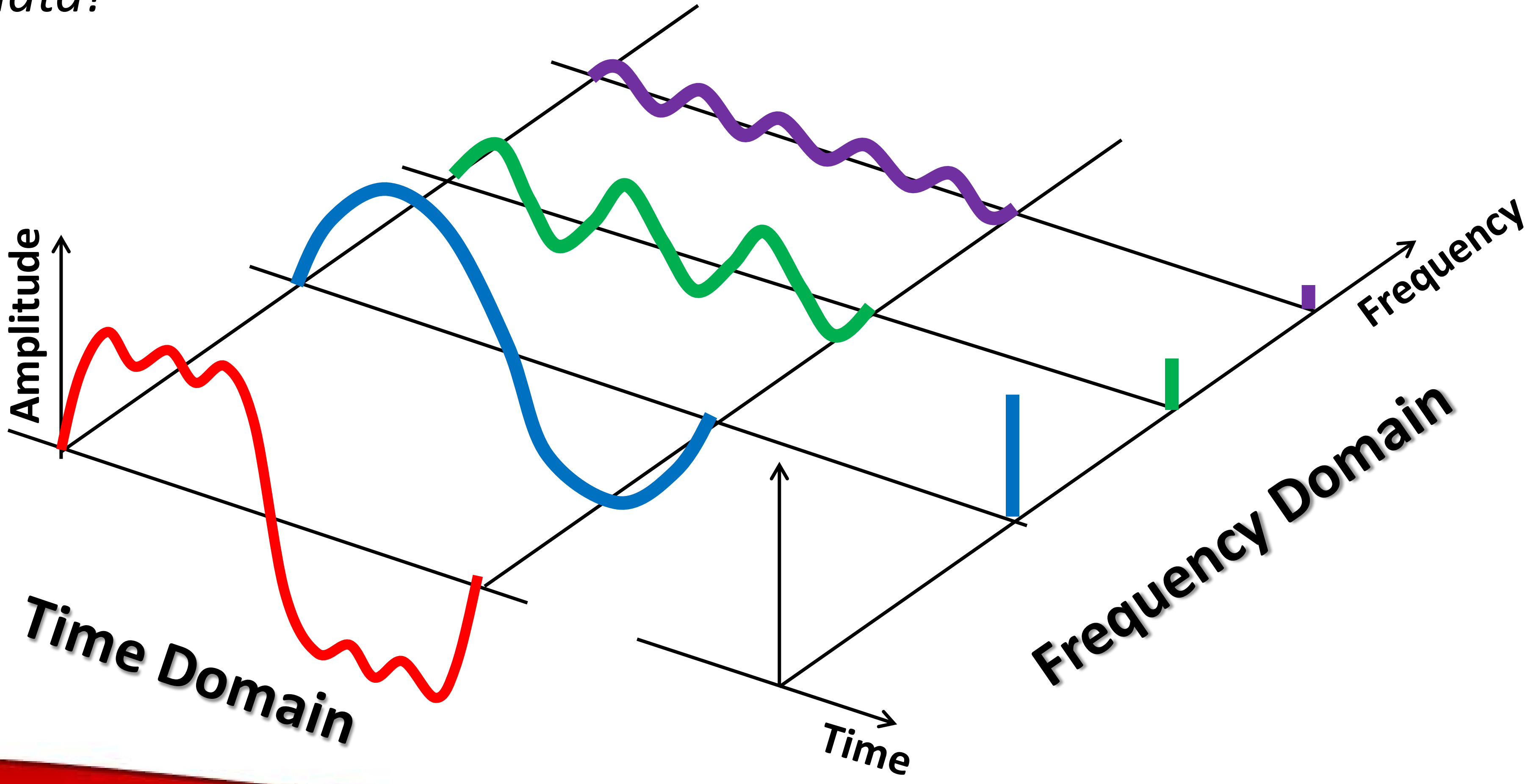
Fourier Transforms

Modified from: http://hometheaterhifi.com/technical/technical-reviews/up-sampling-aliasing-filtering-ringing-a-clarification-of-terminology/

Arguably the most important set of algorithm for analysis and manipulation of discrete data!



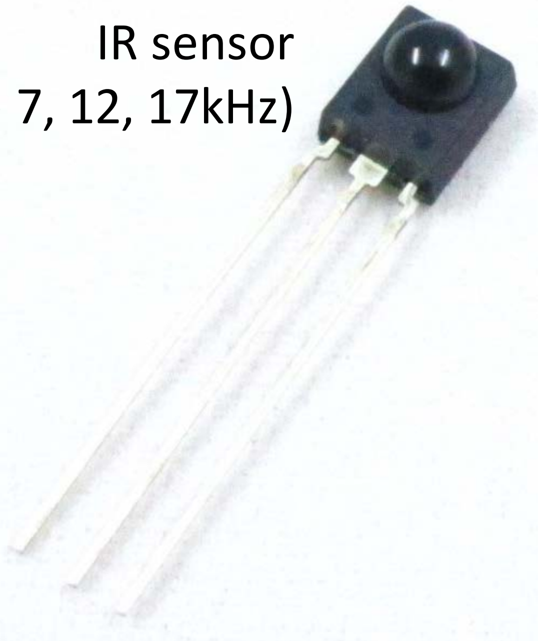
Fourier Transforms

Why should *you* care *now*?

- Because you need to analyze noisy sensor signals!
- Analog filters
  - Bulky and require accurate (expensive) components
  - Low relative Q-values
- Discrete filters
  - Require memory, processing time, and an ADC
  - High relative Q-values and are versatile

Microphone
(Sense 660Hz)

IR sensor
(Sense 7, 12, 17kHz)

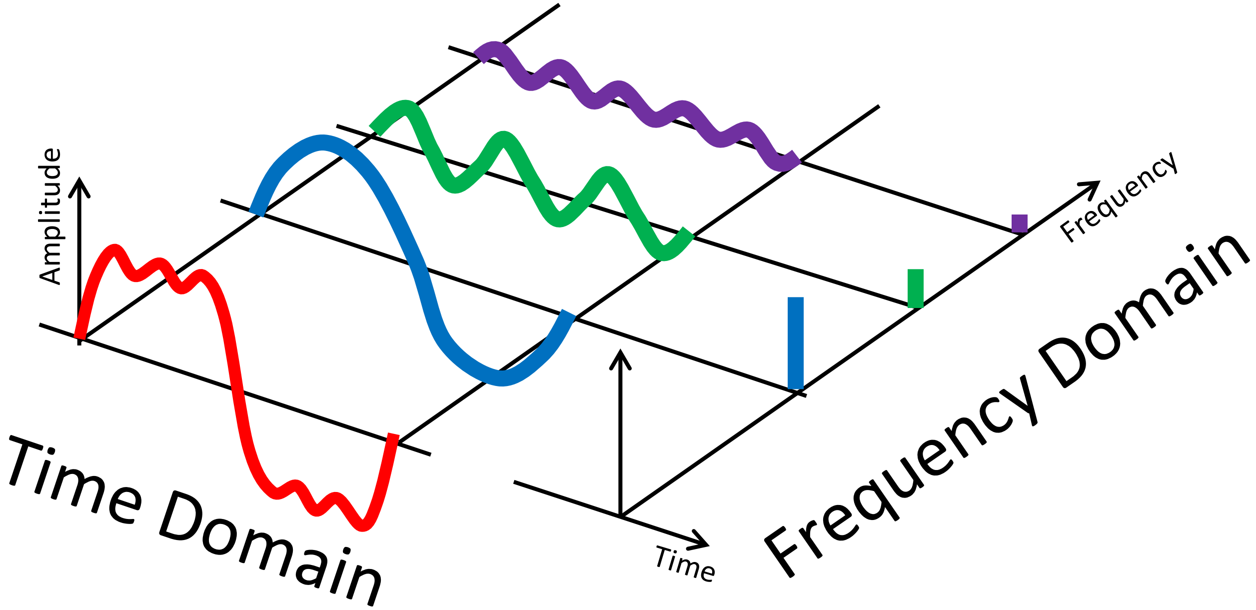


Fast Fourier Transforms

frequency = The sum of contributions to this frequency from each point in time

$$F(x) = \sum_{n=0}^{N-1} f(n) \cdot e^{-i2\pi xn/N}$$

$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x) \cdot e^{i2\pi kn/N}$$

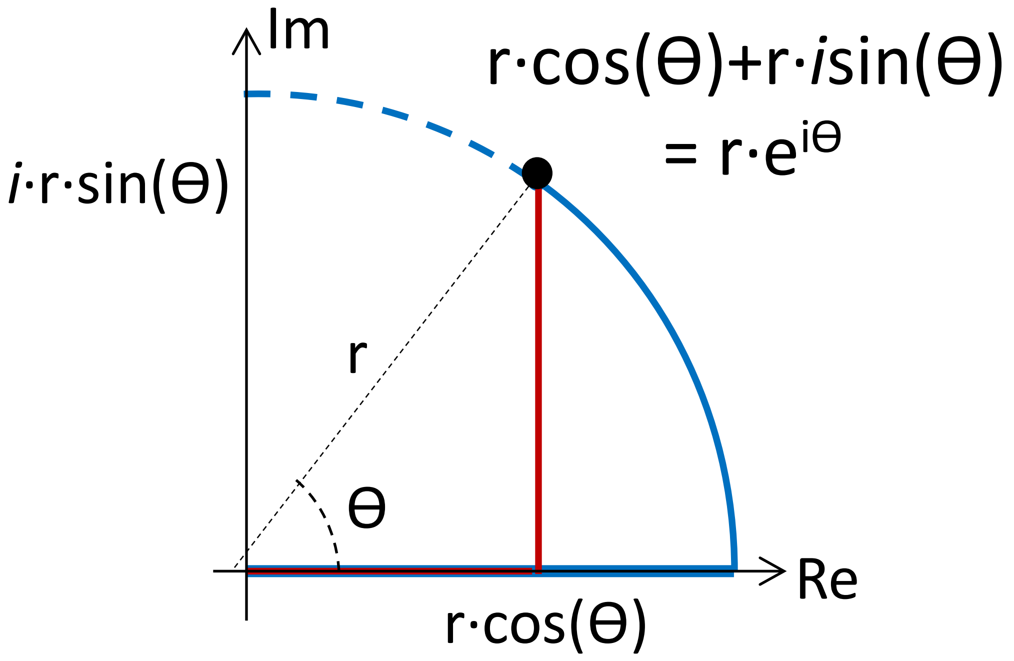


Time point = The sum of contributions to this time point from each frequency

Describing a signal in terms of cycles

$$F(x) = \sum_{n=0}^{N-1} f(n) \cdot e^{-i2\pi xn/N}$$

$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x) \cdot e^{i2\pi kn/N}$$

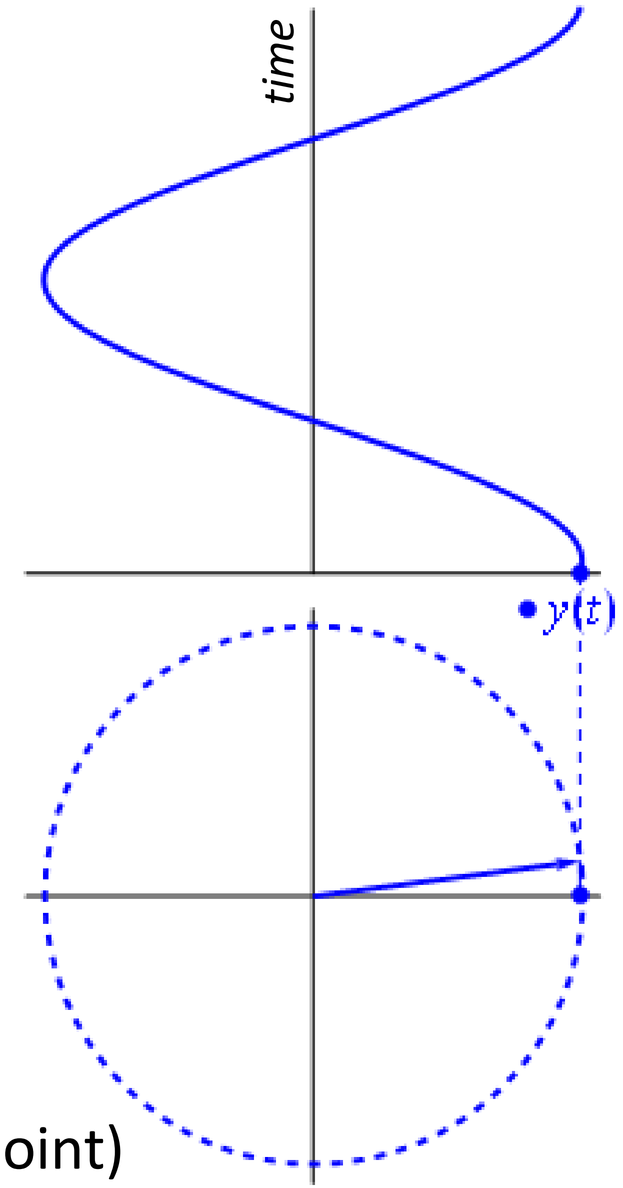


Describing a signal in terms of cycles

$$F(x) = \sum_{n=0}^{N-1} f(n) \cdot e^{-i2\pi xn/N}$$

$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x) \cdot e^{i2\pi kn/N}$$

- Frequency (speed)
- Amplitude (radius)
- Phase Angle (starting point)



Describing a signal in terms of cycles

$$F(x) = \sum_{n=0}^{N-1} f(n) \cdot e^{-i2\pi xn/N}$$

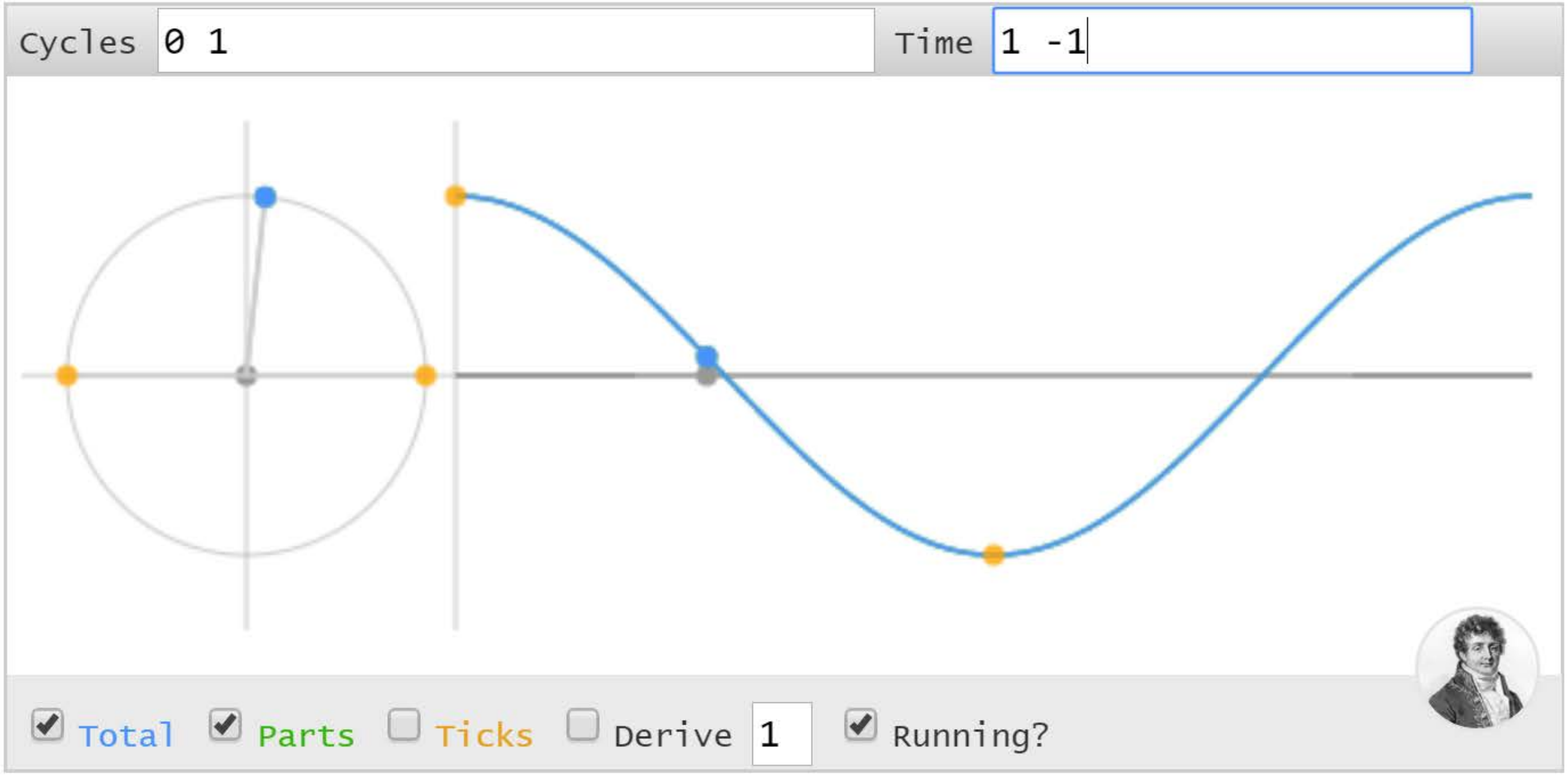
$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x_k) \cdot e^{i2\pi kn/N}$$

To find the energy at a particular frequency, spin your signal around a circle at that frequency, and average a bunch of points along that path.

Stuart Riffle, Blogaday

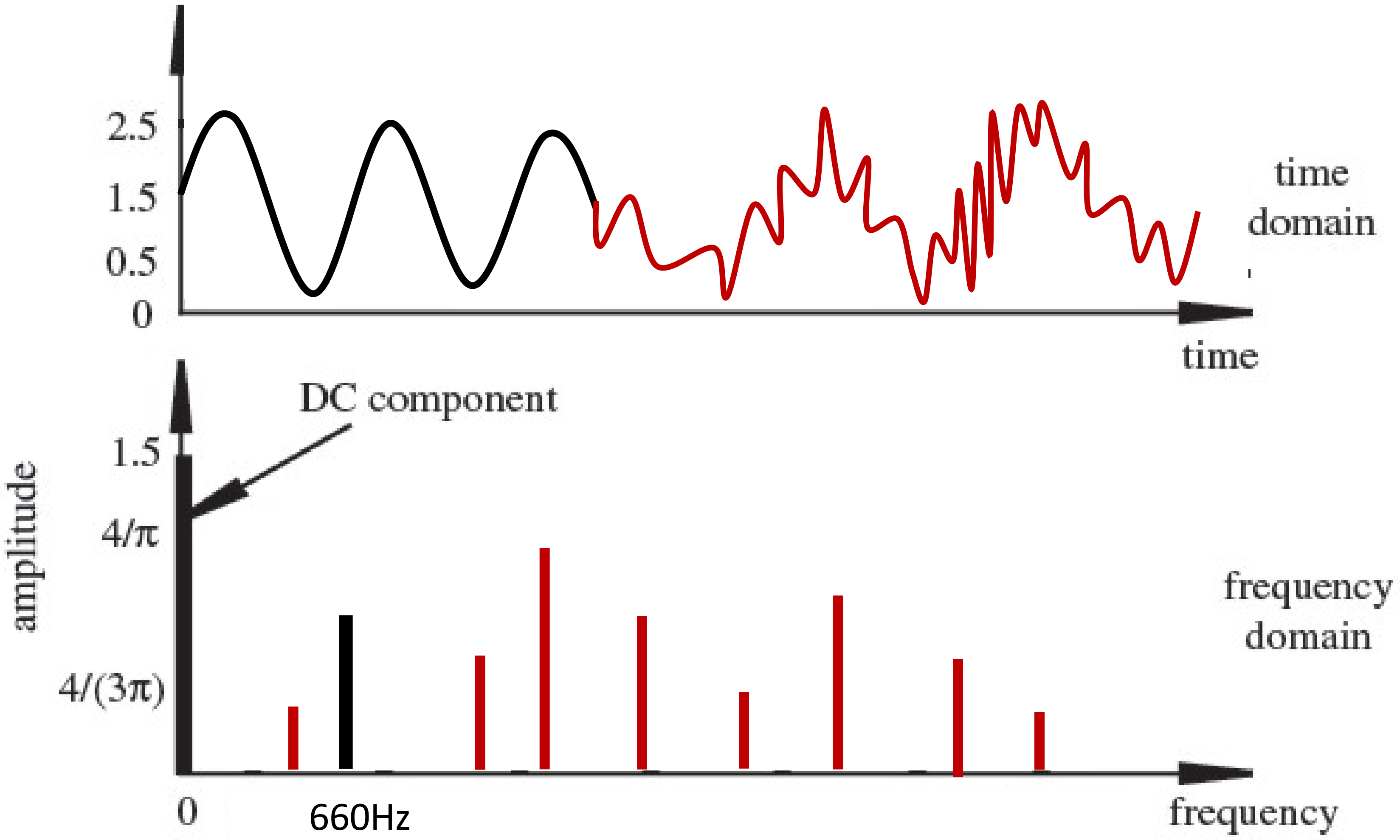
Describing a signal in terms of cycles

https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/

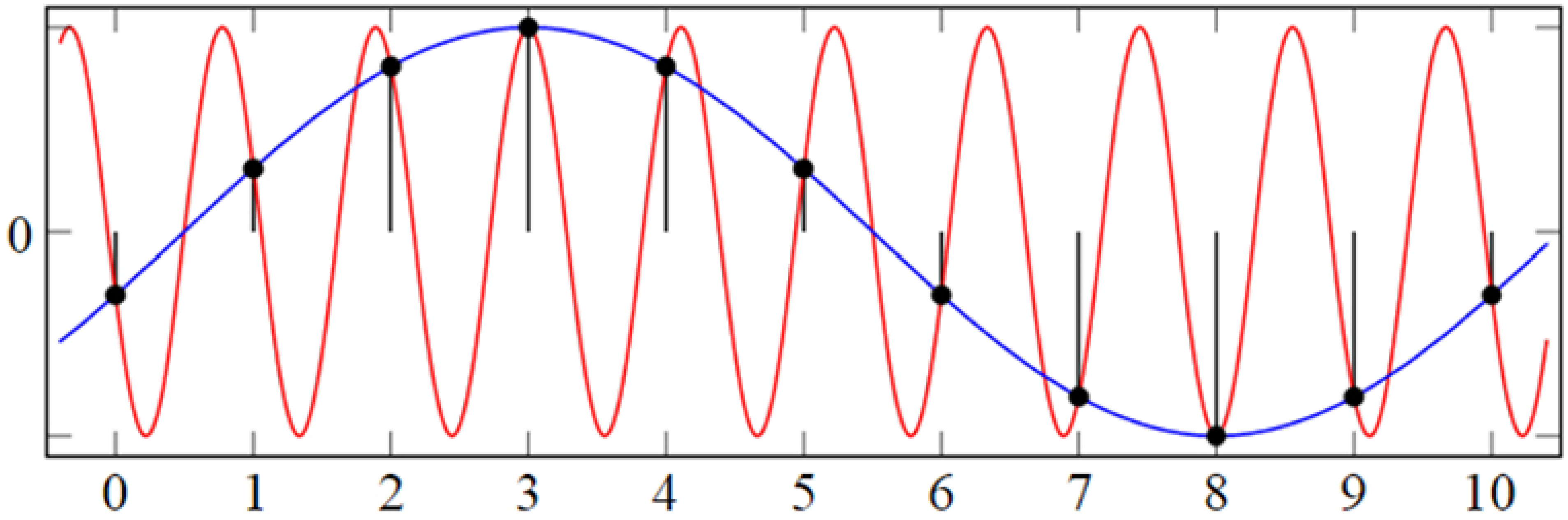


Lab 2

Detect start signal



Discretized Signals and Aliasing

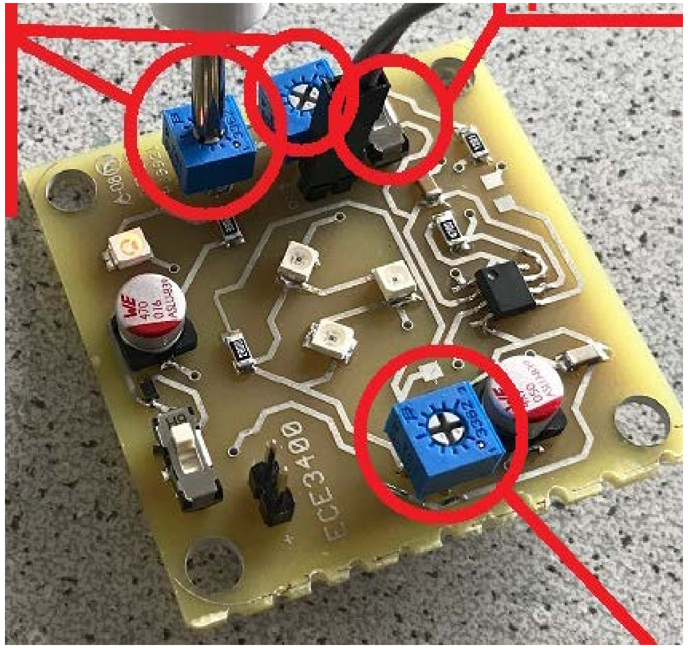


Nyquist Theorem

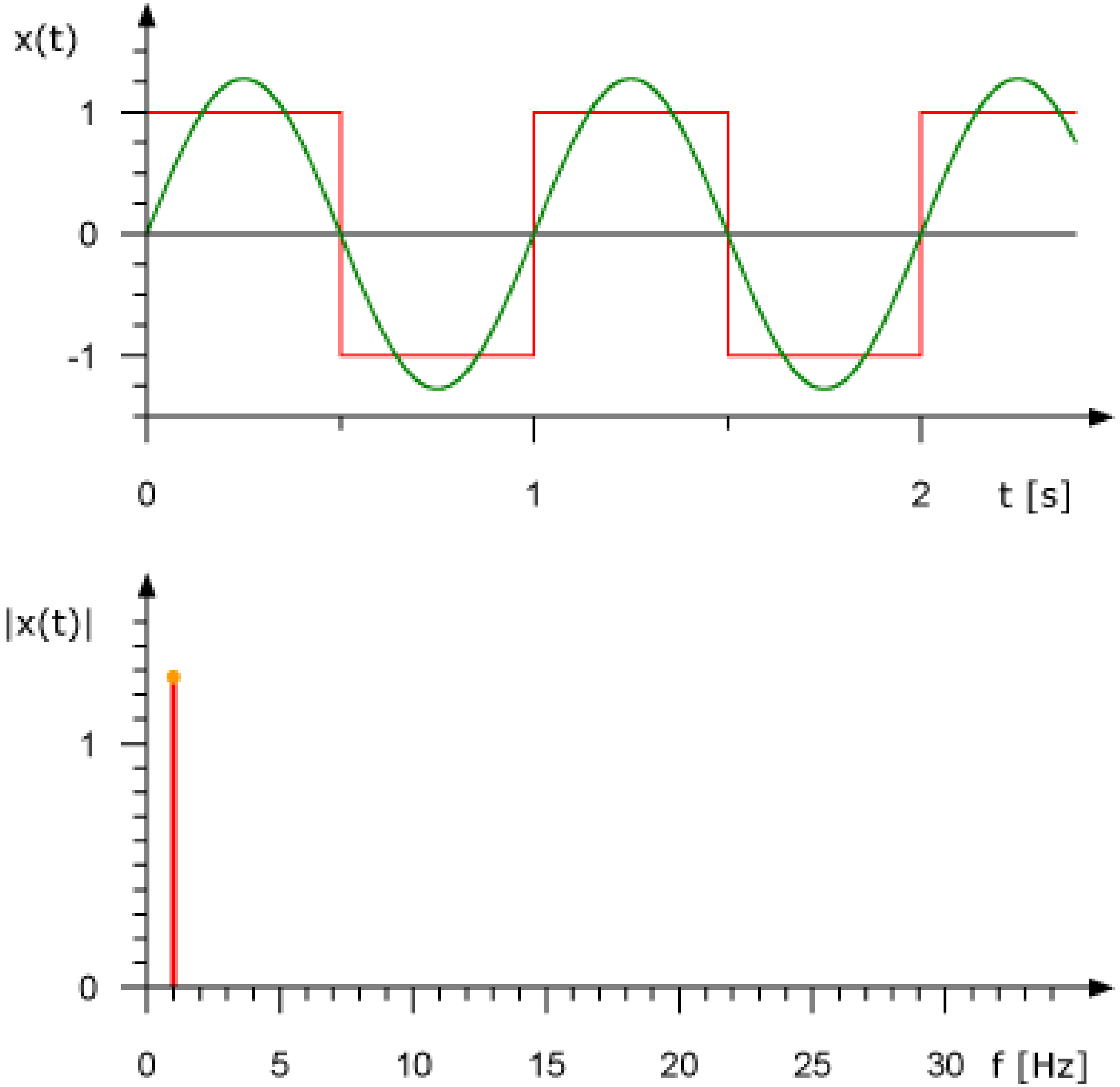
The sampling rate must be at least twice the highest frequency component of the signal:

$$f_{sample} = 2 \cdot f_{max}$$

Lab 2



Treasures
(7kHz, 12kHz, 17kHz)



Open Music Labs FFT Library Functions

Courtesy of Vaidehi Garg

`fft_run()`

- Main FFT functional call
- No input variables, no return variables
- Assumes already re-ordered data is available in memory
- Data is stored in array of size N, called fft_input[]
- The array contains 2 16-bit values per FFT datapoint
- Even positions are for real values, odd for imaginary
- Every two positions corresponds to one bin => N/2 FFT bins

Open Music Labs FFT Library Functions

Courtesy of
Vaidehi Garg

Before calling fft_run():

- Decide N based on the number of FFT bins you want.
- Fill fft_input[] with datapoints from the ADC. Since the datapoints are real, put them in the even positions, and fill the odd positions with 0's.

Open Music Labs FFT Library Functions

Courtesy of
Vaidehi Garg

After calling fft_run():

Use one of the provided functions to process the FFT output and obtain the output array.

fft_mag_log() returns the output in sequential order of FFT frequency bins!

Check the value of the relevant bin (which bin # would you need to check for a 660Hz or 7kHz frequency?)

(do the prelab!)

Go Build Robots!

http://www.owendavey.com/Build-a-Robot