Hanze
University of Applied Sciences
Groningen

Digital Signal Processing

share your talent.
move the world.

ewout bergsma.

Today

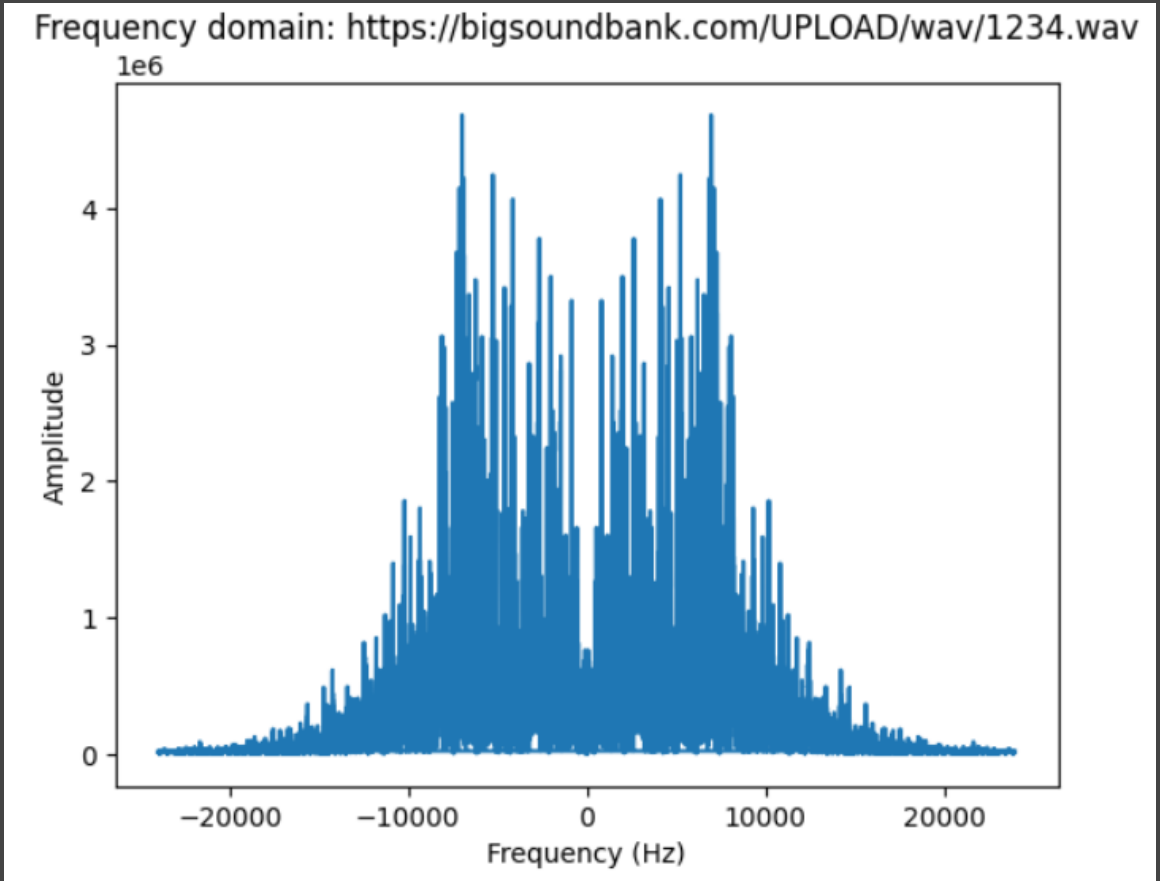
- Some additional stuff regarding last time
  - Negative frequencies
  - Spectral leakage
- Nyquist Frequency and Aliasing
- Exercises

Hanze
University of Applied Sciences
Groningen

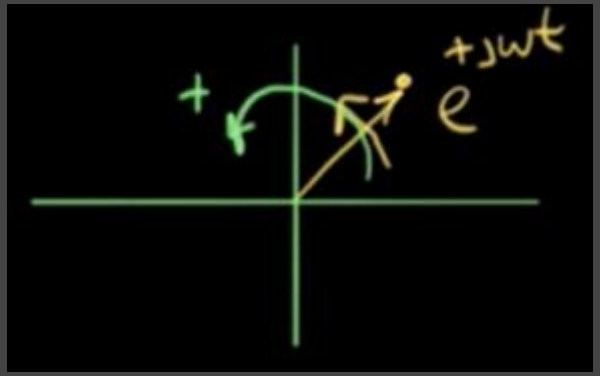
Negative Frequencies

share your talent.
move the world.

ewout bergsma.


Frequency domain: https://bigsoundbank.com/UPLOAD/wav/1234.wav
1e6
4
3
2
1
0
Amplitude
−20000
−10000
0
10000
20000
Frequency (Hz)

Source: https://www.youtube.com/watch?v=SvFfVtRT1UI



Source: https://www.youtube.com/watch?v=SvFfVtRT1UI

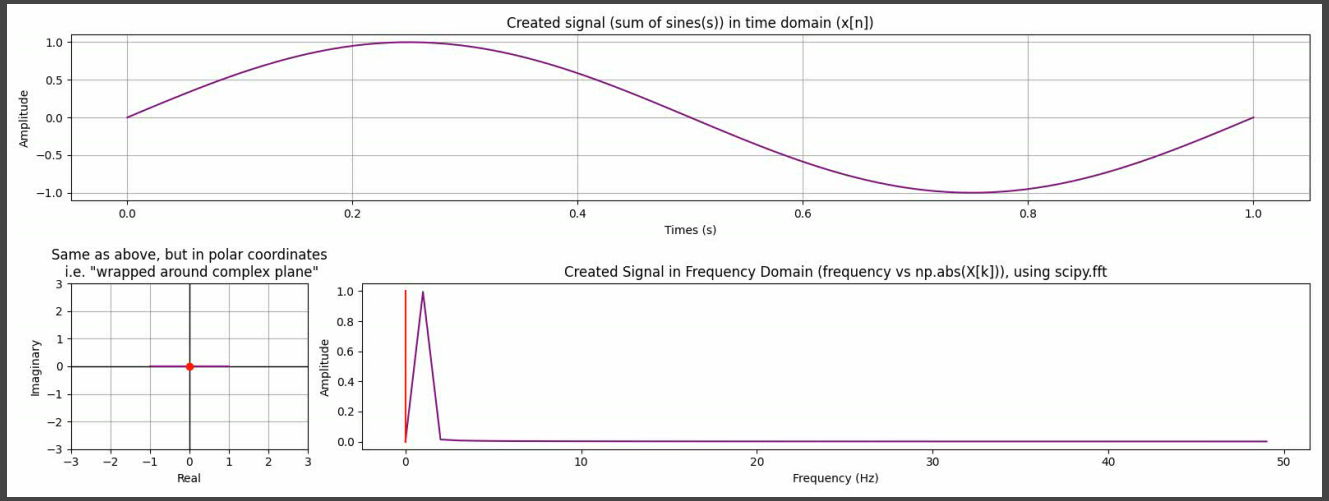
https://www.khanacademy.org/math/algebra-home/alg-complex-numbers

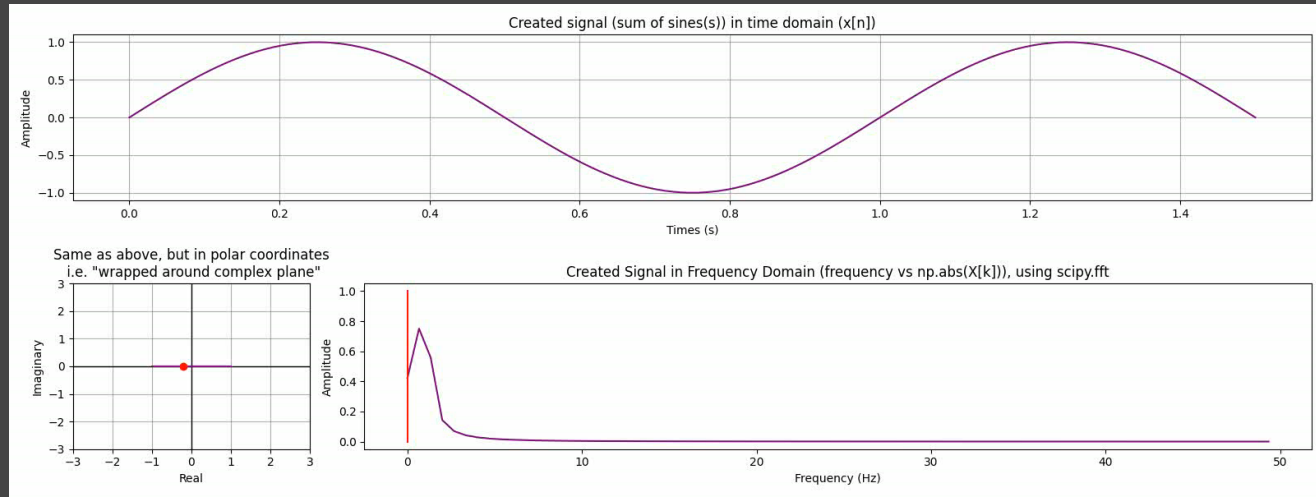
Hanze
University of Applied Sciences
Groningen

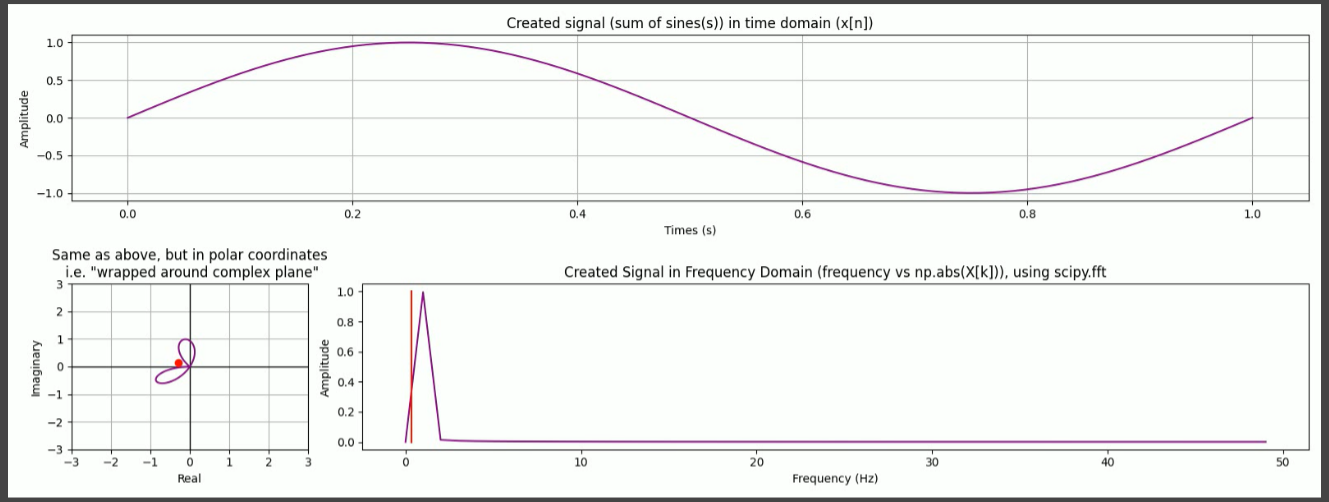
Spectral Leakage

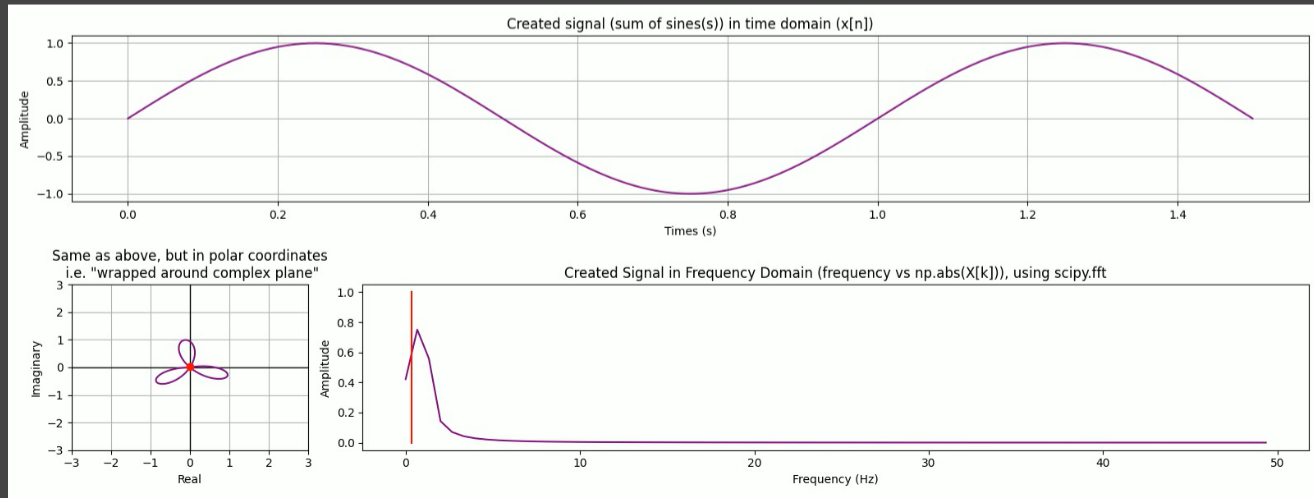
share your talent.
move the world.

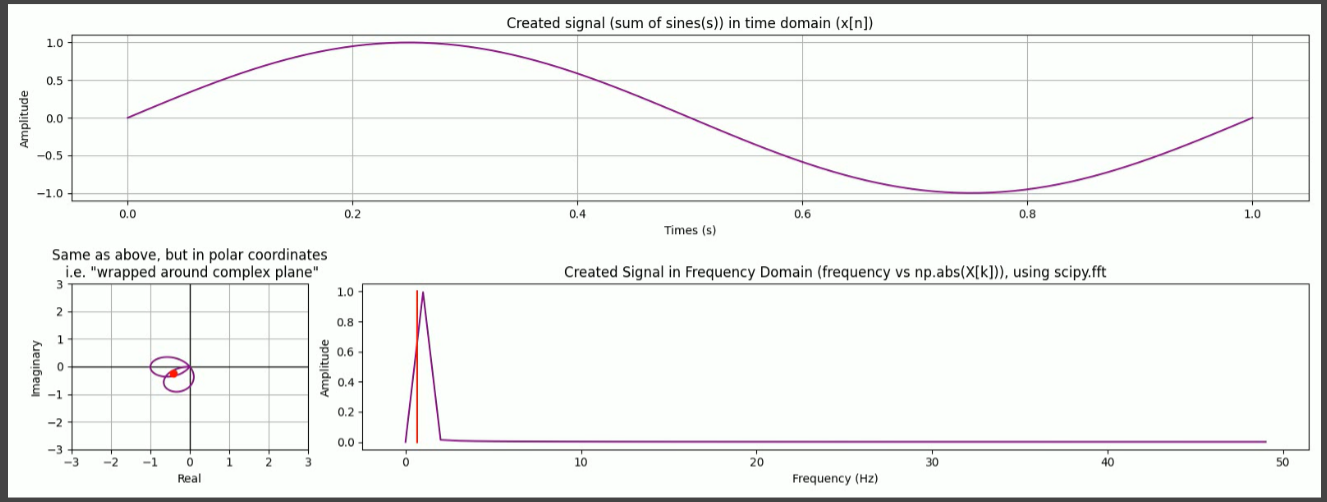
ewout bergsma.

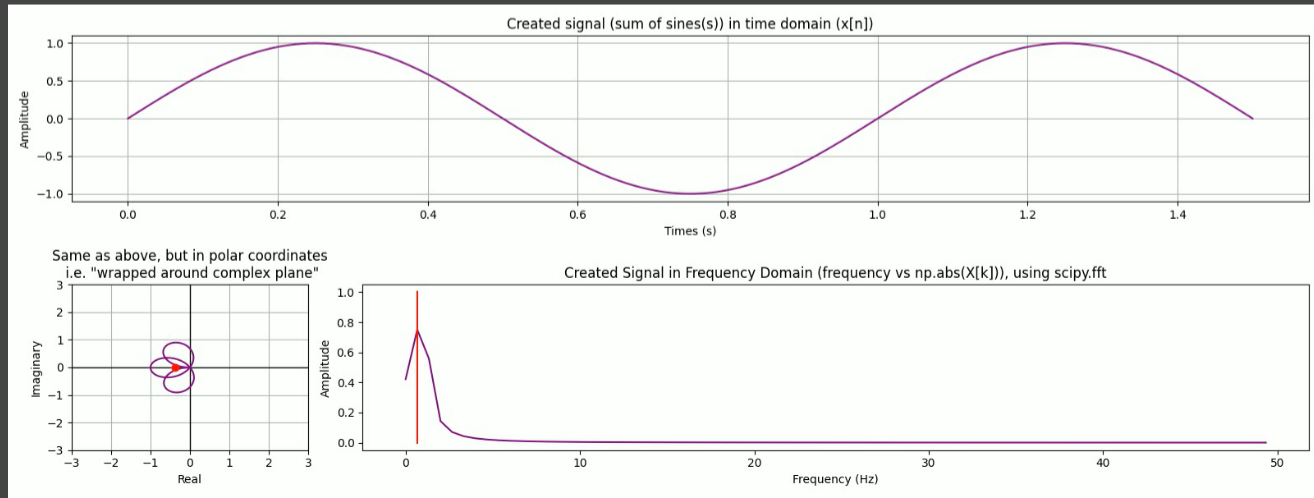

Created signal (sum of sines(s)) in time domain (x[n])
Amplitude
Times (s)
Same as above, but in polar coordinates
i.e. "wrapped around complex plane"
Imaginary
Real
Created Signal in Frequency Domain (frequency vs np.abs(X[k])), using scipy.fft
Amplitude
Frequency (Hz)


Created signal (sum of sines(s)) in time domain (x[n])
Amplitude
Times (s)
Same as above, but in polar coordinates
i.e. "wrapped around complex plane"
Imaginary
Real
Created Signal in Frequency Domain (frequency vs np.abs(X[k])), using scipy.fft
Amplitude
Frequency (Hz)


Created signal (sum of sines(s)) in time domain (x[n])
Amplitude
Times (s)
Same as above, but in polar coordinates
i.e. "wrapped around complex plane"
Imaginary
Real
Created Signal in Frequency Domain (frequency vs np.abs(X[k])), using scipy.fft
Amplitude
Frequency (Hz)


Created signal (sum of sines(s)) in time domain (x[n])
Amplitude
Times (s)
Same as above, but in polar coordinates
i.e. "wrapped around complex plane"
Imaginary
Real
Created Signal in Frequency Domain (frequency vs np.abs(X[k])), using scipy.fft
Amplitude
Frequency (Hz)


Created signal (sum of sines(s)) in time domain (x[n])
Amplitude
Times (s)
Same as above, but in polar coordinates
i.e. "wrapped around complex plane"
Imaginary
Real
Created Signal in Frequency Domain (frequency vs np.abs(X[k])), using scipy.fft
Amplitude
Frequency (Hz)

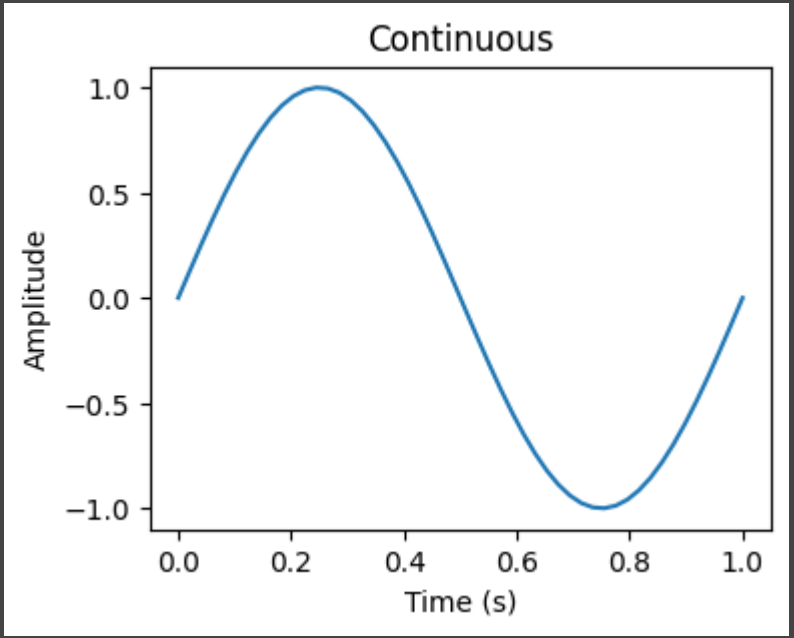

Created signal (sum of sines(s)) in time domain (x[n])
Amplitude
Times (s)
Same as above, but in polar coordinates
i.e. "wrapped around complex plane"
Imaginary
Real
Created Signal in Frequency Domain (frequency vs np.abs(X[k])), using scipy.fft
Amplitude
Frequency (Hz)

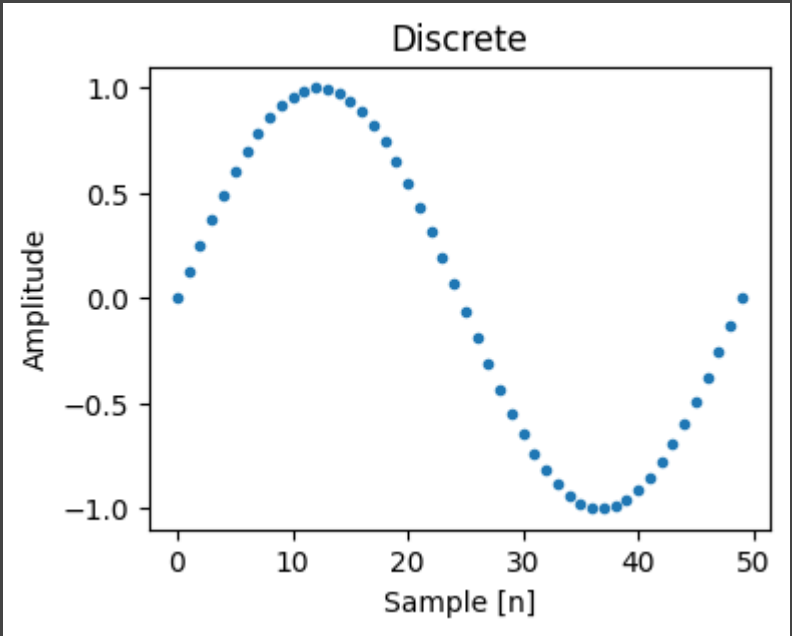
Hanze
University of Applied Sciences
Groningen

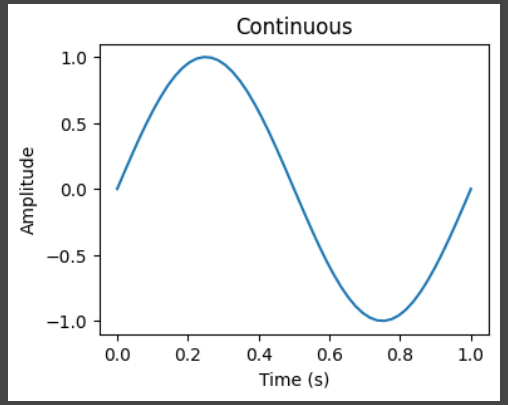
Nyquist Frequency & Aliasing

share your talent.
move the world.

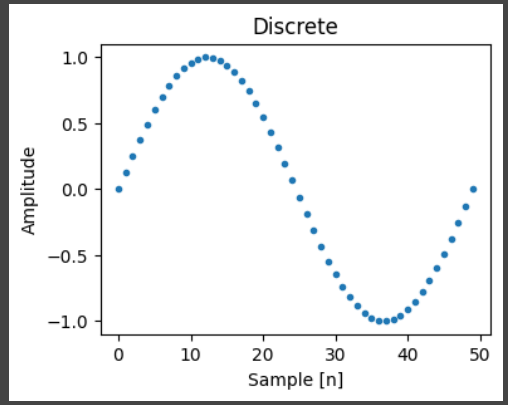
ewout bergsma.


Continuous
1.0
0.5
0.0
−0.5
−1.0
Amplitude
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)

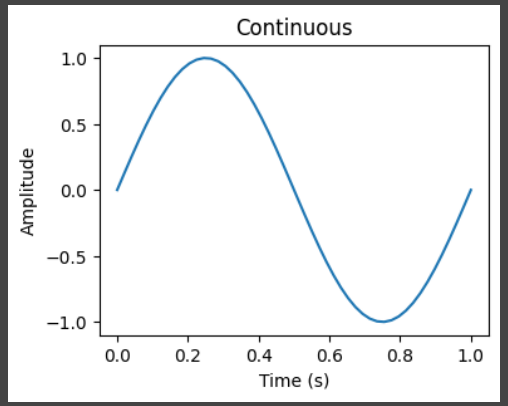

Discrete
1.0
0.5
0.0
−0.5
−1.0
Amplitude
0
10
20
30
40
50
Sample [n]



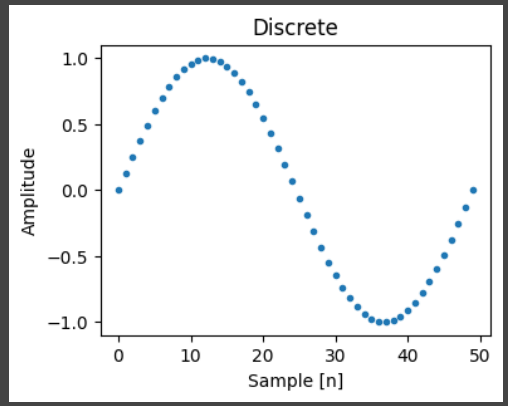
Analog to Digital Converter (ADC)



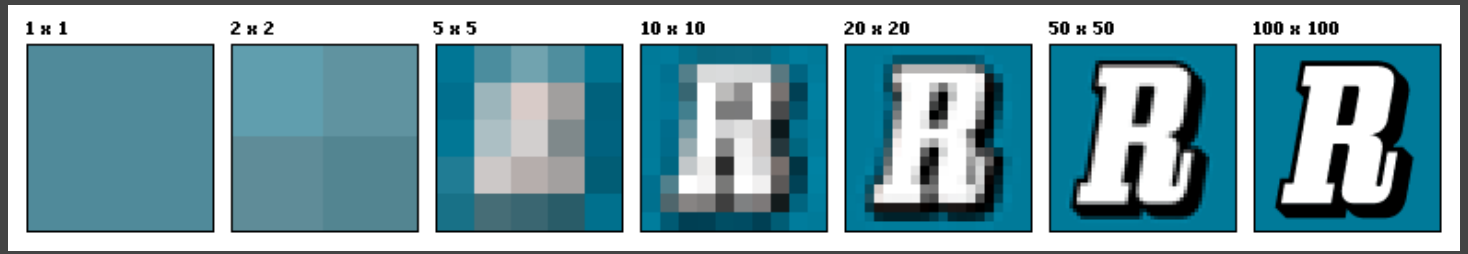
Sample rate



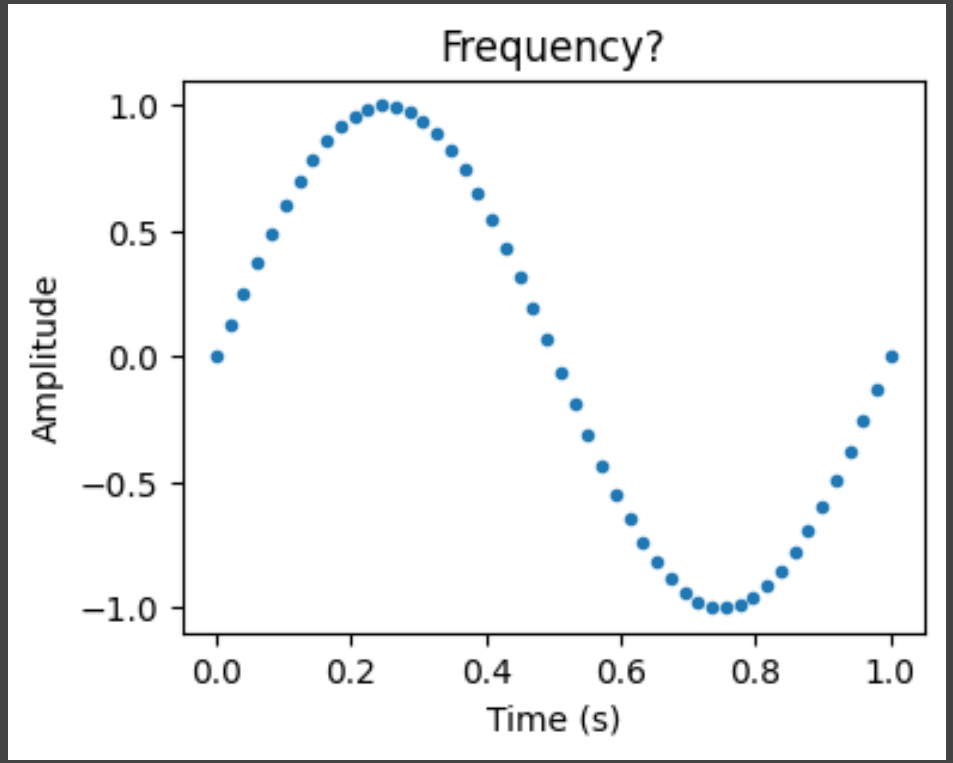
Analog to Digital Converter (ADC)

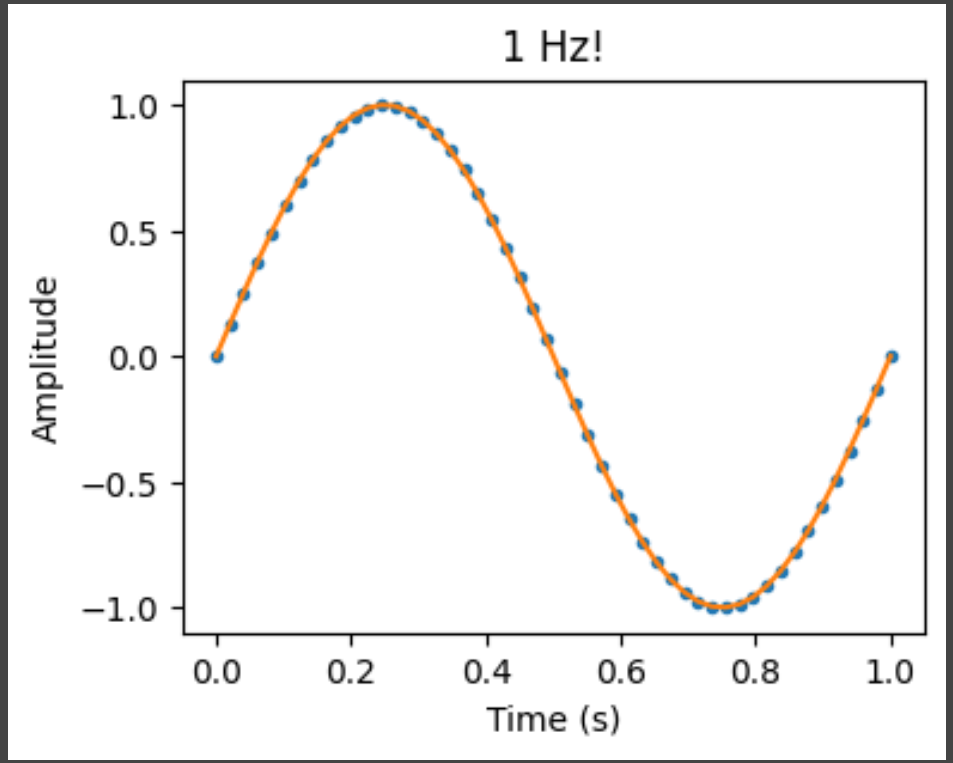


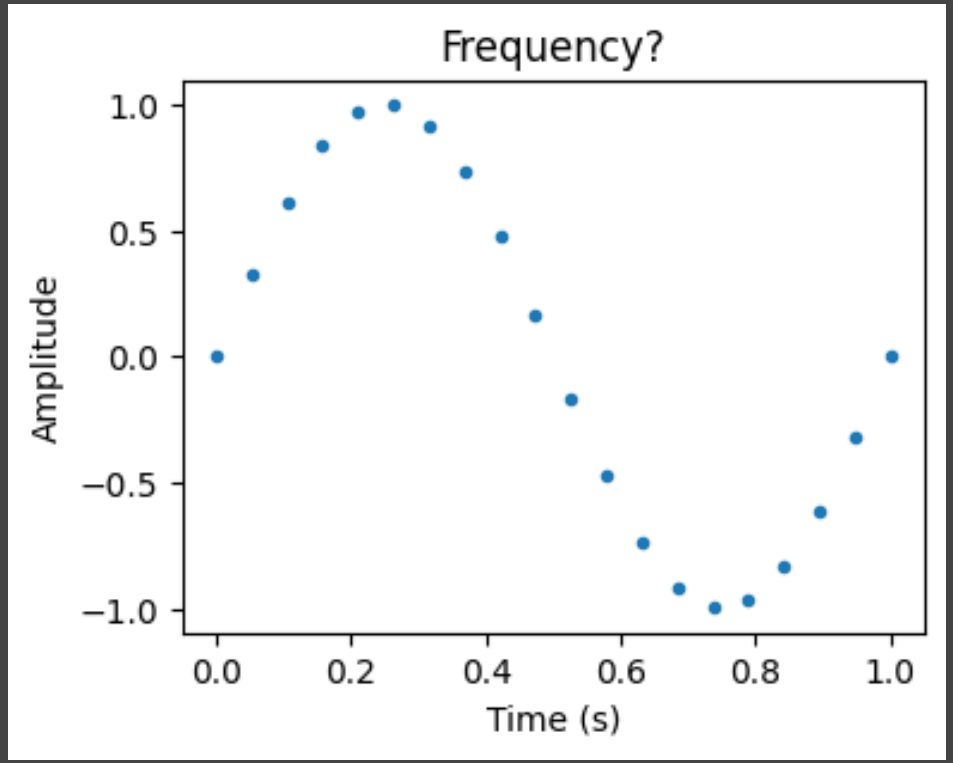
Resolution

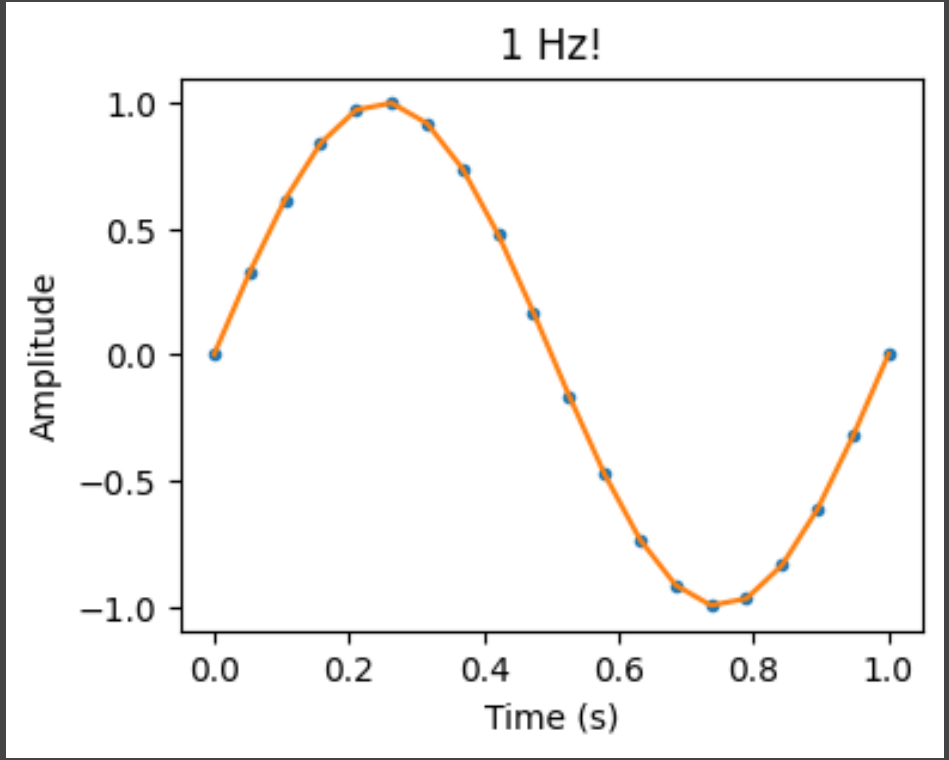


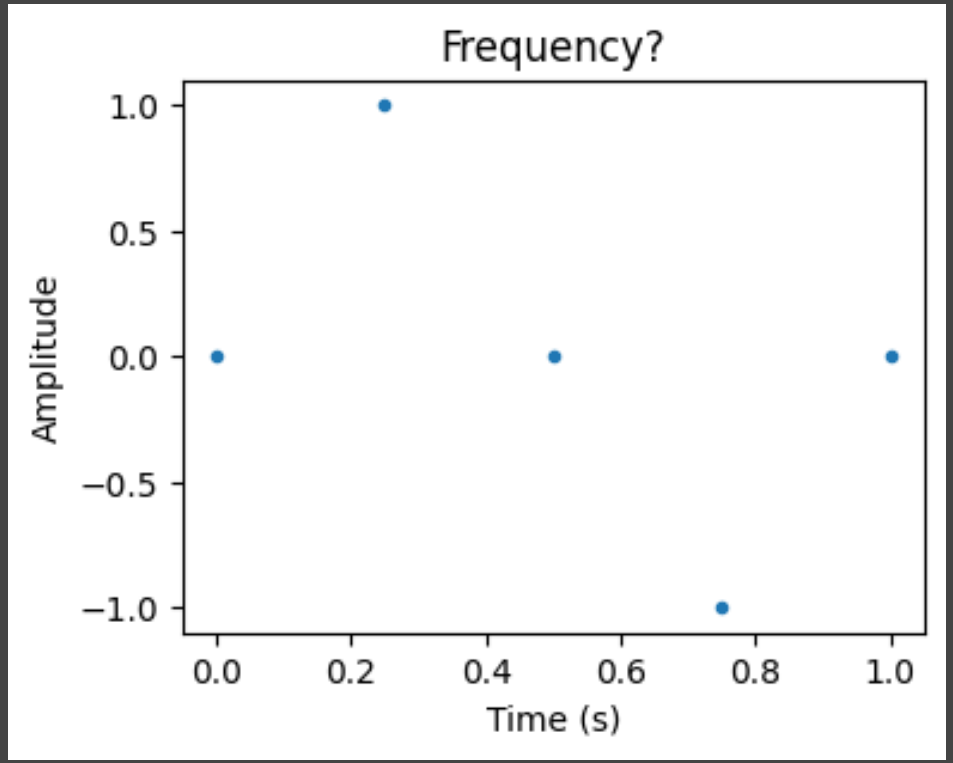
Source: https://commons.wikimedia.org/wiki/File:Resolution_illustration.png

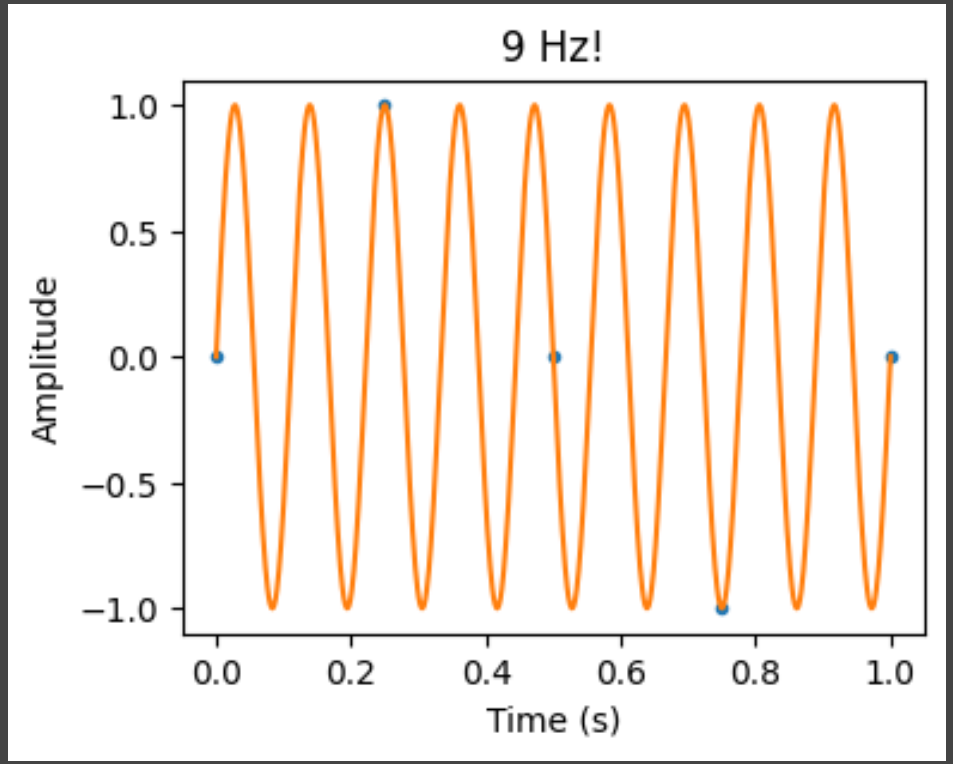

Frequency?
1.0
0.5
0.0
−0.5
−1.0
Amplitude
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)

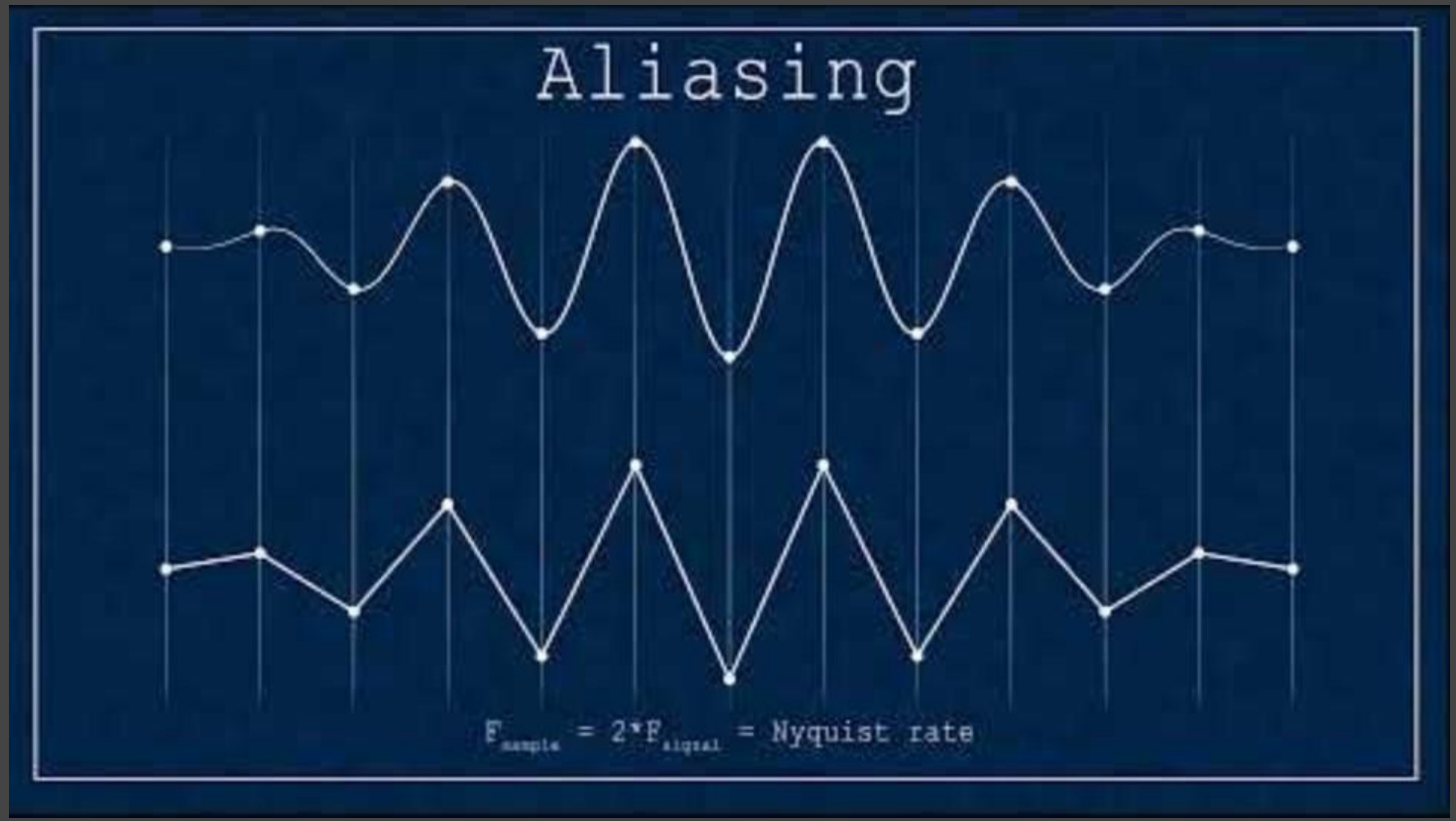

1 Hz!
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)


Frequency?
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)


1 Hz!
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)


Frequency?
1.0
0.5
0.0
−0.5
−1.0
Amplitude
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)


9 Hz!
1.0
0.5
0.0
−0.5
−1.0
Amplitude
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)



Source: https://youtu.be/CTIICx5pHvM?si=TRD44vsEh_-Y7E3u

$$f_{Nyquist} = \frac{1}{2} f_s$$

“The Nyquist frequency is the highest frequency that can be present in a discrete signal, at a given sampling rate, without causing aliasing.”

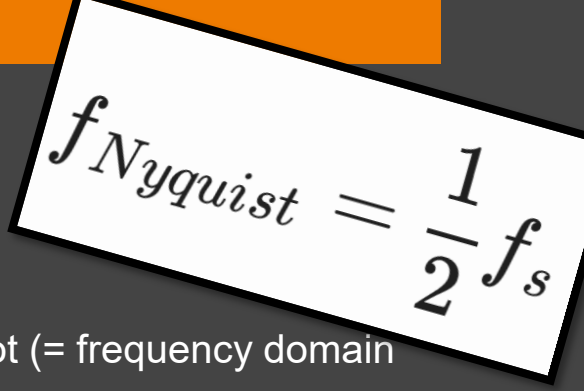
MP3 default sample rate is 44.1kHz, why?

We found 48kHz previously, but that’s besides the point.

$f_s = 44.1\text{kHz}$

Exercises!

Exercise



1. Create timeseries data, then using FFT, create a spectrum plot (= frequency domain graph) in which aliasing is present.

2. Download and load the following: https://bigsoundbank.com/UPLOAD/wav/1064.wav, in Python. Now decimate the signal by a power of 3 (= keep every third datapoint). Play the original sound, then play the decimated sound. What has happened?

3. Plot an FFT of the original and the decimated sound. Does this confirm what you previously noticed?

4. Challenge: Plot a spectrogram of sound data of your liking using https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html
   - Super challenge: Can you try out different *windows* and analyze the difference? What could be the cause of the difference?