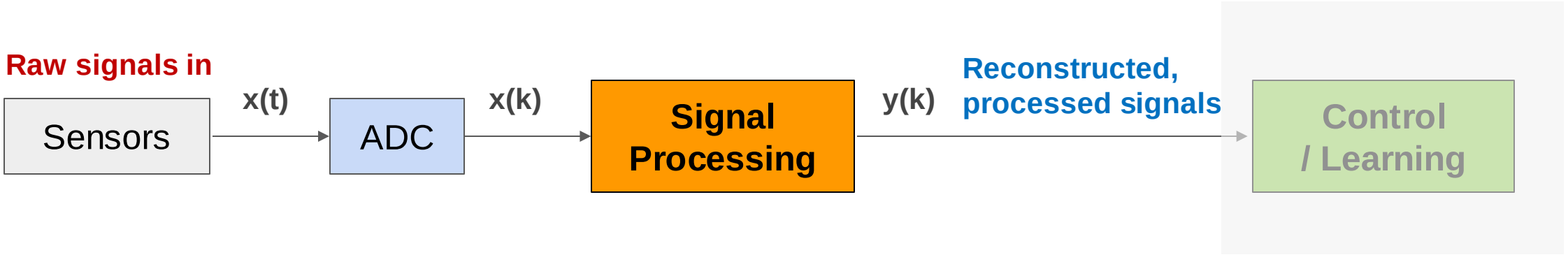
Real-world Multi-agent Systems

Sensing and Perception

Akin Delibasi
Lecturer in Distributed Systems
Computer Science
University College London, UK

Introduction

- Background of sensing & perception
- Part 1: sensing and signal processing
  - ❑ Fundamental of signal processing: Sampling, ADC and DAC, frequency domain
  - ❑ Digital filters & Processing of noisy sensory measurements
- Part 2: Perception (visual perception) and signal processing



Sensing & Perception: Sensing

- Why *Sensing & Perception*? Goal: To enable agents to gather information from their environment, facilitating informed decision-making and interactions. Agents use various sensors and data sources to **collect information**, eg, detect objects, obstacles, other agents, and environmental conditions.

- **Sensing** *allows* agents to perceive their environment. Typical types of sensors:
  - ❑ Encoders for robot joints, wheels.
  - ❑ Cameras for visual data.
  - ❑ Microphones for audio data.
  - ❑ Lidar for 3D mapping, SLAM.
  - ❑ GPS for location data (autonomous vehicles, outdoor cases).

Sensing & Perception: Perception

- **Perception**: the process of interpreting and understanding the sensory data collected, ie, how to understand the collected information/data.
- **To perceive**: Agents use perception algorithms and techniques to make sense of the sensory input.
- **Perception** enables agents to *extract meaningful information* from raw or processed sensing data.
- Typical examples of perception tasks:
  - ❑ Object recognition (QR code/markers).
  - ❑ Obstacle avoidance.
  - ❑ Speech recognition.
  - ❑ Gesture recognition.

Importance of Sensing & Perception

Sensing & Perception are crucial for several reasons:

- Situation Awareness: Agents can assess their surroundings and respond appropriately.
- Decision-Making: Informed decisions rely on accurate sensory data and perception.
- Interaction: Agents can interact with their environment and other agents effectively.
- Adaptation: MAS can adapt to dynamic environments and changing conditions.

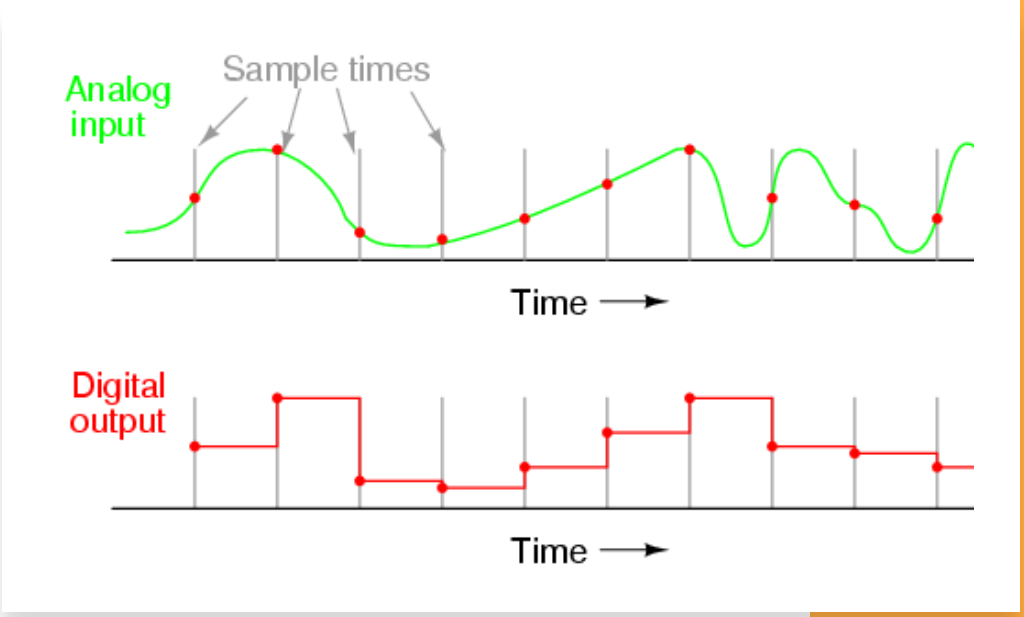
Challenges in Sensing & Perception

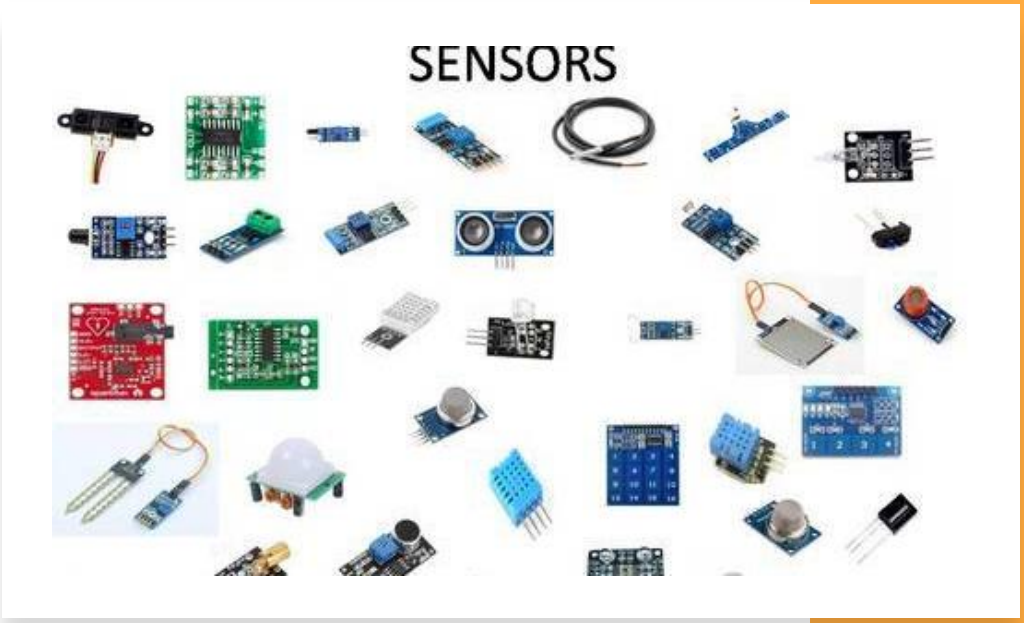
- Limited accuracy/resolution: Sensors may have accuracy subject to environmental constraints.
- Uncertainty: Sensory data can be noisy or incomplete.
- Data Fusion: Combining data from multiple sensors can be complex; directly measurement is preferable, if computation and cost permit.
- Real-Time Processing: minimal delays, essential for real-time control algorithms and timely decision-making.

Sensing in digital systems

Digital signal

- Sensing is realized in a digital format in digital systems, such different sensing information, measurement acquisition are mostly represented as digital signals.
- Digital signals are a type of signal used in digital electronics to represent and transmit information.



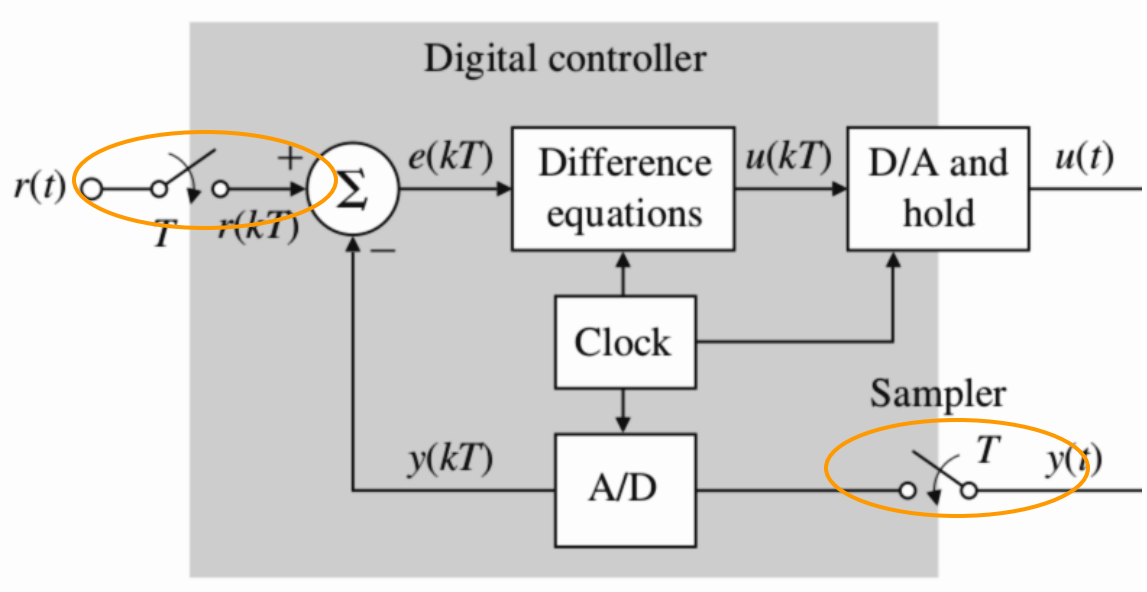


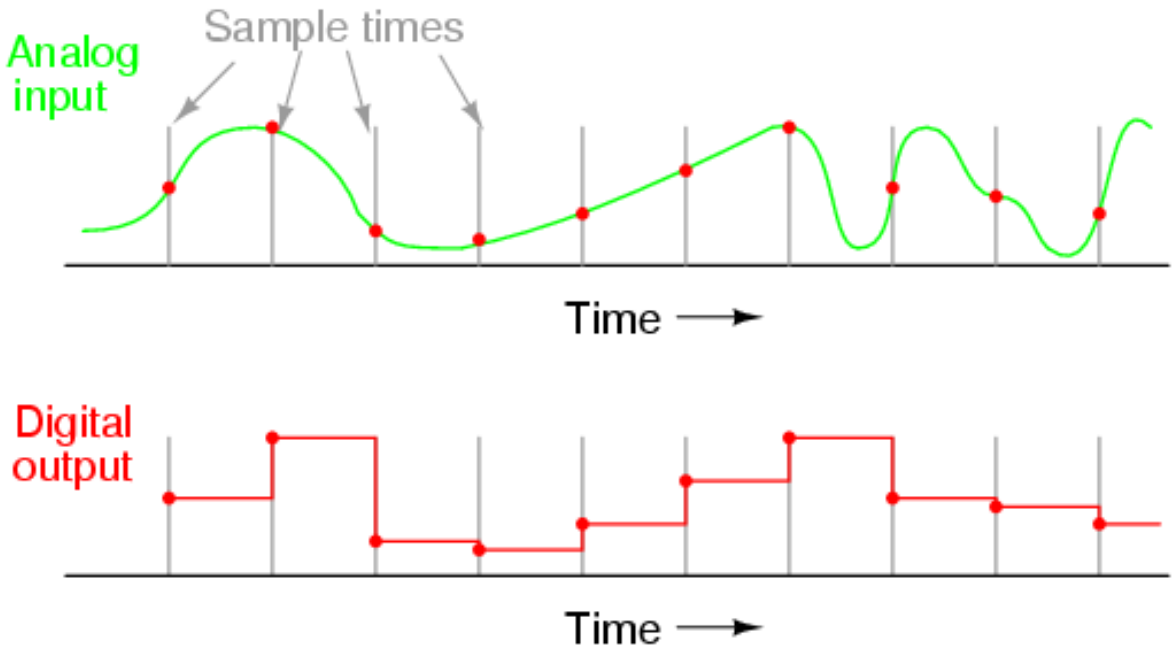
Signal processing

- Digital signal processing is the **algorithmic processing and computation** of digital signals using various mathematical algorithms and techniques to perform tasks such as filtering, noise reduction, compression, and more.

- Core categorization of its nature: it is a **computational process via algorithms**

Digitization - From analog to digital





Digitization – key components

Sampling: This is the process of converting continuous-time analog signals to discrete-time digital signals by measuring the signal amplitude at regular intervals in time.

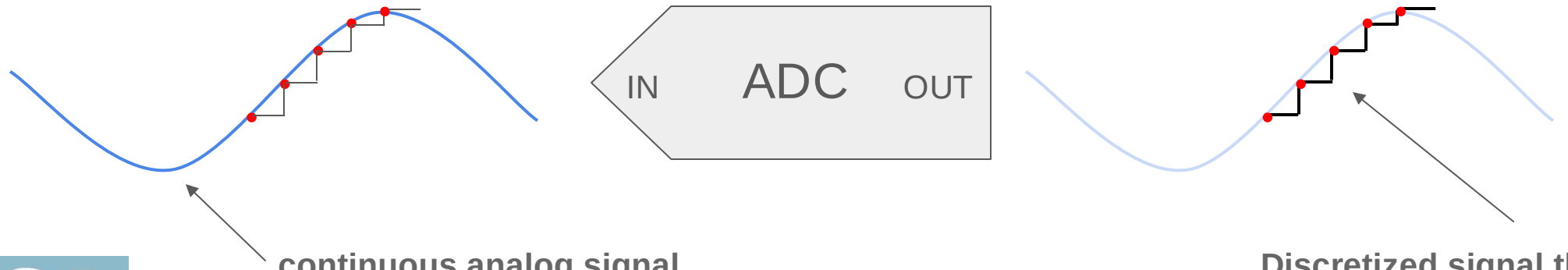
Quantization: This process involves mapping each sample value to the nearest available value in a finite set of possible values. The result is a digital signal with a finite number of possible values.

Discretization: sampling

Digital computers/chips:

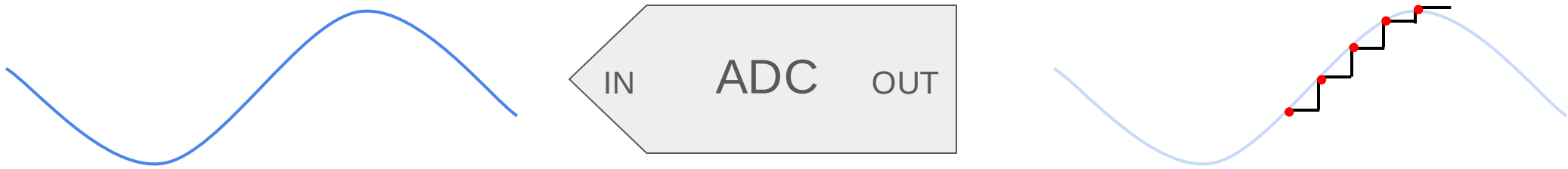
- Observe physical variables from time to time
- Use numerical integration of continuous-time ordinary differential equations (ODE)

Sampling is done by analog-to-digital converter (ADC), and the sampled value is hold over a period of **sampling time T**, ie a zero-order hold (ZOH).



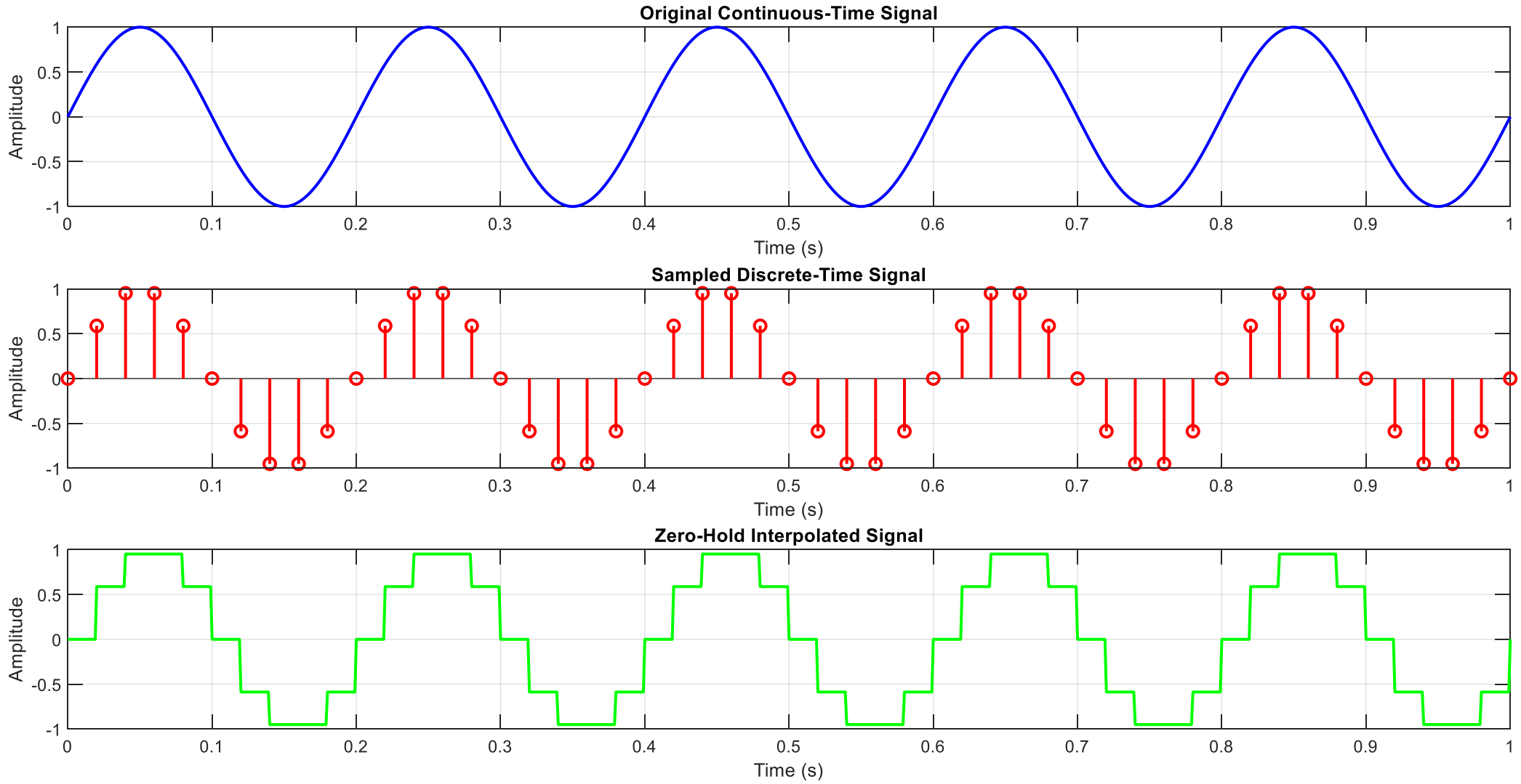
Discretization: analog-to-digital converter (ADC)

An analog-to-digital converter (ADC) samples a continuous physical signal, most commonly an electrical voltage, and converts it into a binary number that usually consists of a number of bits, eg 16-bit ADC.



Let T (T_s) be the sampling time/period, conversion from the analog signal y(t) to the samples, **y(kT)**, occurs repeatedly at an interval of T seconds. The **sampling rate/frequency f (f_s) is f=1/T in Hz**.

Sampling


Original Continuous-Time Signal
Amplitude
Time (s)
Sampled Discrete-Time Signal
Amplitude
Time (s)
Zero-Hold Interpolated Signal
Amplitude
Time (s)

Quantization

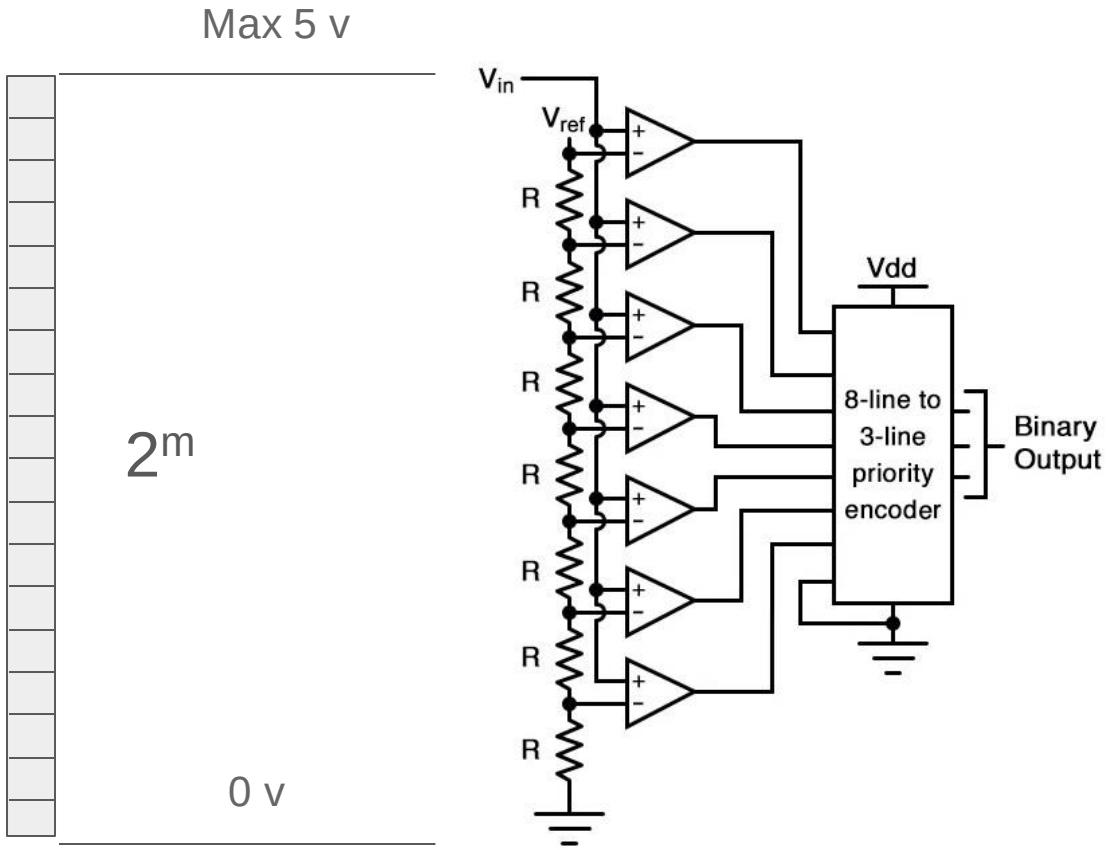
- Definition:
  - Quantization is the process of approximating continuous signal values with a limited set of discrete values.
- Importance of Quantization:
  - In practical applications, signals are typically measured and processed using digital devices that have **limited precision**.
  - Quantization allows us to represent and store signals efficiently using a finite number of bits.

Quantization

- Resolution:
  - Number of discrete levels or steps available for representing a signal.
  - Higher resolution provides more **accurate** representation, more bits!
- Quantization Error:
  - Difference between the analog signal and the closest available digital value at each sampling instant from the A/D converter.
  - Quantization Noise: This error introduces noise, called quantization noise, to the sample signal.

Quantization

Computer-aided systems are typically be quantized. A m-bit ADC can only round up or down to a real value within **2^m numbers**.



ADC: precision

In practice, the range of voltage corresponds to the range of a particular physical quantity you want to measure, therefore, the resolution Q of a sensor:

$$Q = \frac{E}{2^m}$$

E: maximum range of a physical quantity; M: number of bits

Example

Input voltage: 0~5 V, 12-bit ADC converter, what is the resolution?

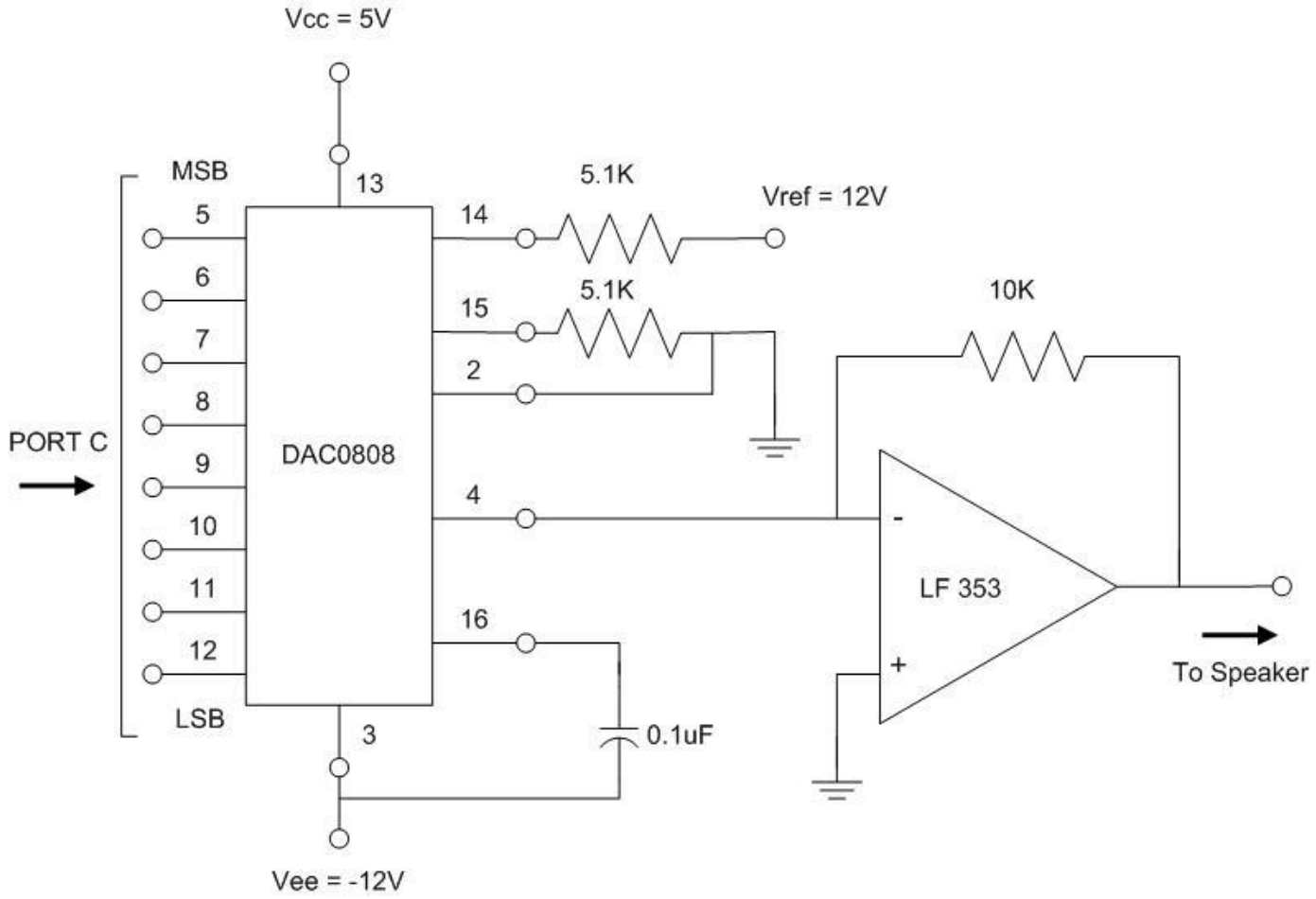
The resolution Q of the ADC:

Q=5/2^(12)=0.0012 V

Q=5/2^(16)=7.6294e-05 V

Digital-to-analog converter (DAC)

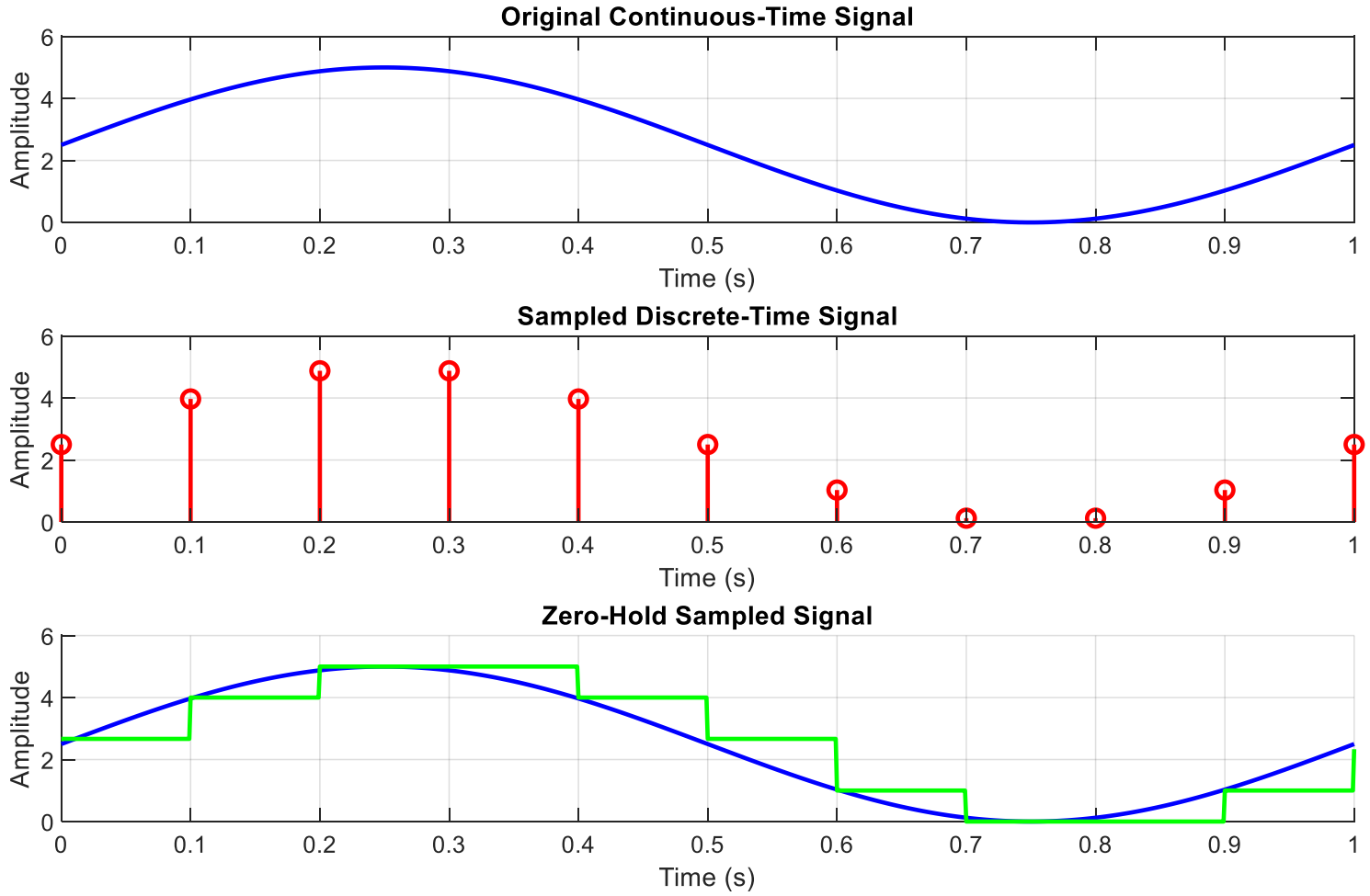
A digital-to-analog converter (DAC) is an electronic device/chip that converts digital signal into an analog signal, performing the reverse function as (ADC) .



$$Q = \frac{E}{2^m}$$

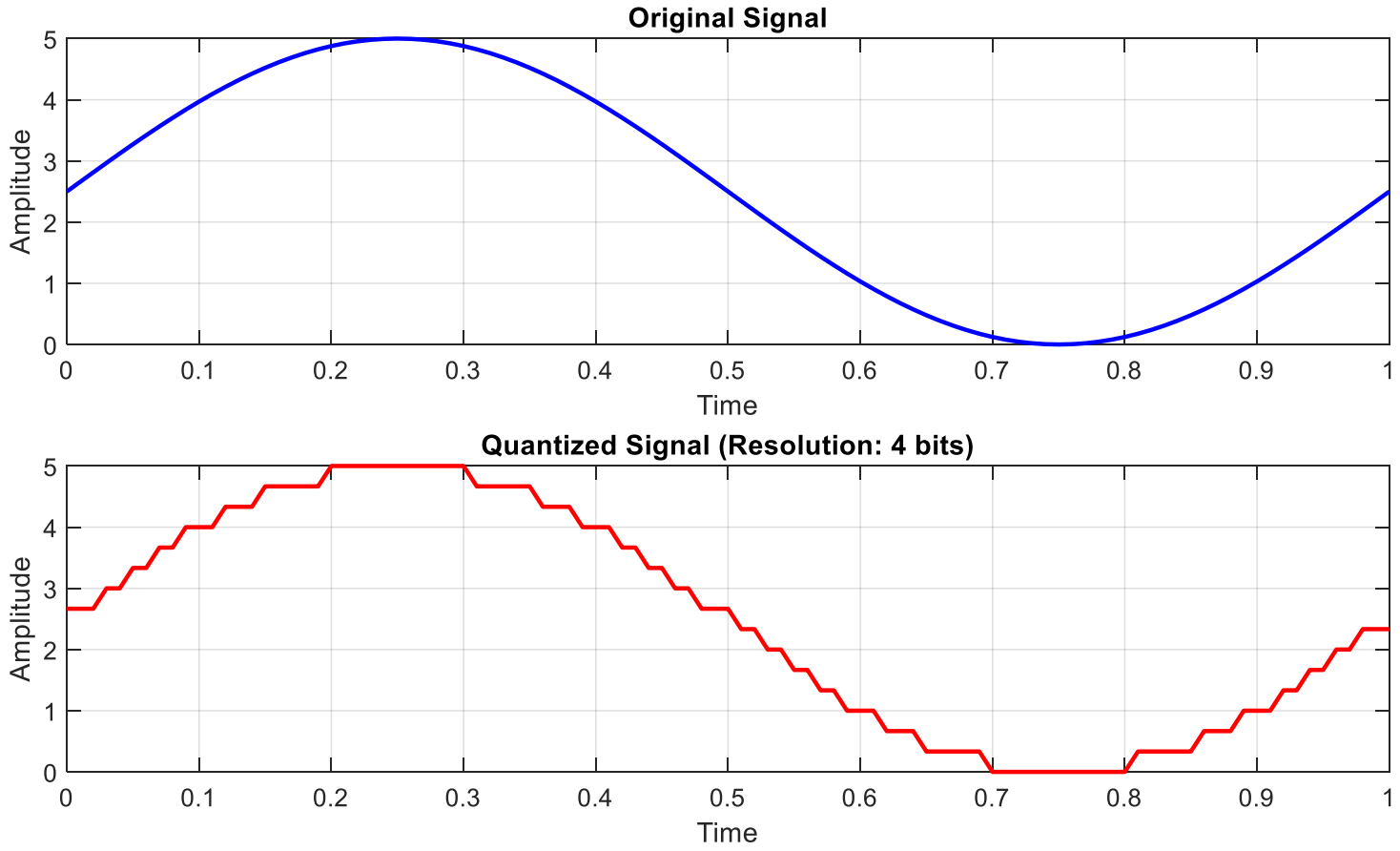
Sampling & quantization

- Example: 4-bit ADC converter; the full range is divided into 16 grids, 1/(2^4), 6.25% per grid. 5 V range has 0.3125 V resolution.

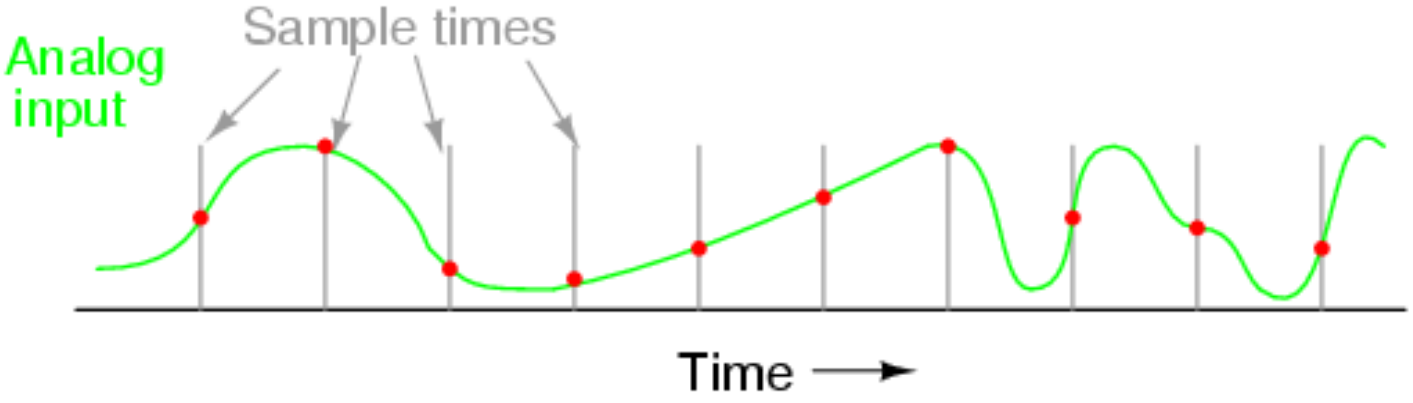


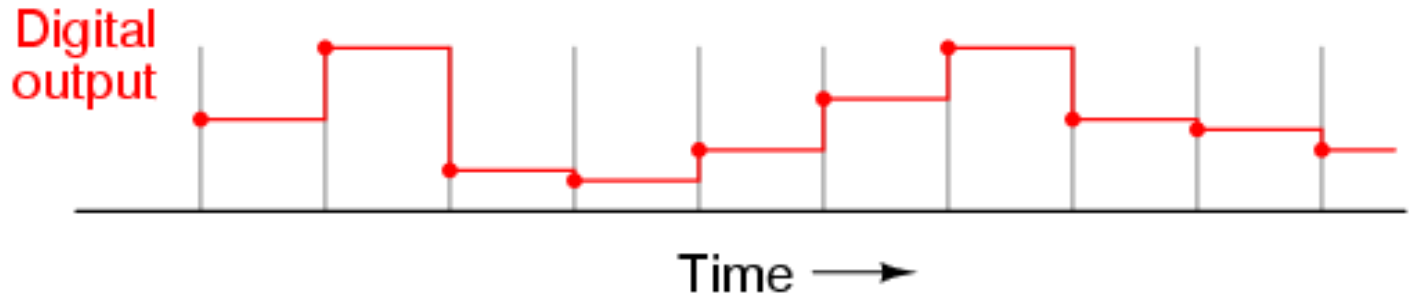
Sampling & quantization

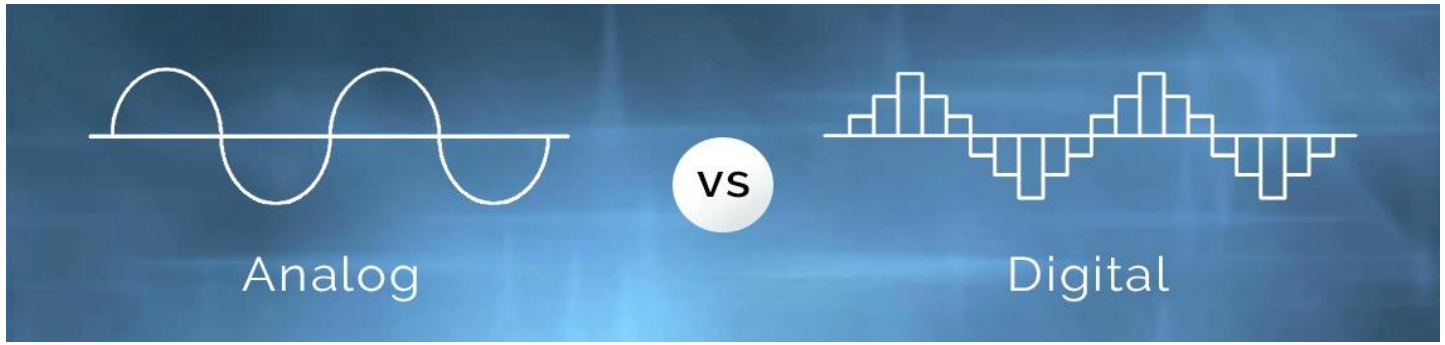
Example: 4-bit ADC converter, if we sample faster, the smoothness is limited by the resolution.



From analog to digital: takeaway messages







Quantization

- Resolution and Quantization Error: The higher the resolution of the A/D converter, the lower the quantization error and the smaller the quantization noise.

- Modern High-Precision Sensors: In modern times, with high-precision sensors and advanced analog-to-digital converters (ADCs), quantization error has become less of a problem.

- Negligible Impact: With modern technology, these errors can be minimized to a level where they have negligible impact on the overall system performance.

Sampling

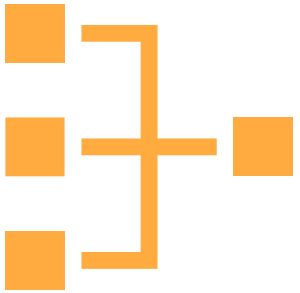
Sampling frequency/rate is crucial in modern high-resolution sensors:

- **Nyquist-Shannon Sampling Theorem**: To avoid loss of information, the sampling frequency should be at least **twice** the highest frequency of the signal, so that it can accurately represent in the digital domain.
- **Resolution and Bandwidth**: The sampling rate determines the maximum frequency that can be accurately captured, which is equal to half the sampling rate (known as the Nyquist frequency). This directly affects the bandwidth and resolution of the digital signal.
- **Accurate Amplitude and Waveform**: The sampling rate is important for determining the maximum amplitude and correct waveform of the signal. To get close to the correct peak amplitude in the time domain, it is important to sample at least 10 times faster than the highest frequency of interest.

Time-Domain to Frequency-Domain: Fundamental concepts

Rethink

How digital physical signals are **composed**?



How to **re-construct** signals? What elements are involved?

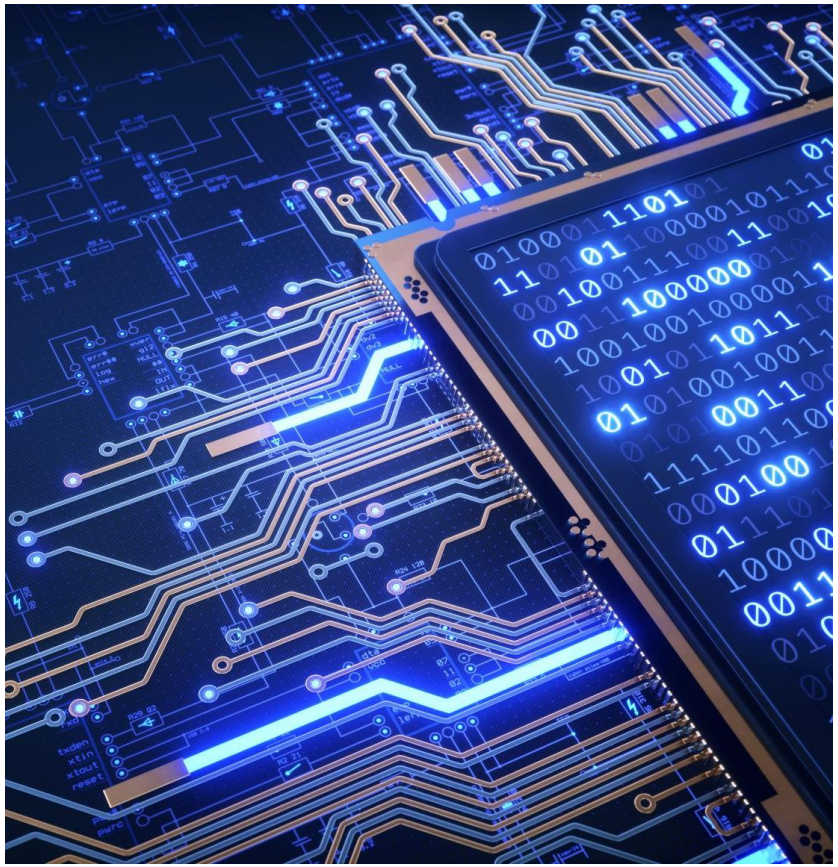
Fundamentals

- How to **re-construct** signals: **Sampling frequency, and resolution (Quantization)**

- How are signals **composed**? **Time domain → frequency domain; Spectral analysis; Fourier Transform**

Definition of Signal

- **Signal**: a physical quantity happens and varies over time, which can be measured in analog or digital form, and carries information we want.
- Signal processing: algorithmic processing of signals to extract **useful information** or improve the quality:
  - Reconstruct the ground truth
  - Infer important information that cannot or hard to be directly measured (**perception**).
- A wide range of applications: control engineering, image and audio processing.



Fourier Series and Fourier Transform

- Any signal can be represented as **a sum of several frequency components**, each with its own amplitude and phase.

- **Fourier series** is a mathematical method to represent **periodic functions** as a sum of sine and cosine functions of different frequencies.

- **Fourier transform** is an extension of Fourier series to non-periodic signals. It transforms a signal **from the time domain to the frequency domain**, where the signal is represented as a sum of complex exponential functions with **different frequencies and amplitudes**.

Time domain and frequency domain

- **Time domain** is the representation of a signal as a function of time.
  - Data plot: plot direct measured physical quantitie over time.
- **Frequency domain** is the representation of the signal as a function of frequency.
  - Data plot: plot the frequency components of the physical quantities over frequency.

Any signal can be represented as **a sum of several frequency components**, each with its own amplitude and phase.

Fourier decomposition of a square wave

- How are signals **composed**? Spectral analysis and Fourier Transform

- Any periodic function can be expressed as **an infinite sum of sinusoidal functions**.

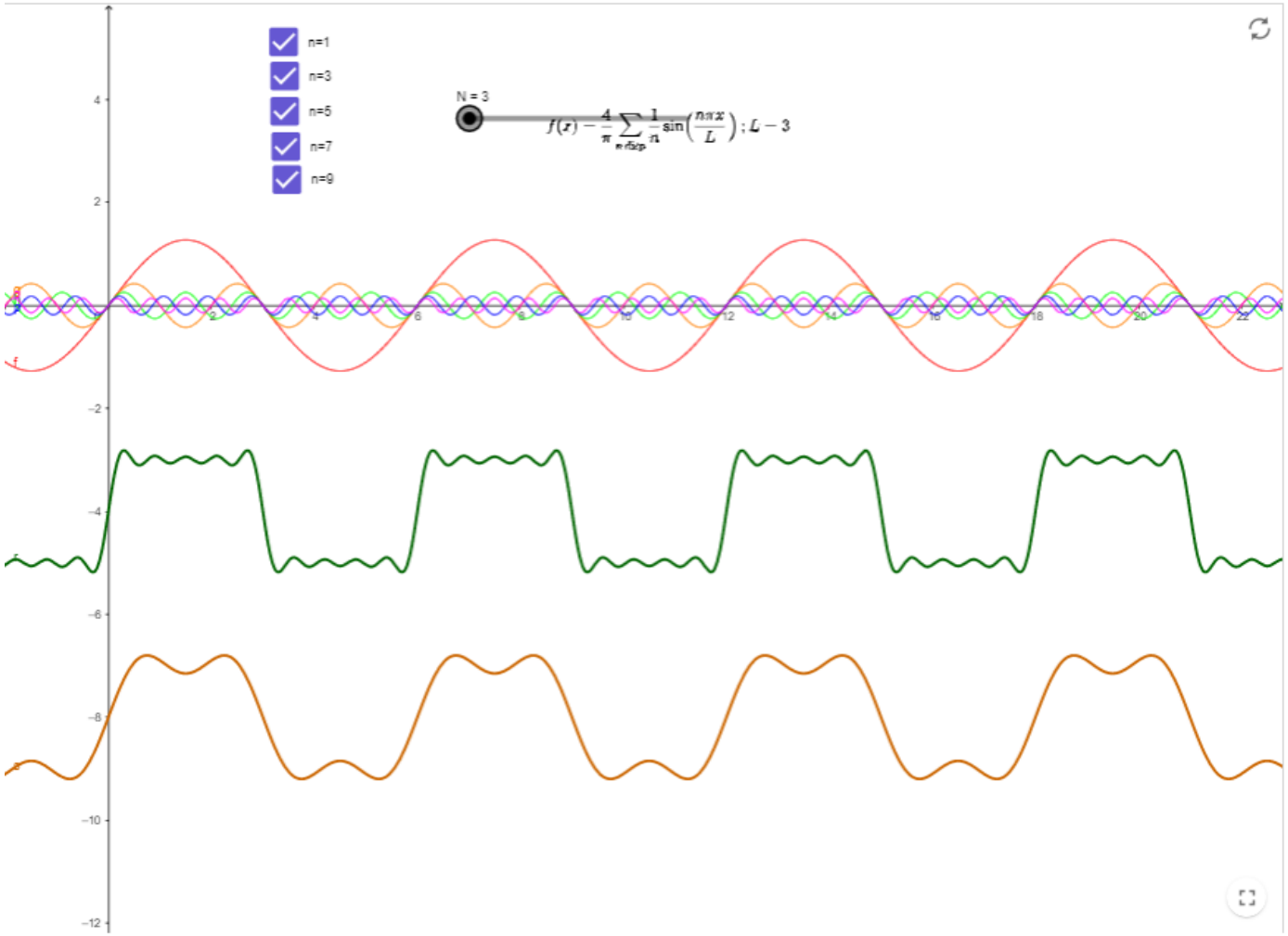
Fourier series of a Square Wave

$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin\left(\frac{n \pi x}{L}\right)$$

Fourier decomposition of a square wave

Fourier series of a Square Wave, N=3

$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin\left(\frac{n\pi x}{L}\right)$$



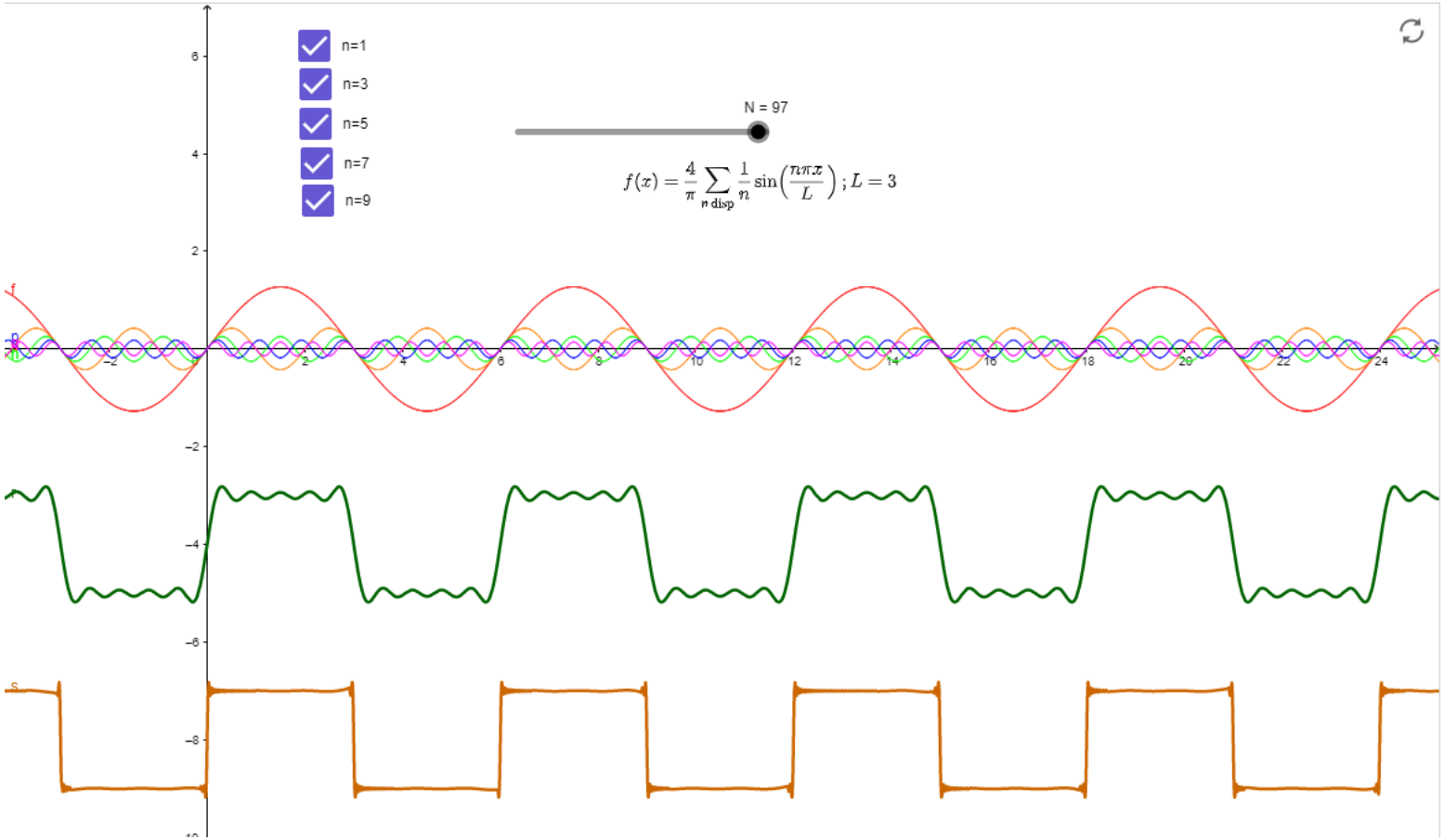
N=5

N=3

Fourier decomposition of a square wave

Fourier series of a Square Wave, N=97

$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin\left(\frac{n\pi x}{L}\right)$$

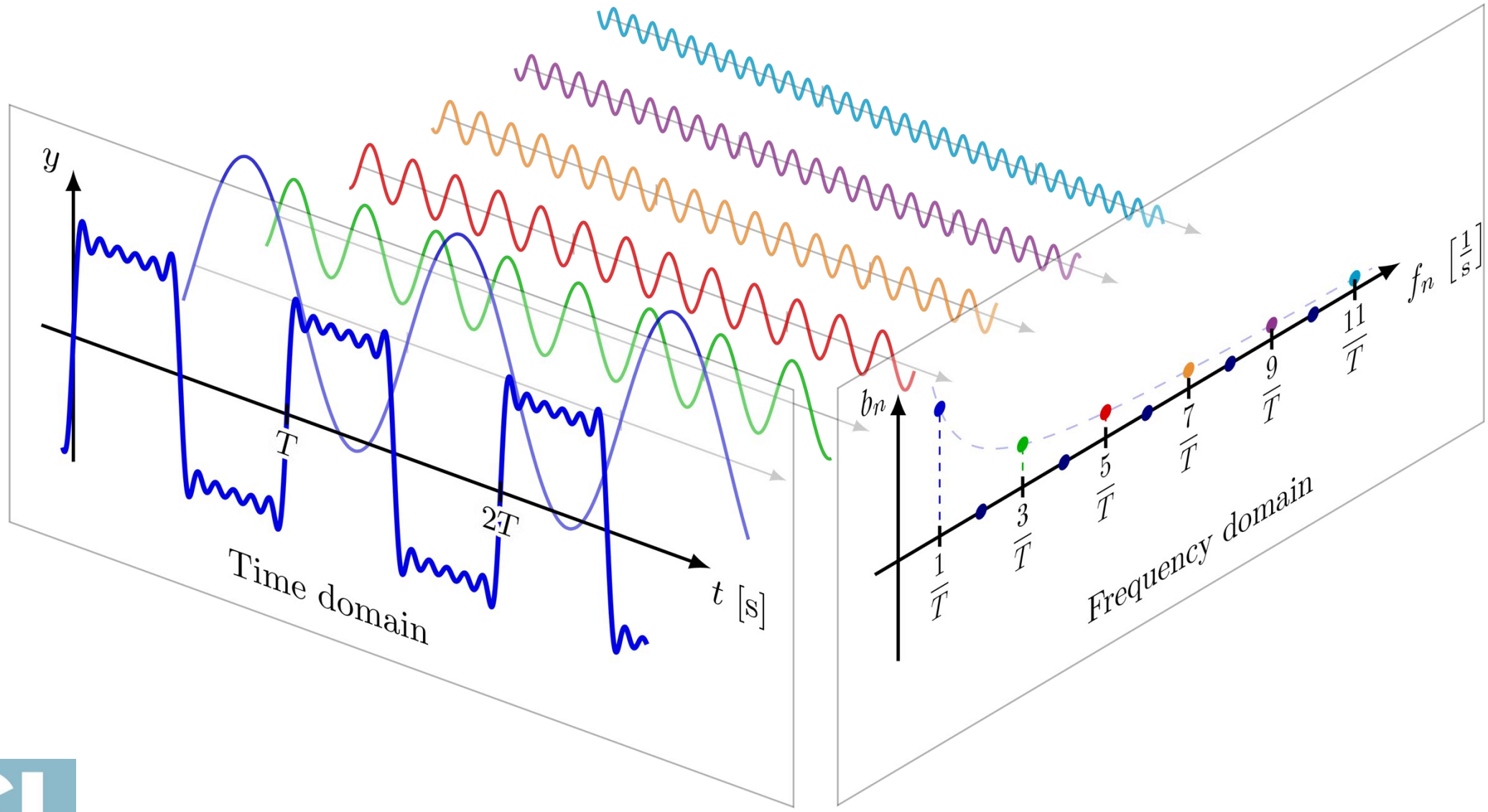


N=5

N=97

Try it out: https://www.geogebra.org/m/wUanseCs

Time domain → frequency domain



Spectral analysis of signals

- Spectral analysis or frequency analysis is specifically designed for periodic signals.

- A complex signal is the combination of a series of simpler frequency components

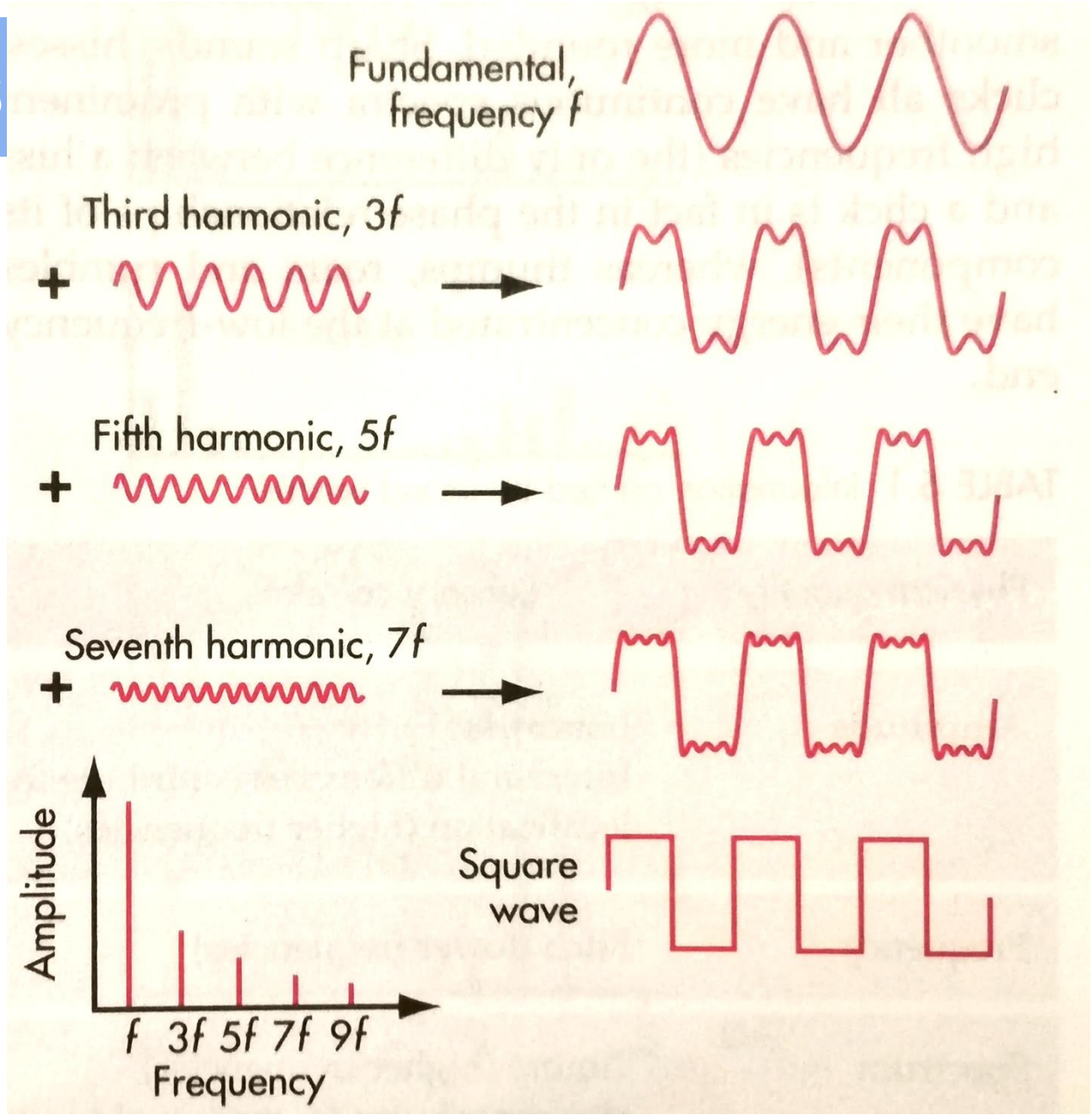


Image source: *Neurophysiology* R.Carpenter

Spectral analysis

Decompose the signal into the “frequency domain”, ie, spectrum of the signal.

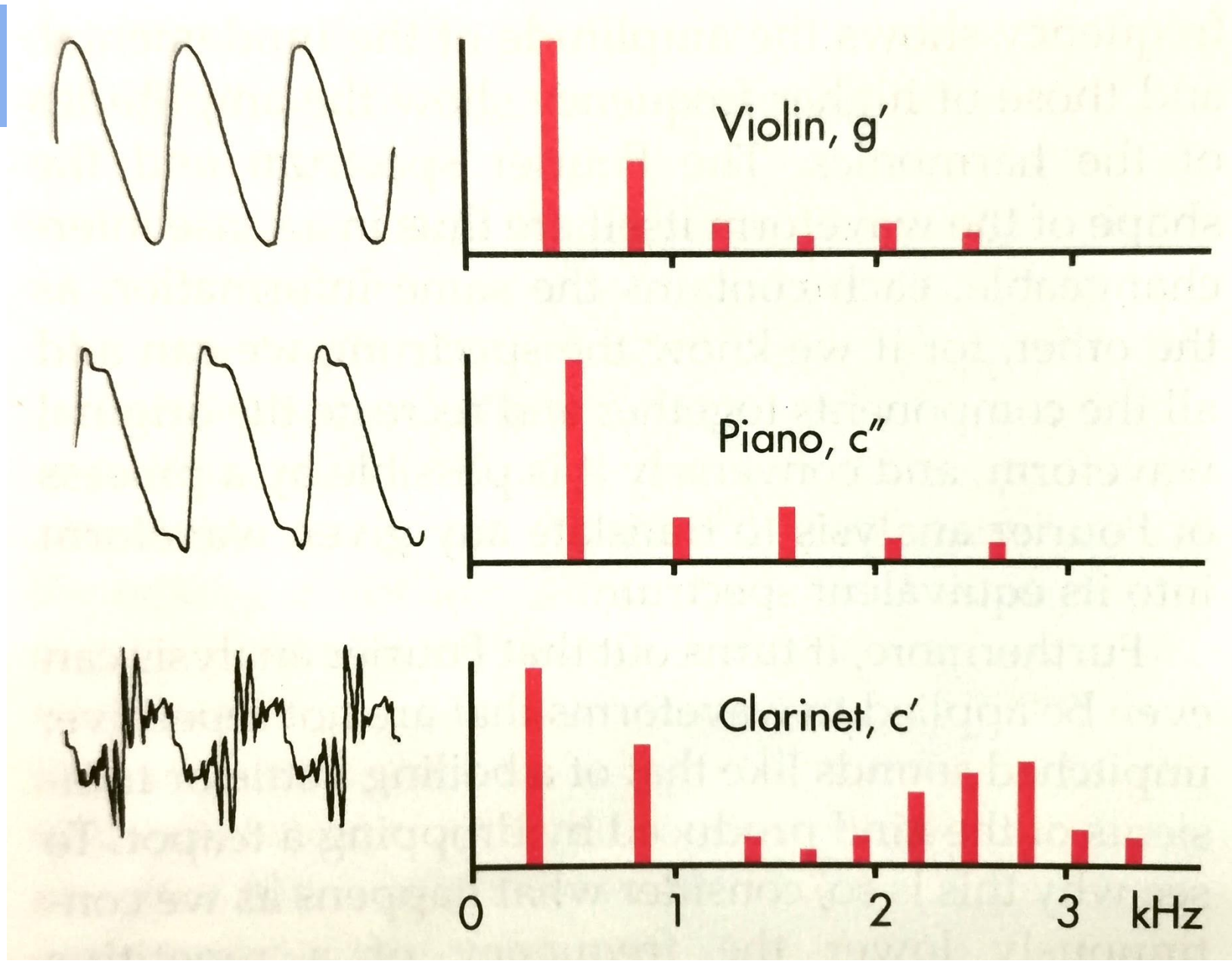


Image source: *Neurophysiology* R.Carpenter

Takeaway messages

- Fourier series: specifically applicable to periodic signals, a mathematical representation that decomposes a periodic function into a sum of sinusoidal or complex exponential functions.

- Fourier Transform:
  - ❑ Applicable to both periodic and non-periodic signals, a more general tool that can be applied to both periodic and aperiodic signals.
  - ❑ Representation of signals in the entire frequency range, converts a time-domain signal into its frequency-domain representation

Takeaway messages

- Signal: a physical quantity happens and varies over time, which can be measured in analog or digital form, and carries information we want. It usually contains signal components of different frequencies.
- Signal processing: algorithmic processing of signals to extract useful information or improve the quality:
  - Reconstruct the ground truth
  - Infer important information that cannot or hard to be directly measured (perception).
- Filtering is the process of removing unwanted components, improving desired components of a signal, by modifying its frequency components.
- Frequency response (output-input relation) is a measure of how a filter responds to different frequencies of input signals.

Digital Filtering

Digital Filtering Techniques

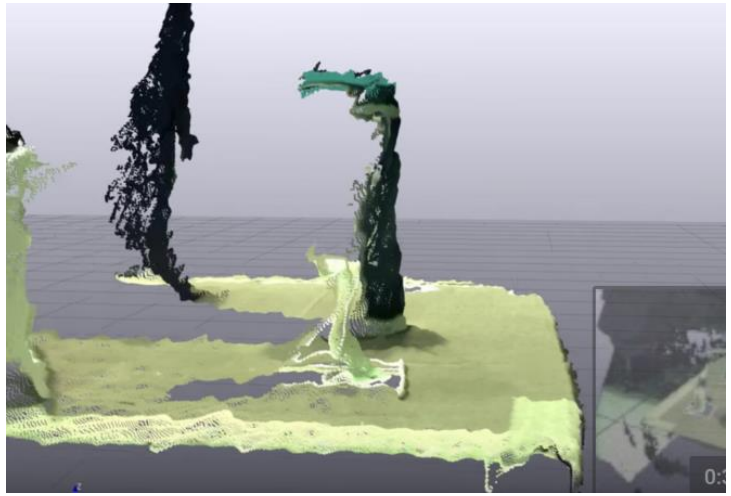
- Fundamental concepts: frequency domain, spectrum analysis of signals
- Digital filters and different types
- Principles of low-pass, high-pass, band-pass filters

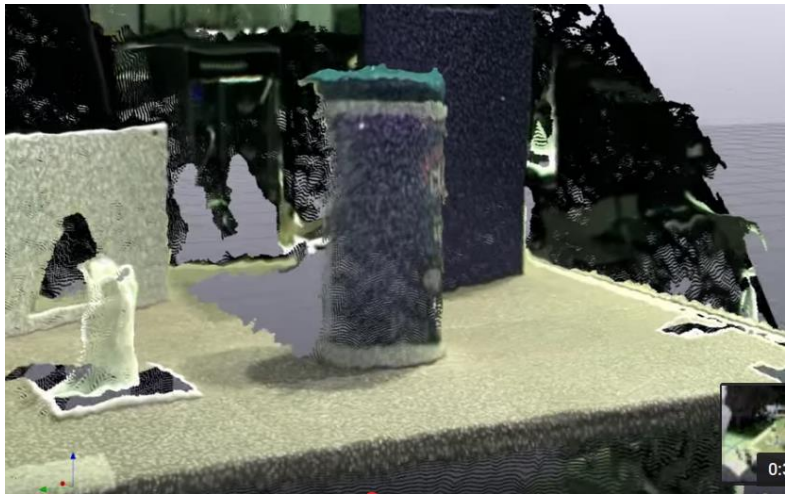
Real world: sensory feedback is imperfect

Consider a real-world scenario: We have a humanoid robot Valkyrie and need to design behaviours using visual perception.

First of all, we need accurate estimates of objects

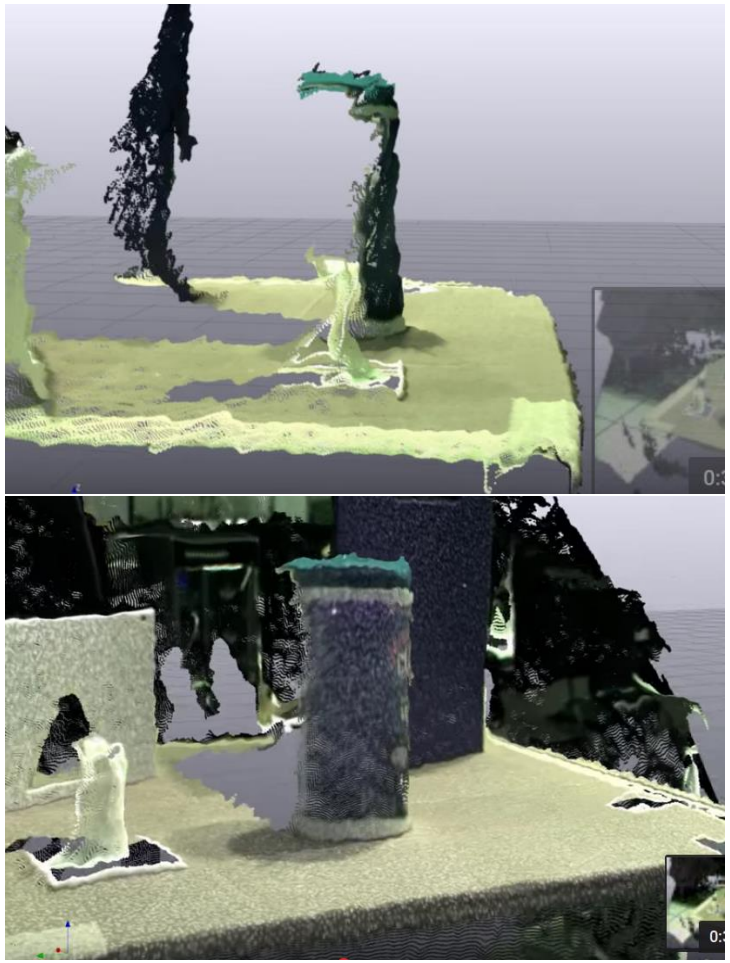
Stereo camera from multi-sense is like this

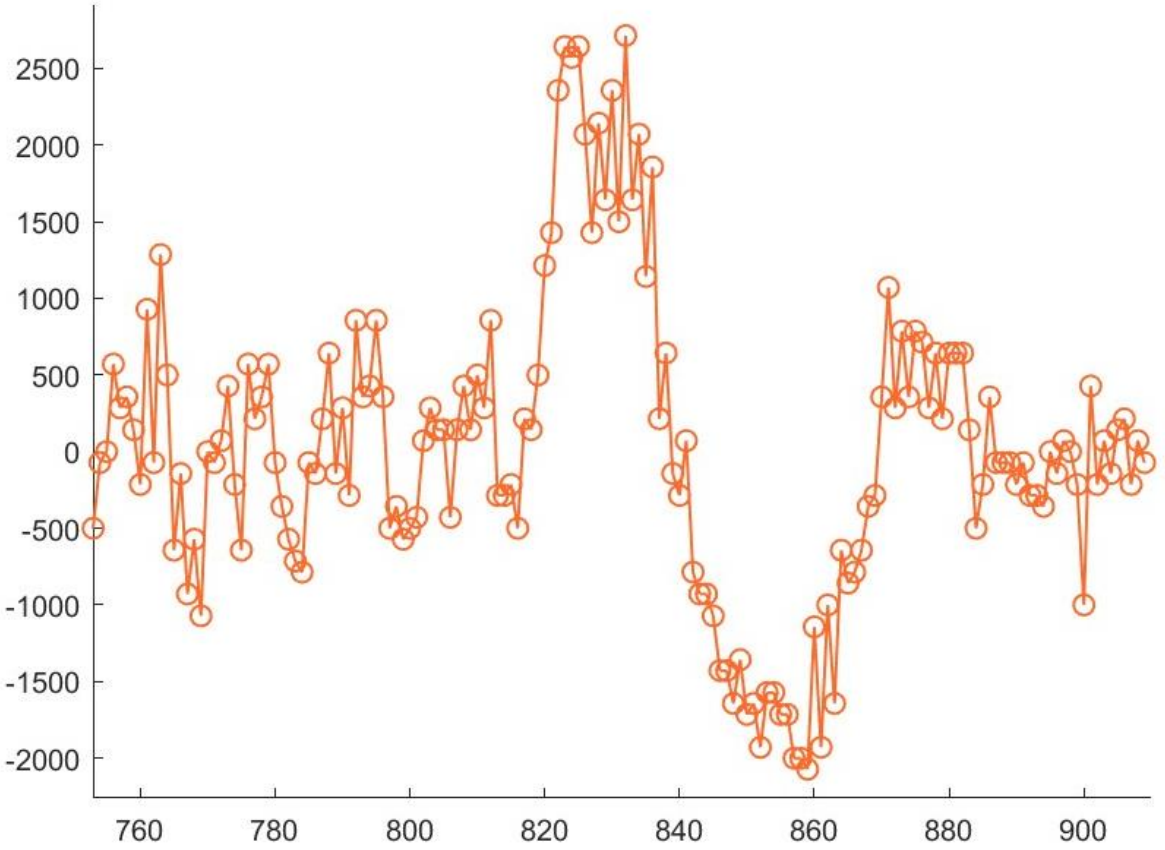




Real world: sensory feedback is not perfect

- Consider real world scenarios:





Stereo **camera** from multi-sense

Encoder measurement from a robot

Real world has noises, uncertainties anywhere

- Sources of uncertainties
- Models are approximations of the reality
- Signals are noisy, drifting
- Sensors have limited accuracy/resolution
- Environment is changing
- Unconsidered factors, eg temperature, humidity, illumination
- Lab/factory environment is more controllable, while real world is **not**

Why filtering?

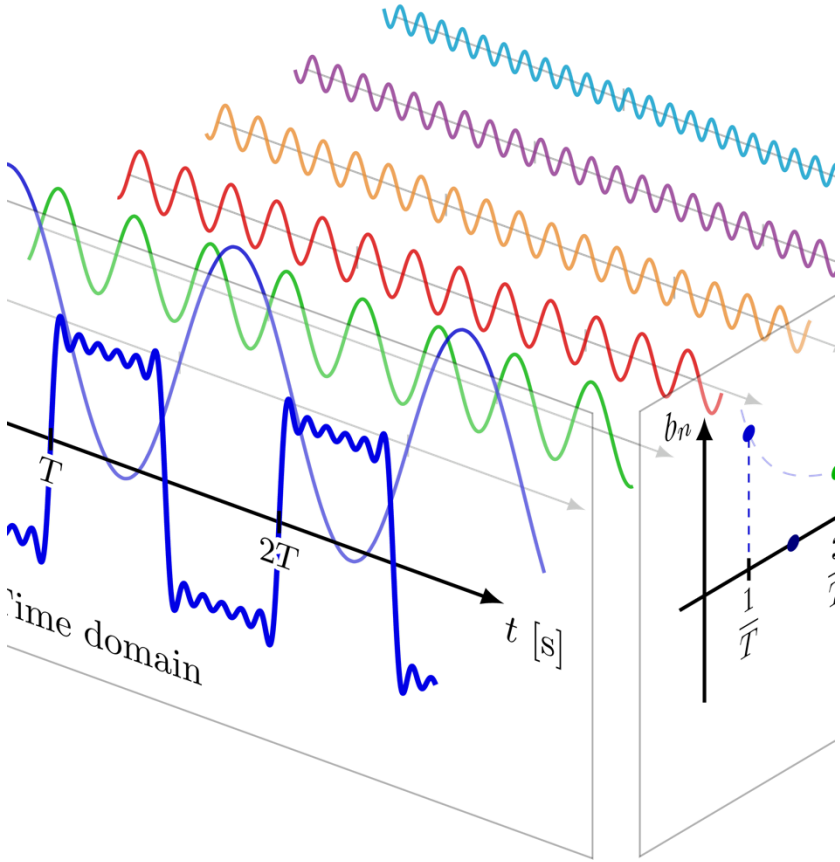
Before using raw sensory measurement, , we need to inspect the **quality of signals** first.

Usually, procedure of signal filtering is required because:

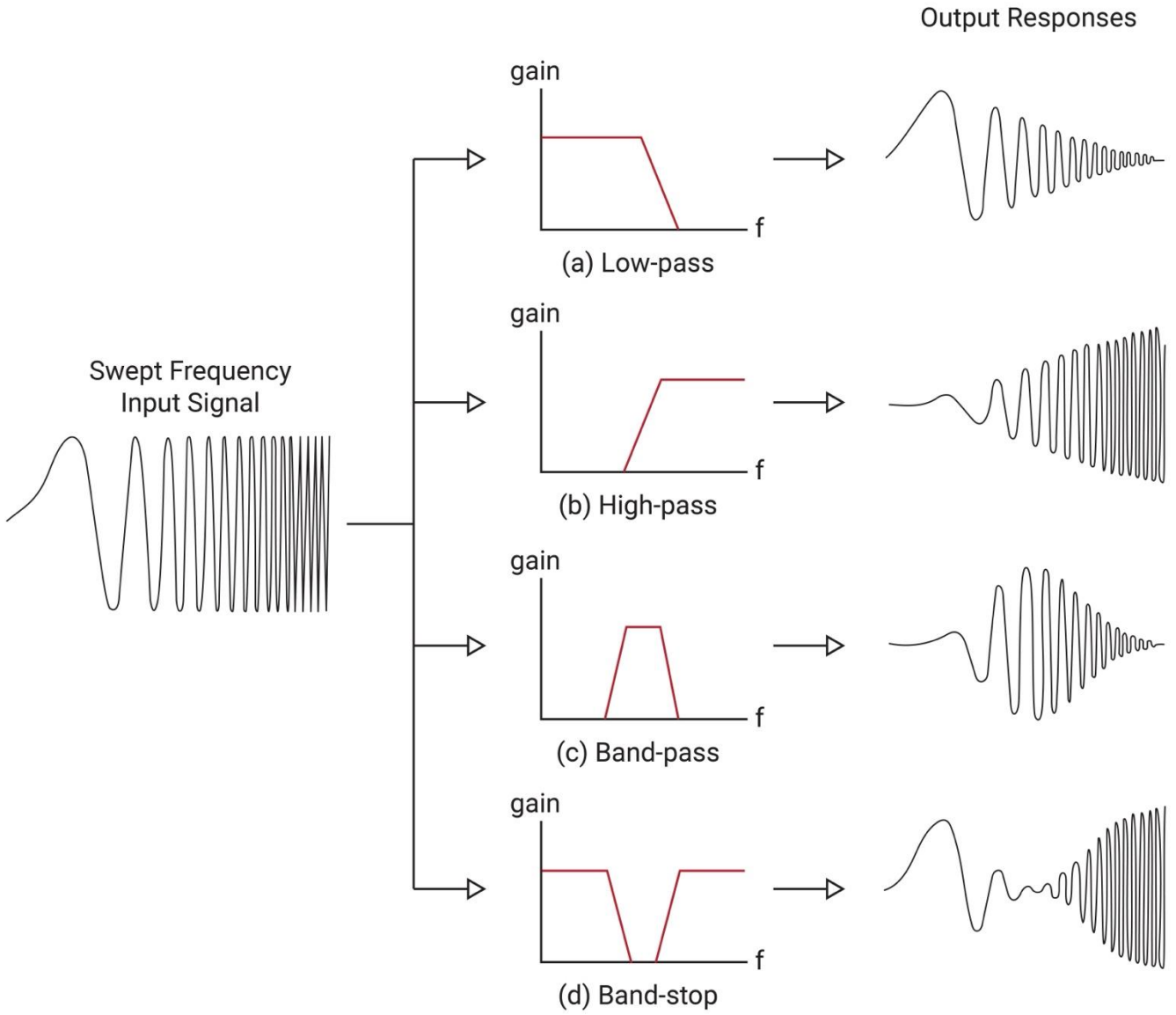
- A direct let-go of noises will jeopardize state estimation algorithm
- Direct use of noisy signals cause instability issues in control
- Necessary pre-filtering can reduce noise to a satisfactory level

Digital filtering

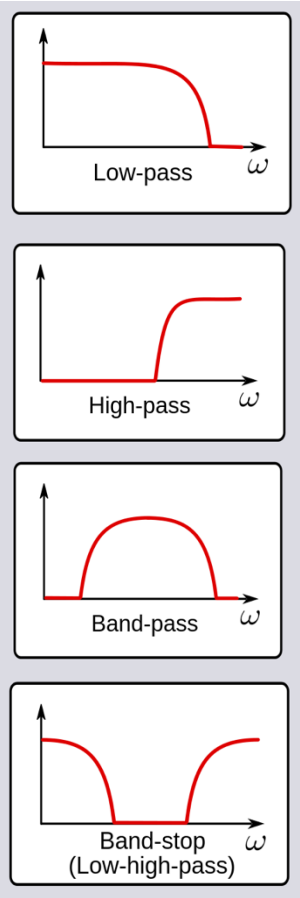
- Filtering is the process of removing unwanted components, improving desired components of a signal, by modifying its frequency components.
- **Frequency response** is a measure of how a filter responds to different frequencies of input signals.
- How it processes signals of varying frequencies: a graph of the system's **output amplitude or power as a function of input frequency**.



Filter effect and response



Common filters used in applications

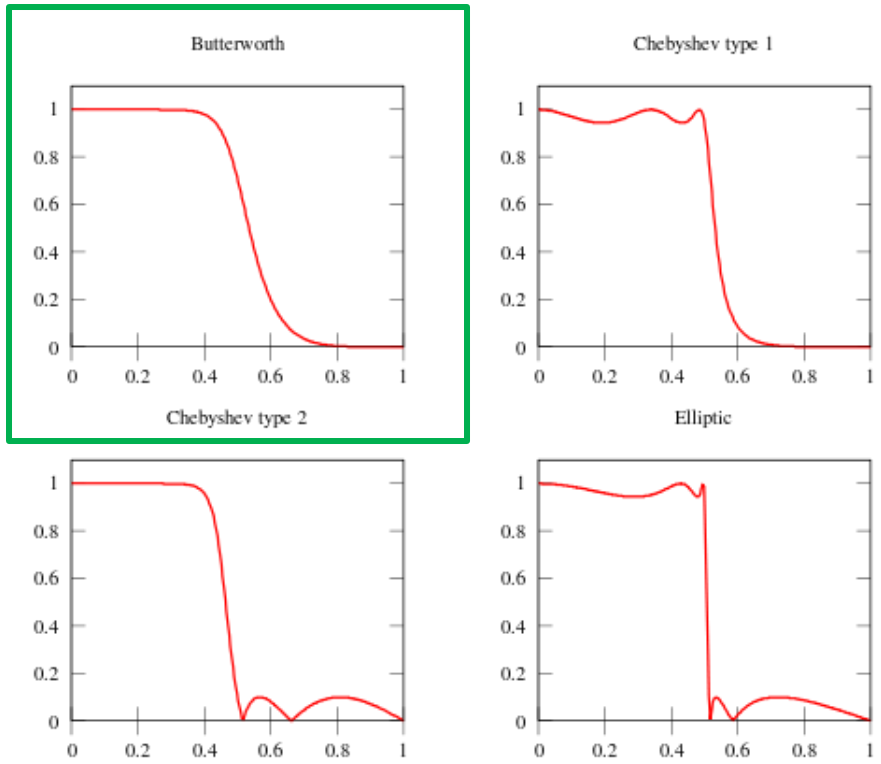


- Low-pass filter: frequencies lower than cut-off frequency are passed
- High-pass filter: frequencies higher than cut-off frequency are passed
- Band-pass filter: only frequencies within a frequency band are passed
- Band-stop filter: only frequencies in a frequency band are attenuated

Common filter tools

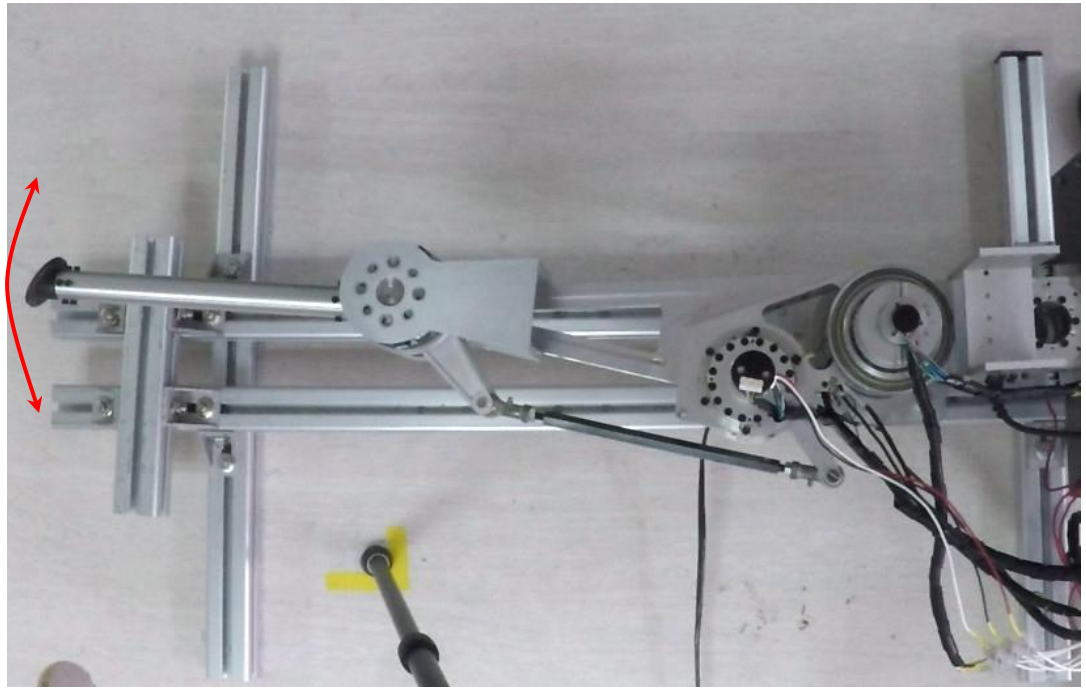
- Low-pass filters are the most common ones used in practice, aim is to remove/reduce noise
- Occasionally, high-pass filters are needed for eliminating low frequency drifts or unwanted offset.
- **Butterworth filter** is the most commonly used.

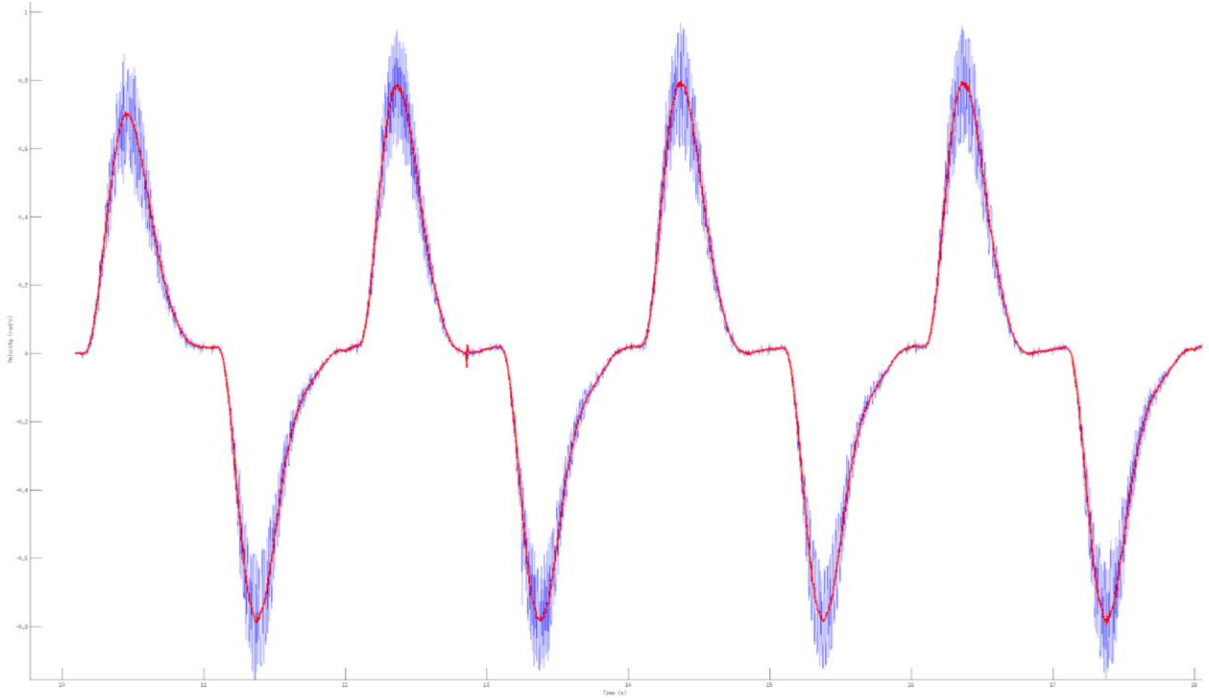
```
function [x_filter]=lowpass(cutoff,x, SamplingTime)
    samplingf=1/SamplingTime;
    fNorm = cutoff / (samplingf/2);
    [b,a] = butter(3, fNorm, 'low');
    x_filter = filtfilt(b, a, x);
end
```



Why filters are needed?

- Signals have noises and the direct use of noisy signals cause instability issues.
- Necessary pre-filtering can reduce noise to a satisfactory level.



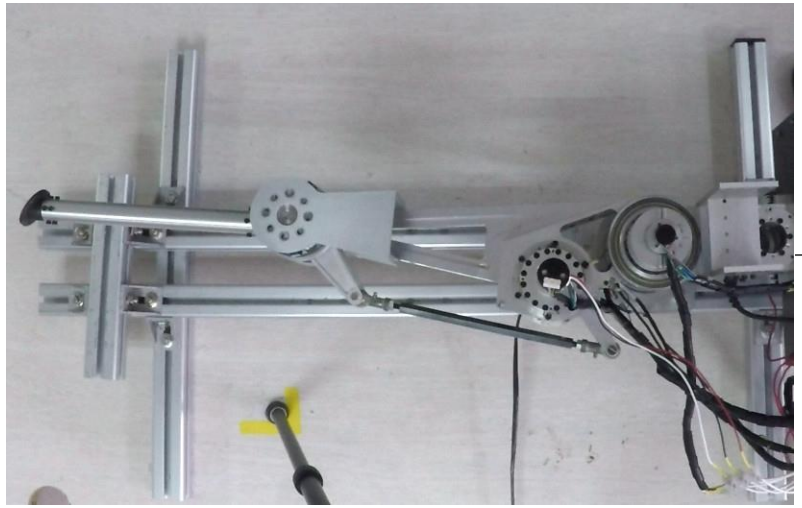


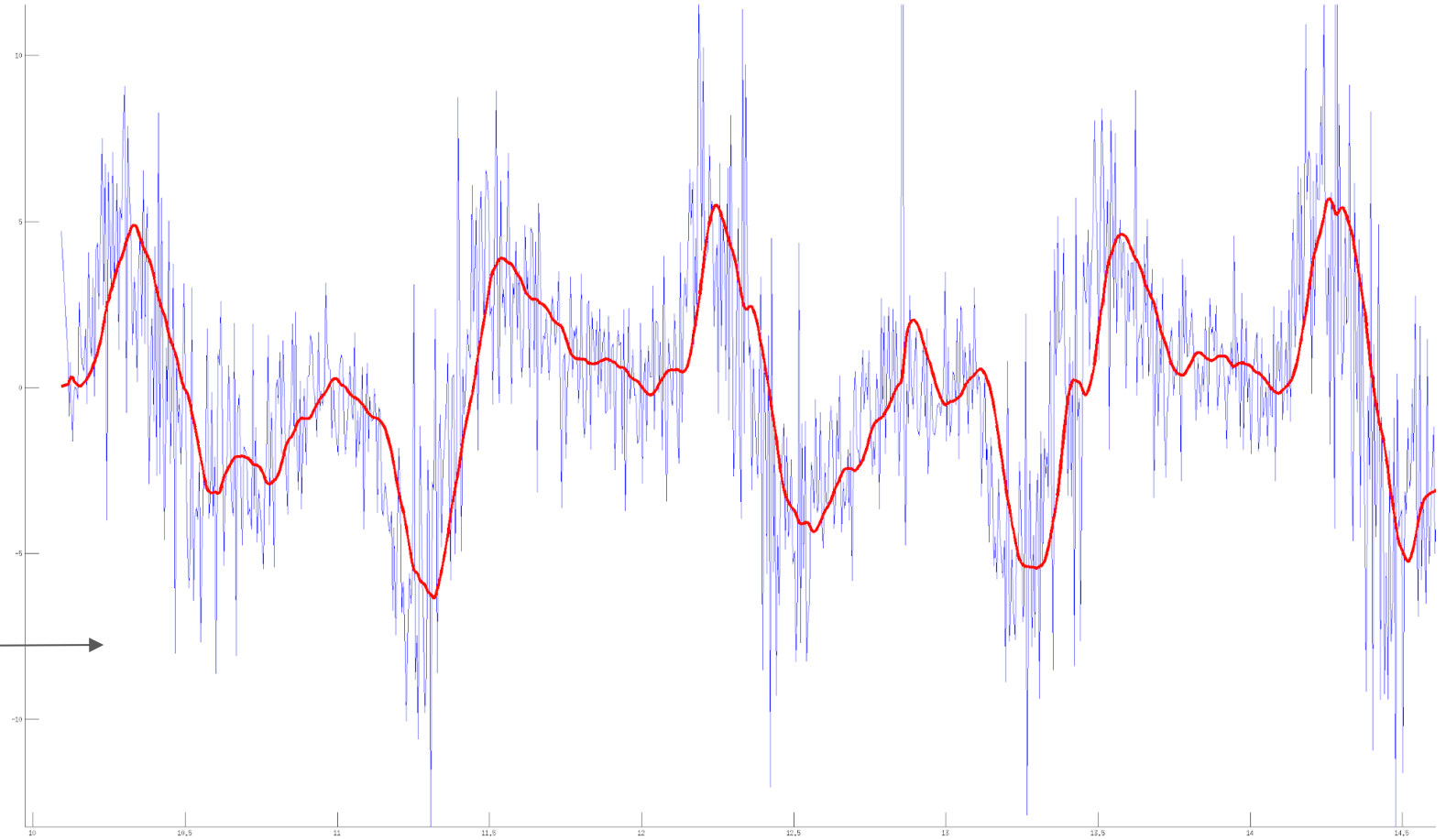
Velocity signal

How real data look like: filtering of link acceleration

Case study:

1. Link encoder resolution of **19-bit**;
2. 2nd order Butterworth filter, cut-off frequency of **5 Hz**.





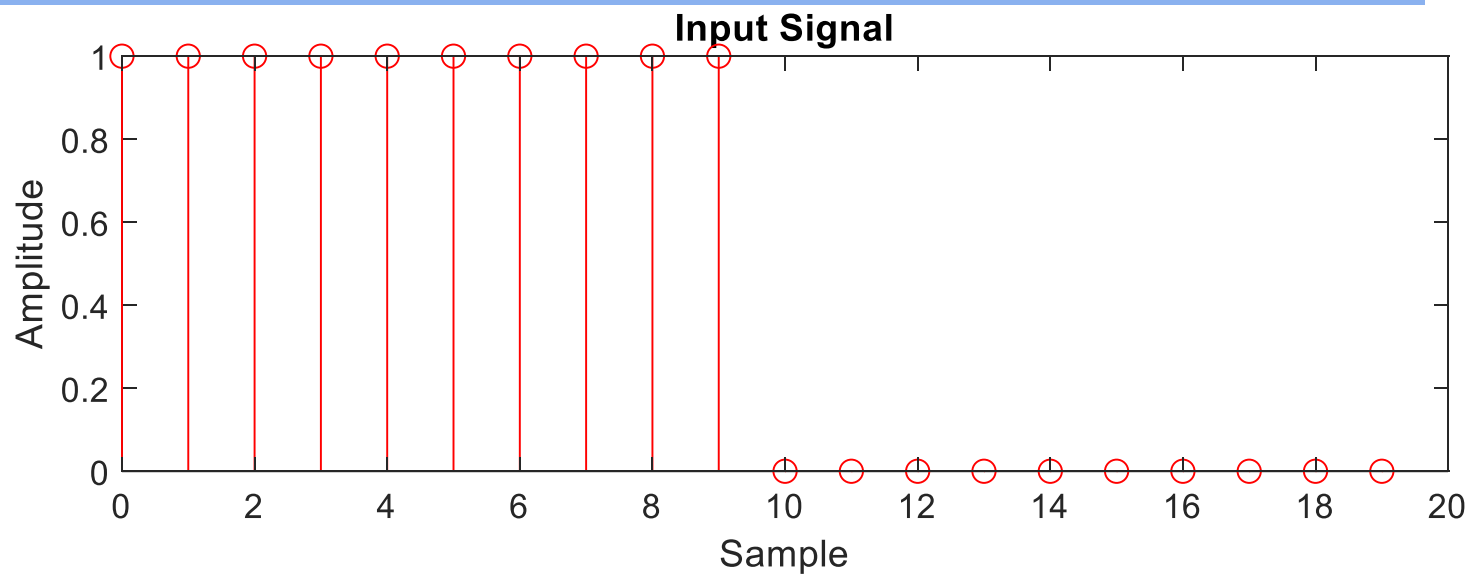
Acceleration signal

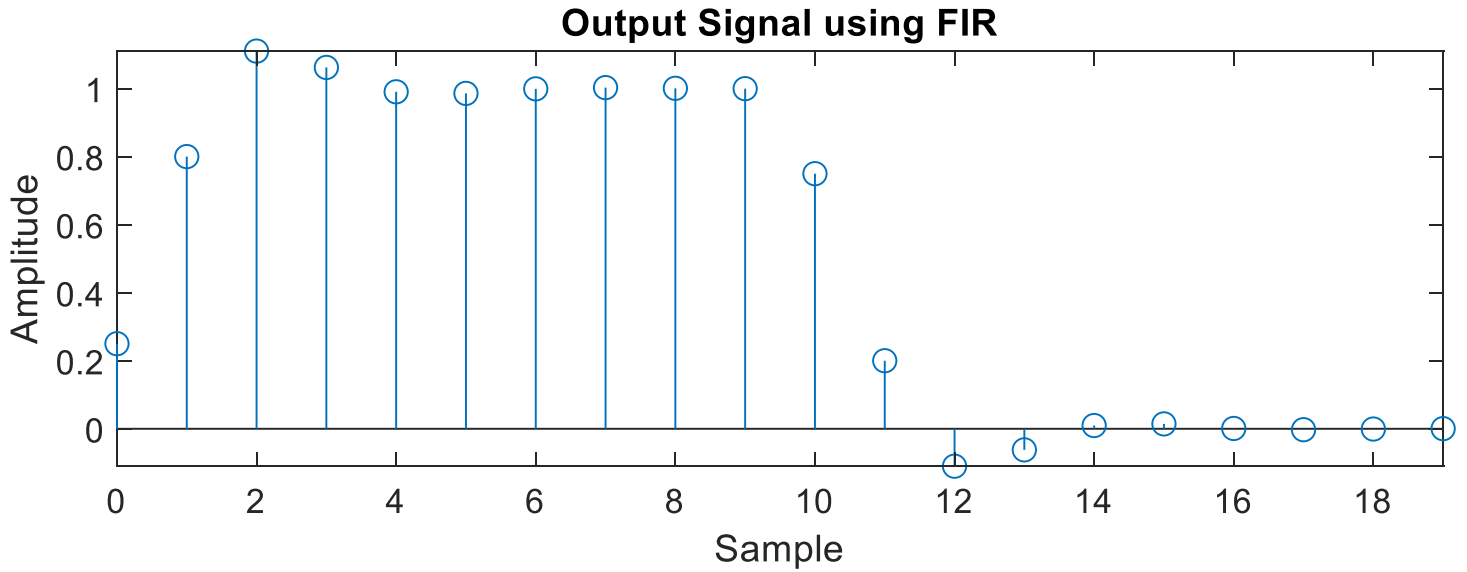
Case study of filter design

IIR example

- Matlab code example

```
b = [0.25, 0.5, 0.25];
a = [1, -0.2, 0.2];

input = [ones(1, 10), zeros(1, 10)];

output = filter(b, a, input);
```



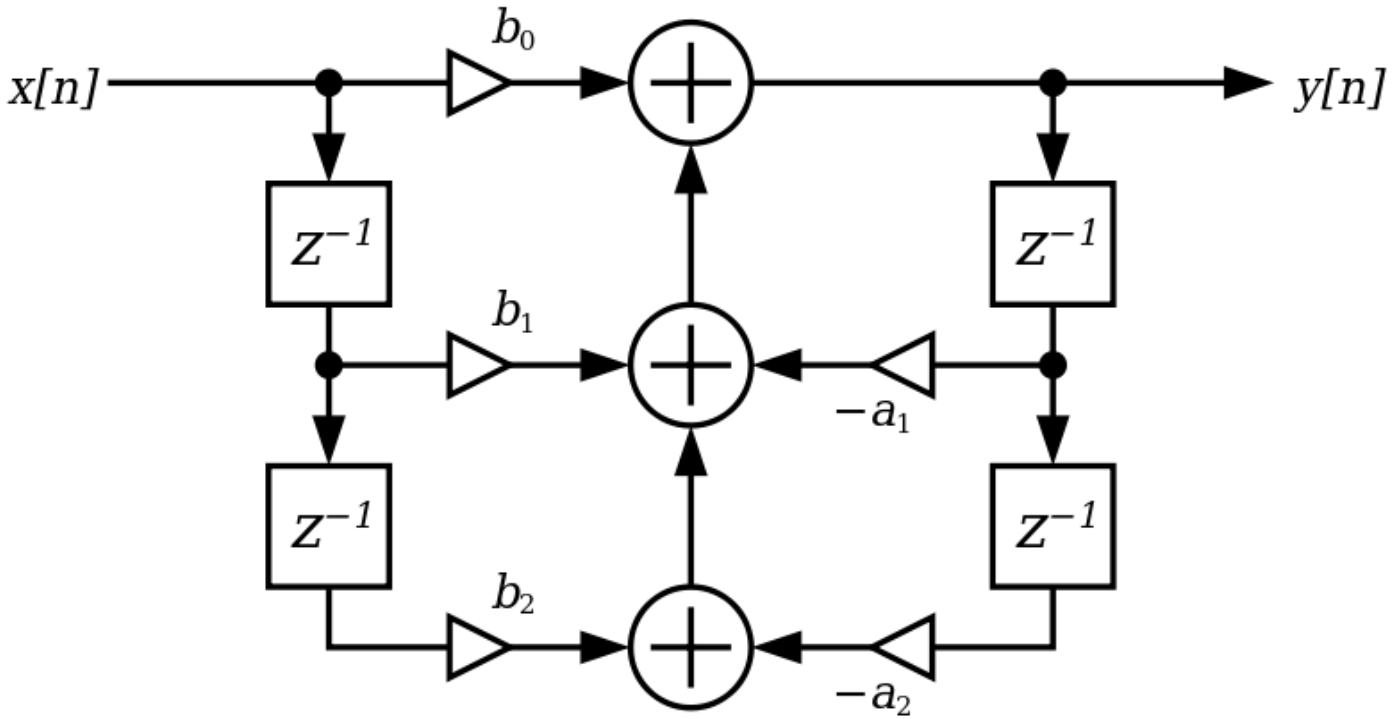


How to change the IIR filter to FIR filter?

- Matlab code example

```
b = [0.25, 0.5, 0.25];

a = [1, -0.2, 0.2];

input = [ones(1, 10), zeros(1, 10)];

output = filter(b, a, input);
```

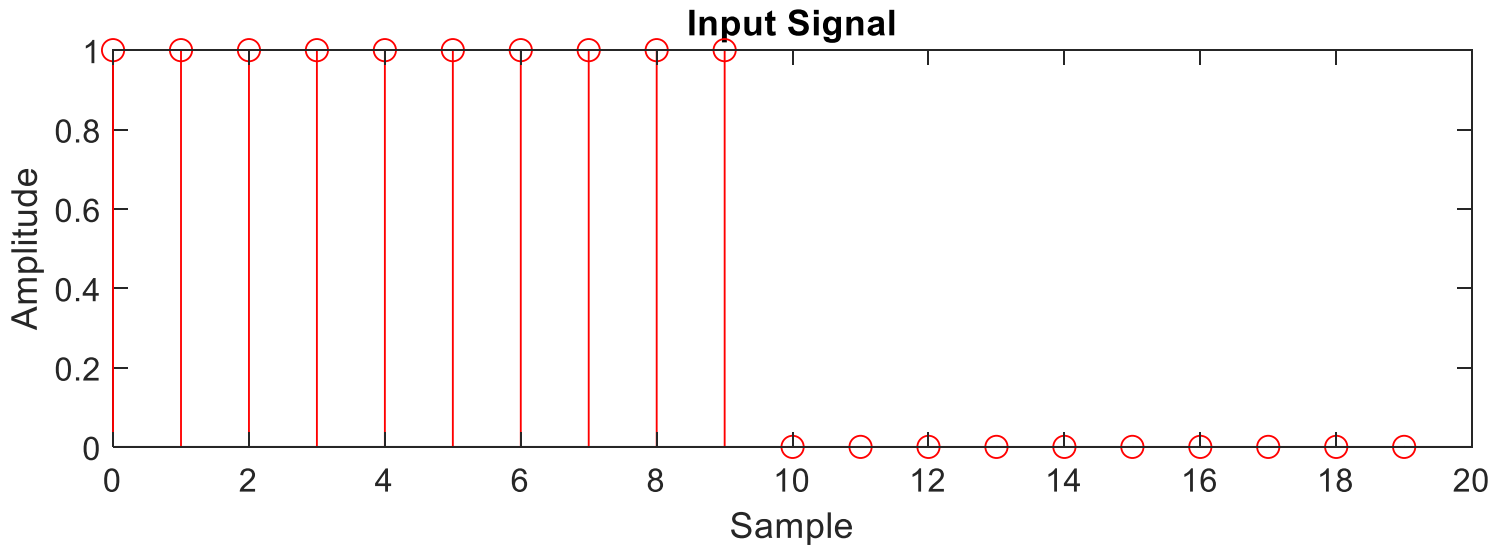
$$\frac{Y(z)}{X(z)}$$

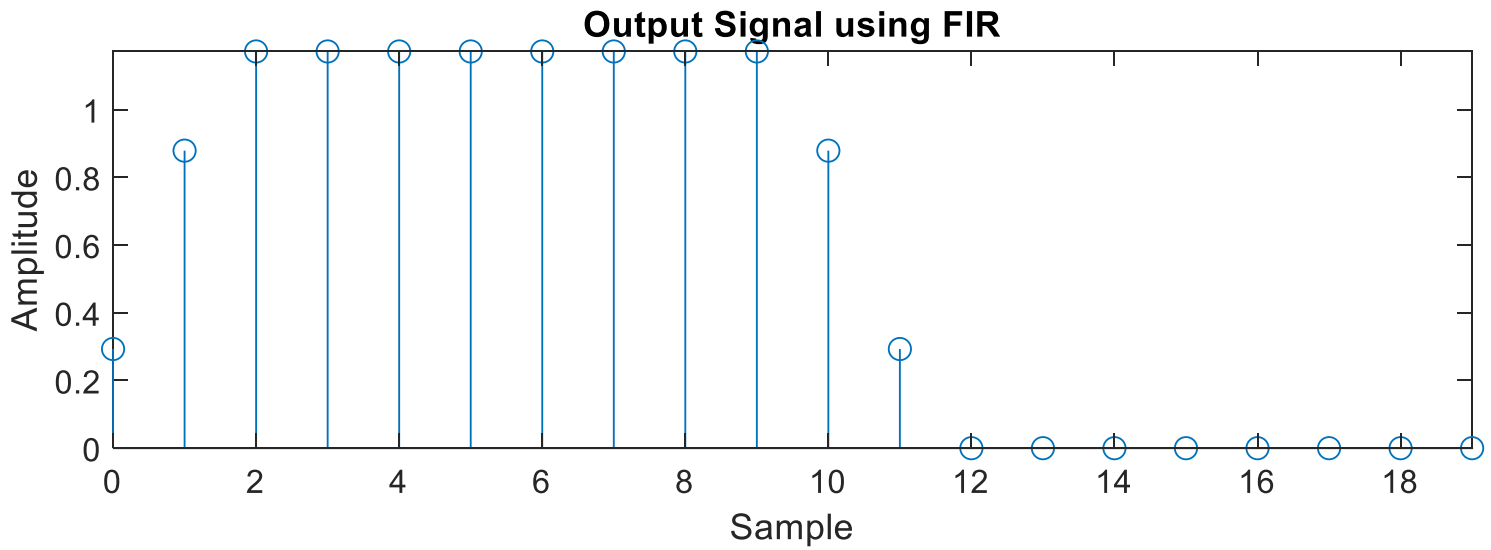


How to change the IIR filter to FIR filter?

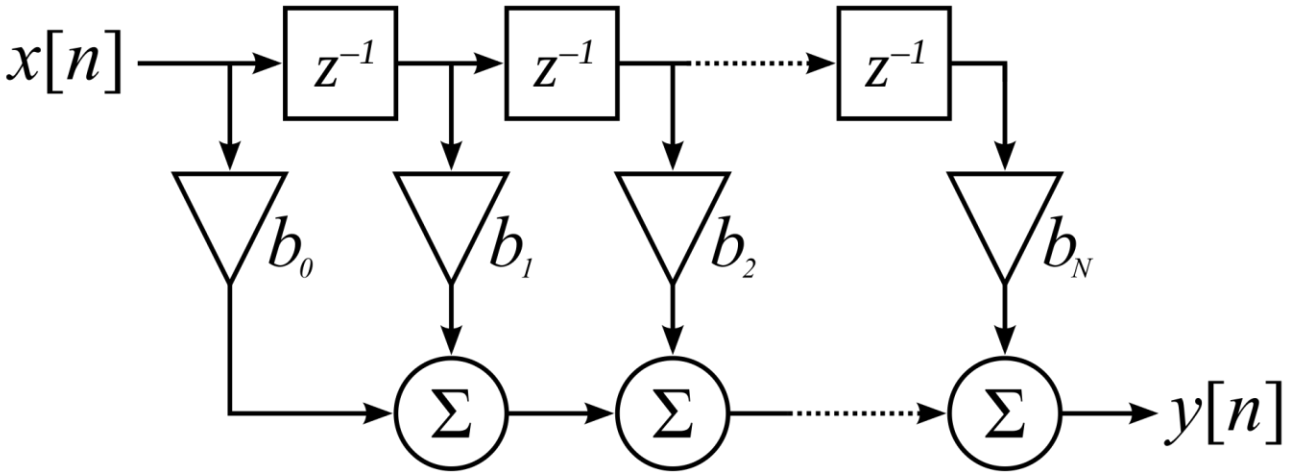
- Matlab code example

```
b = [0.25, 0.5, 0.25];
a = [1, 0, 0];

input = [ones(1, 10), zeros(1, 10)];

output = filter(b, a, input);
```



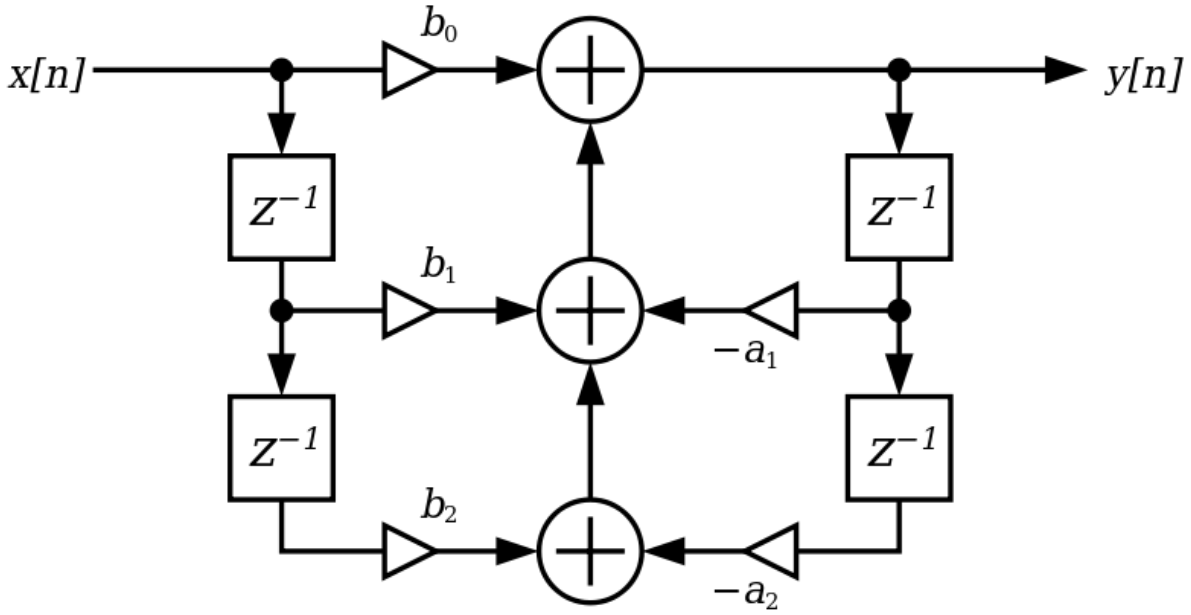


Difference between IIR and FIR filters



- A practical design choice: Why is FIR better than IIR filter?
- **Stability**: As FIRs do not use previous output values to compute their present output, i.e. no feedback from the output, they can never become unstable for any type of input signal -- a distinct advantage over IIR filters.

Difference between IIR and FIR filters

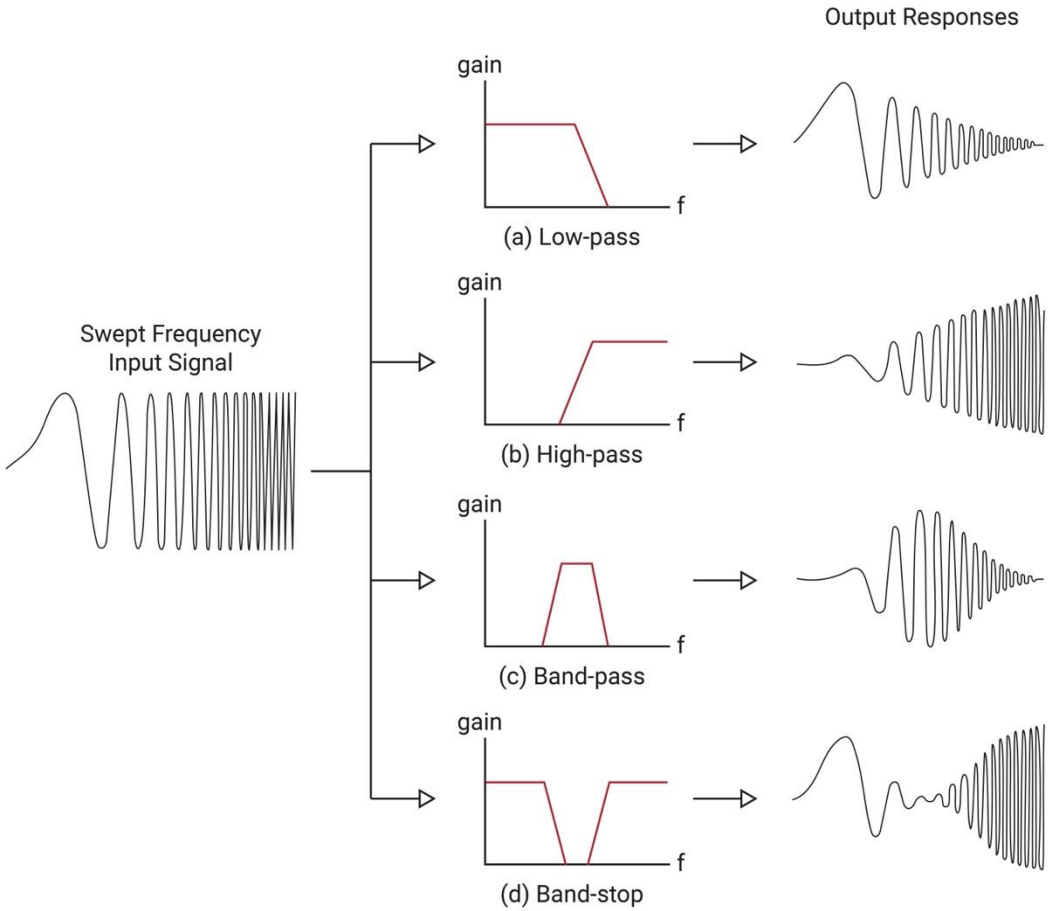


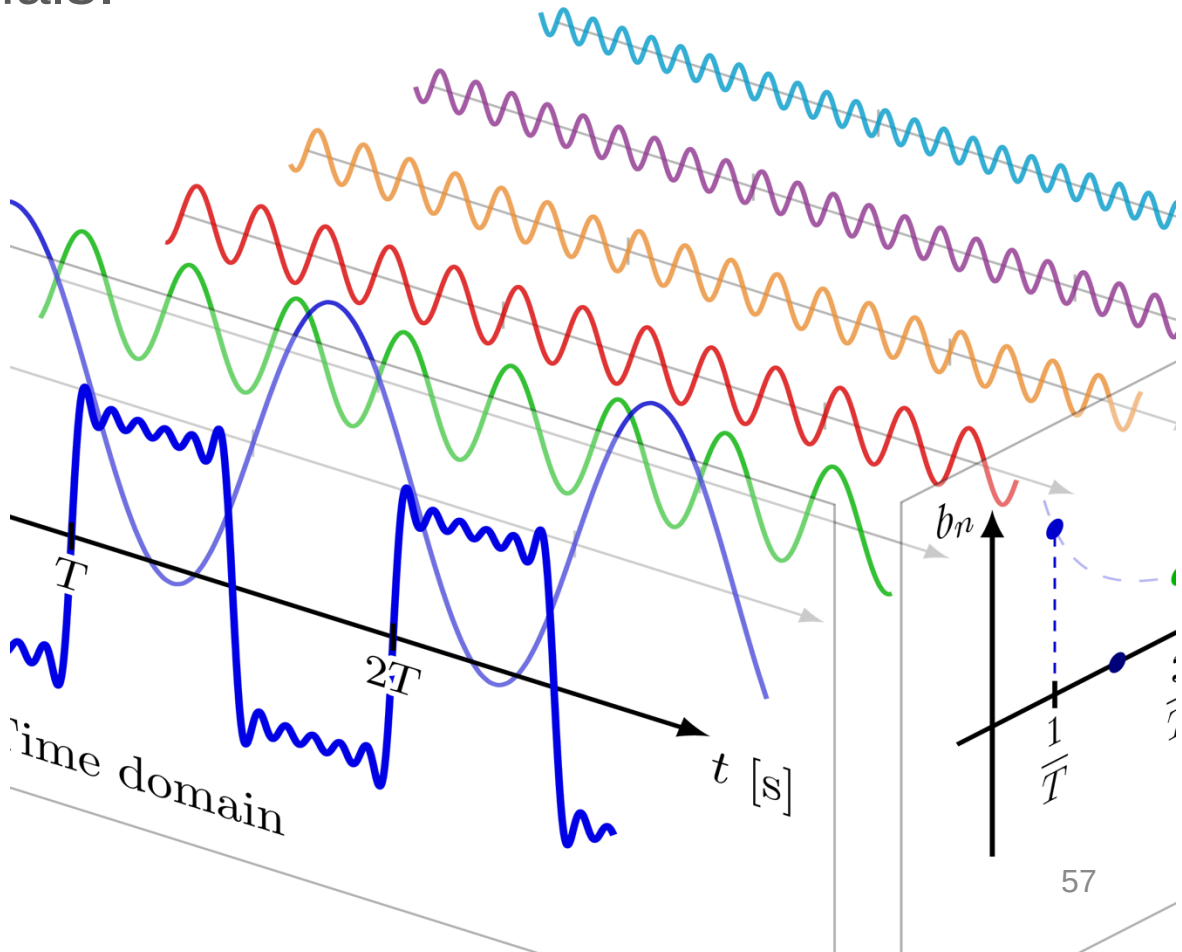
- Pros and cons with IIR filter?
- **Stability**: FIR's advantage over IIR filters.

Force sensor

Digital filtering

- Filtering: remove unwanted components, improve desired components of a signal, by **modifying its frequency components**.
- **Frequency response** is a measure of how a filter responds to different frequencies of input signals.



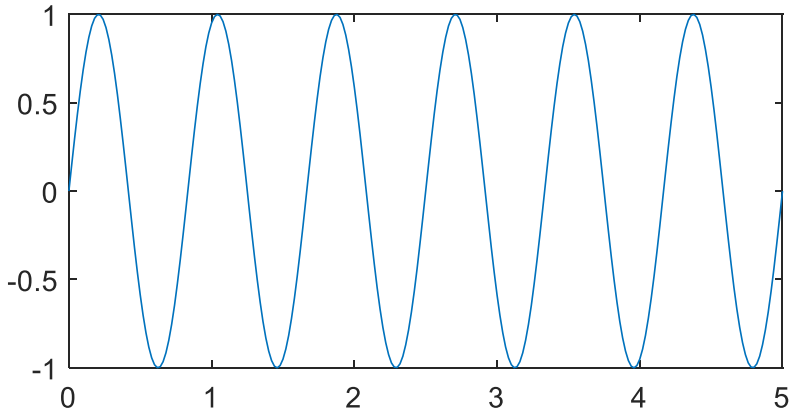


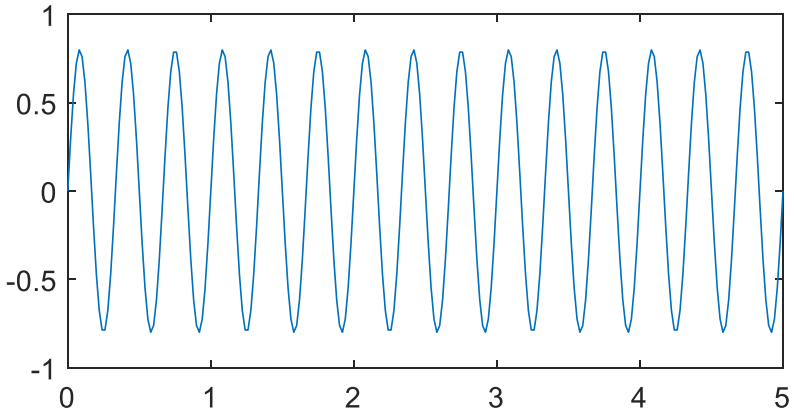
Lowpass filtering to denoise

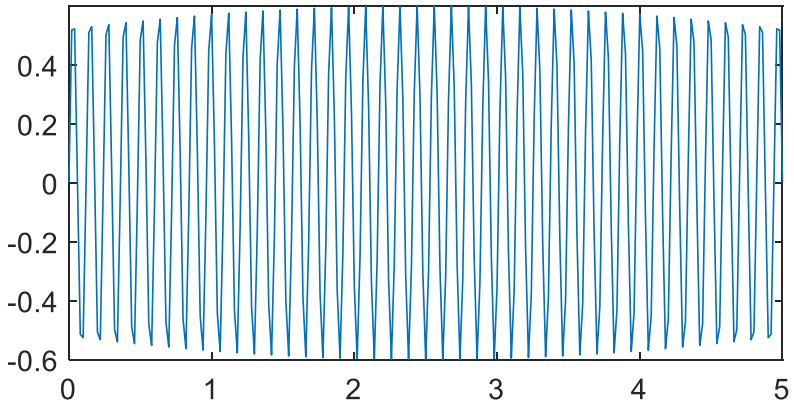
Necessary pre-filtering to **reduce noise to a satisfactory level**

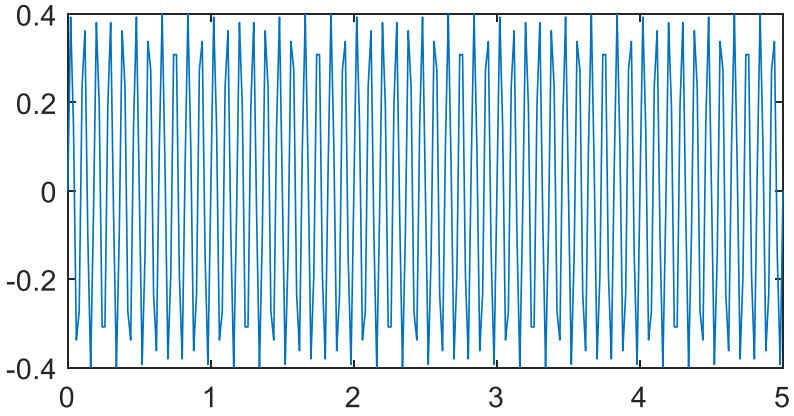
Code design of lowpass filters

- Signal: data_raw = sin(**1.2***2*pi*t) + 0.8*sin(**3***2*pi*t) + 0.6*sin(**8.3***2*pi*t) + 0.4*sin(**11.0***2*pi*t);

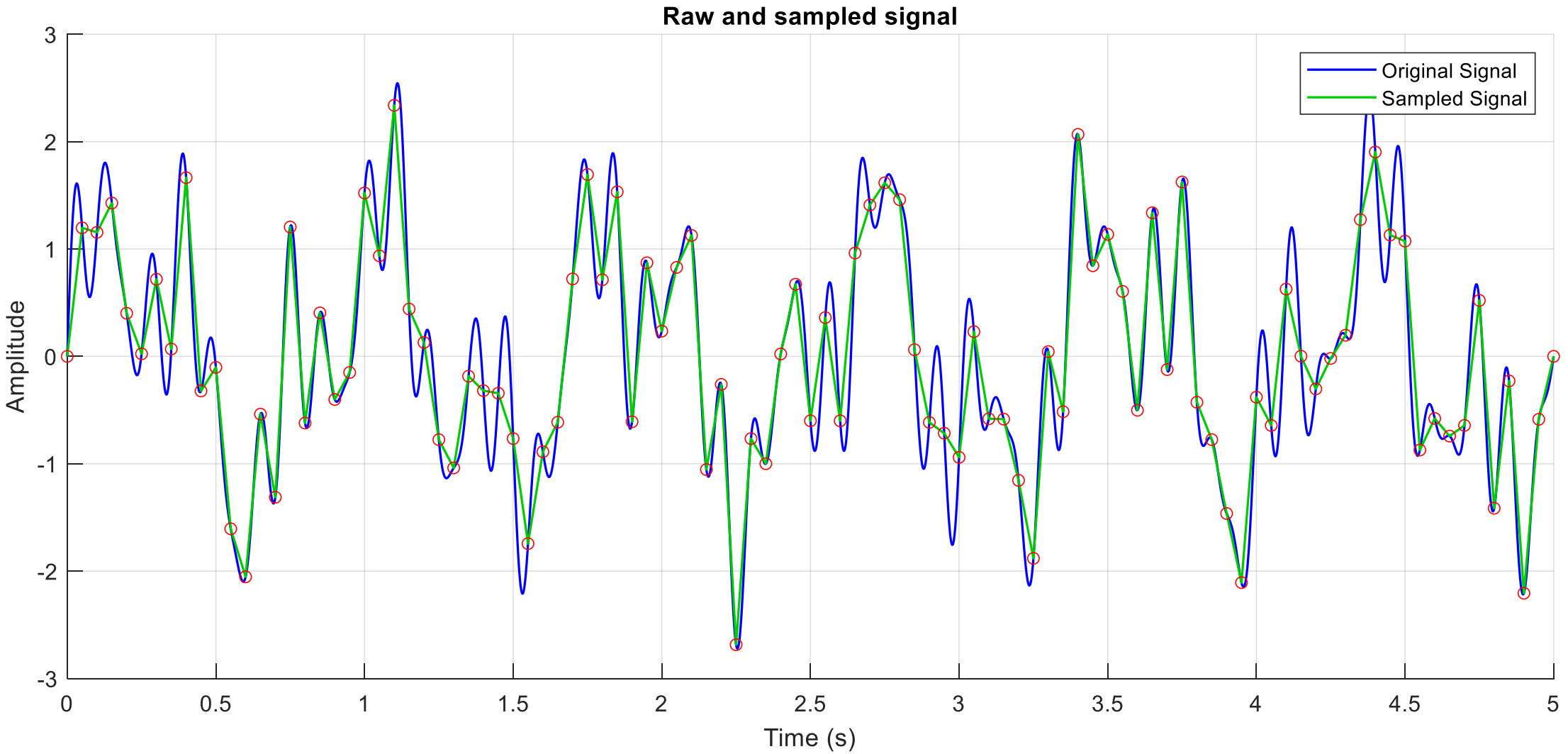




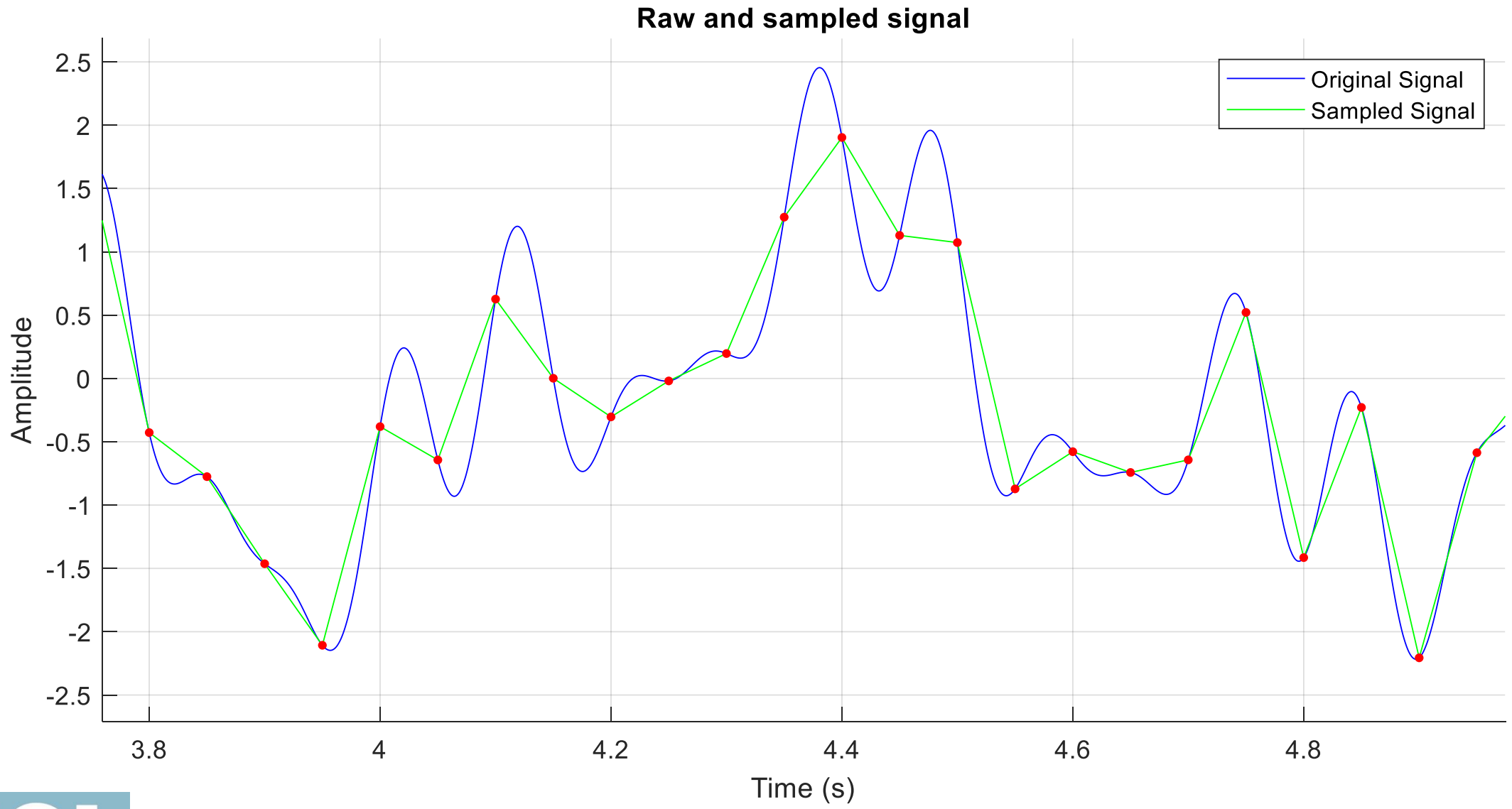




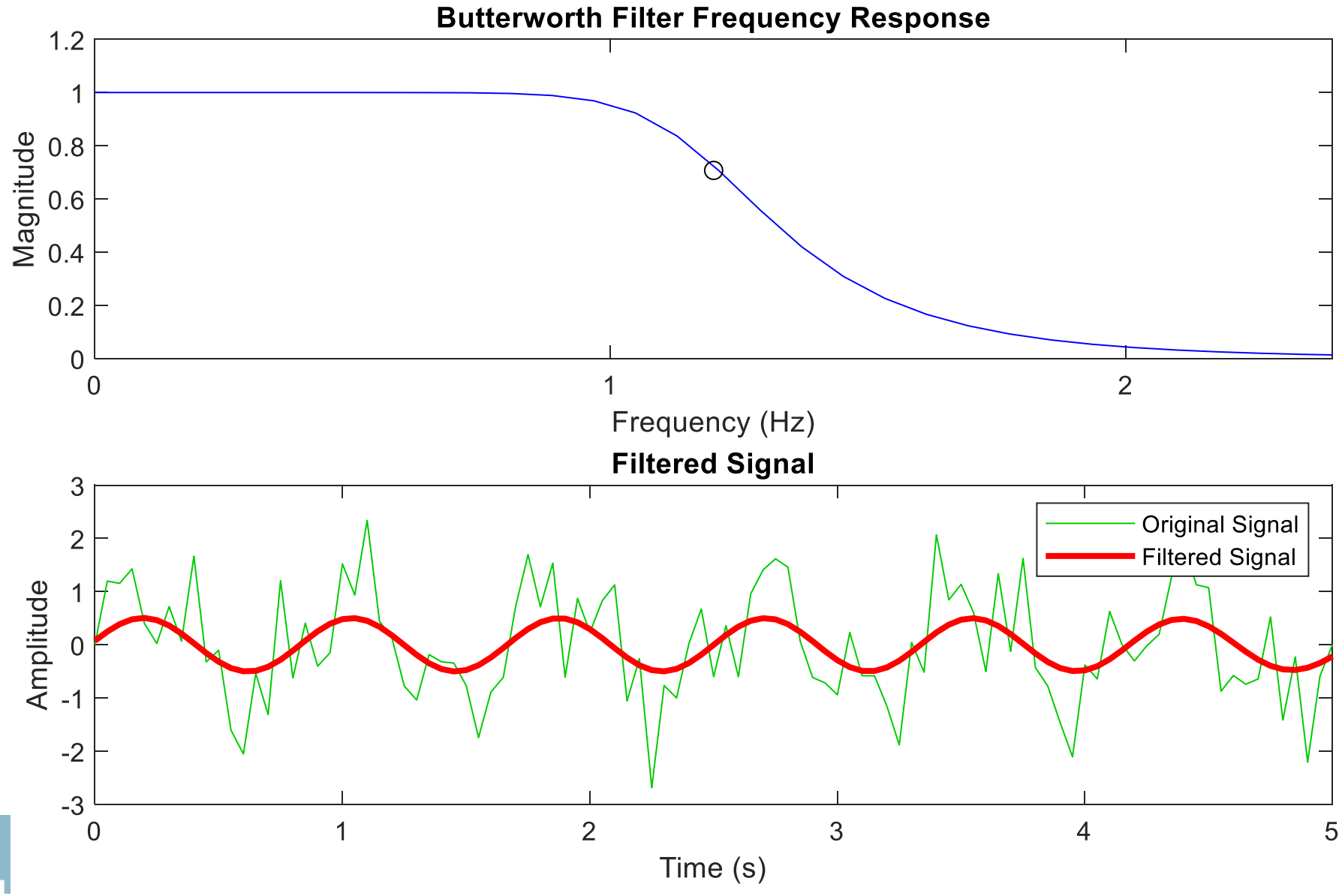
Sampled data



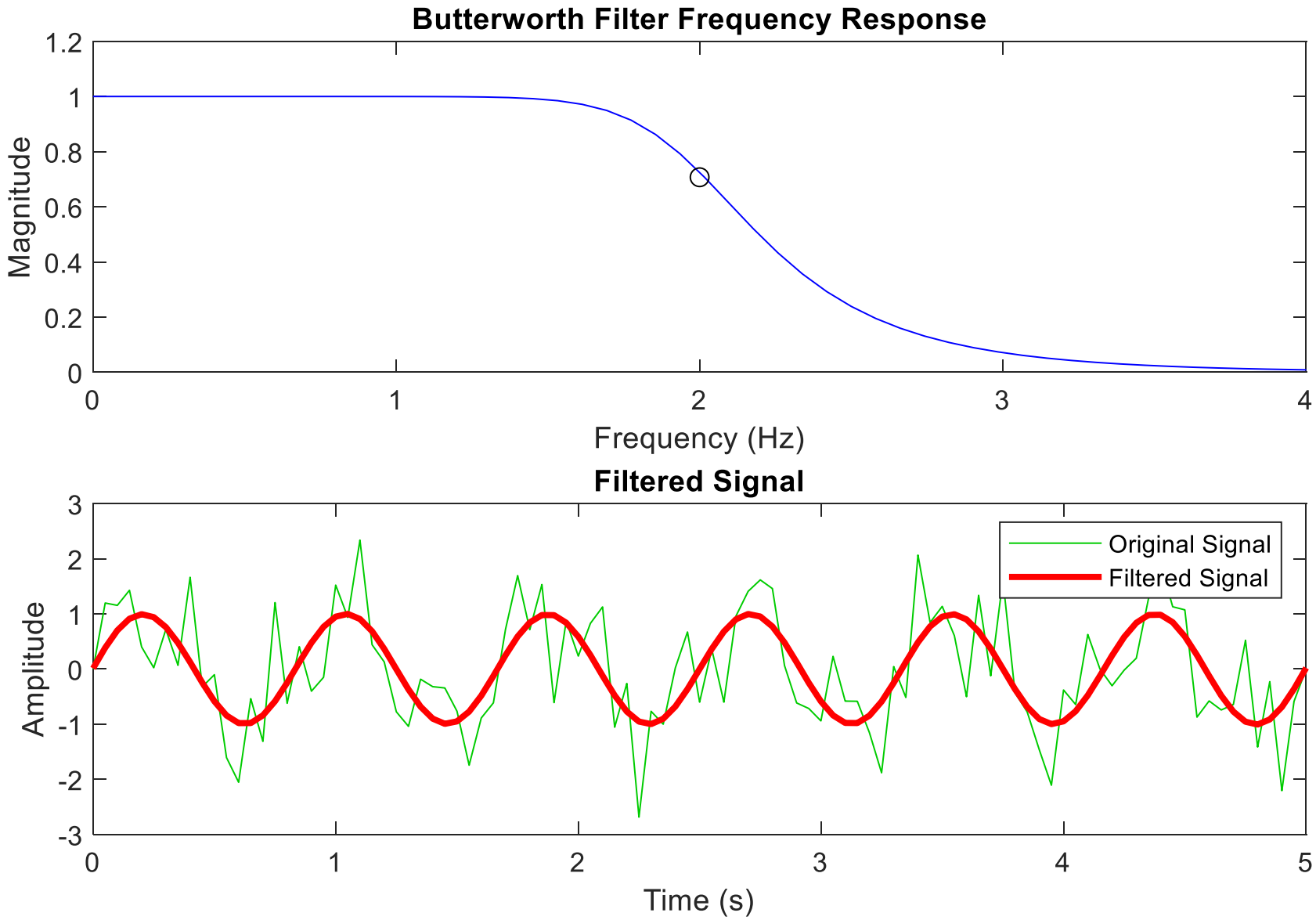
Effects of sampling



Frequency response and filtered data (offline)



Frequency response and filtered data (offline)



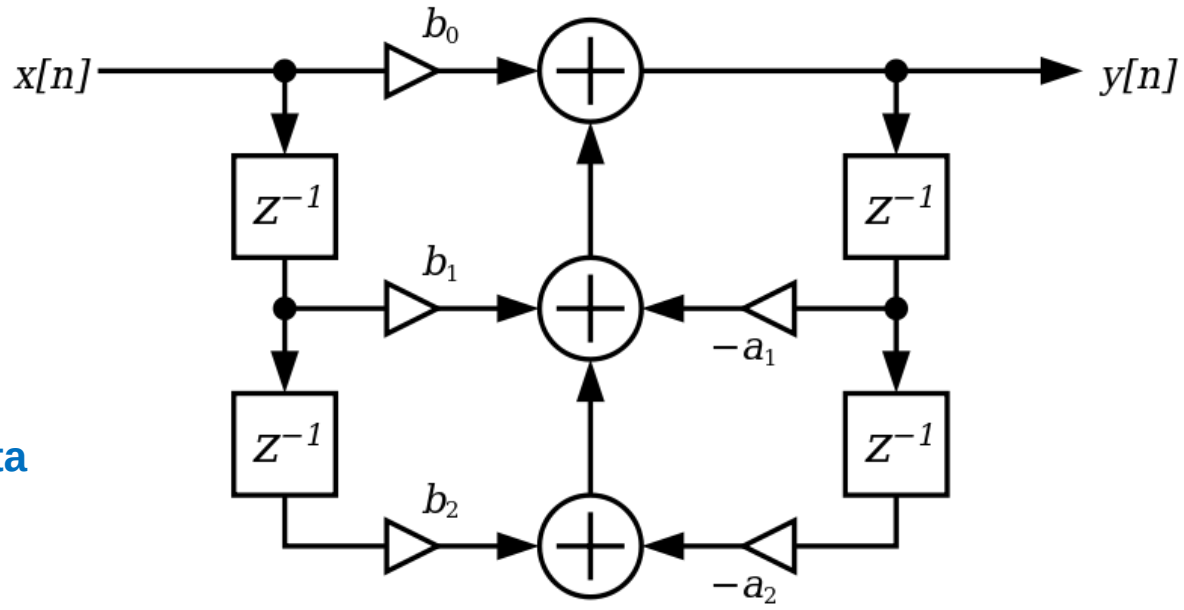
From offline to online signal processing

- Online processing / signal filtering is essential in real-time applications, where:
  - ❑ immediate data results is needed and crucial
  - ❑ it allows for the continuous analysis and filtering of incoming data, enabling real-time use of the processed signals
- Offline data processing involves processing pre-recorded data in batches, allows iterative data process and (re-)analysis, but not suitable for time-critical applications.
- Online processing ensures the ability to handle ongoing signals, making it useful in various applications, eg, control systems, and voice recognition, online audio/video calls.

Offline to online signal processing

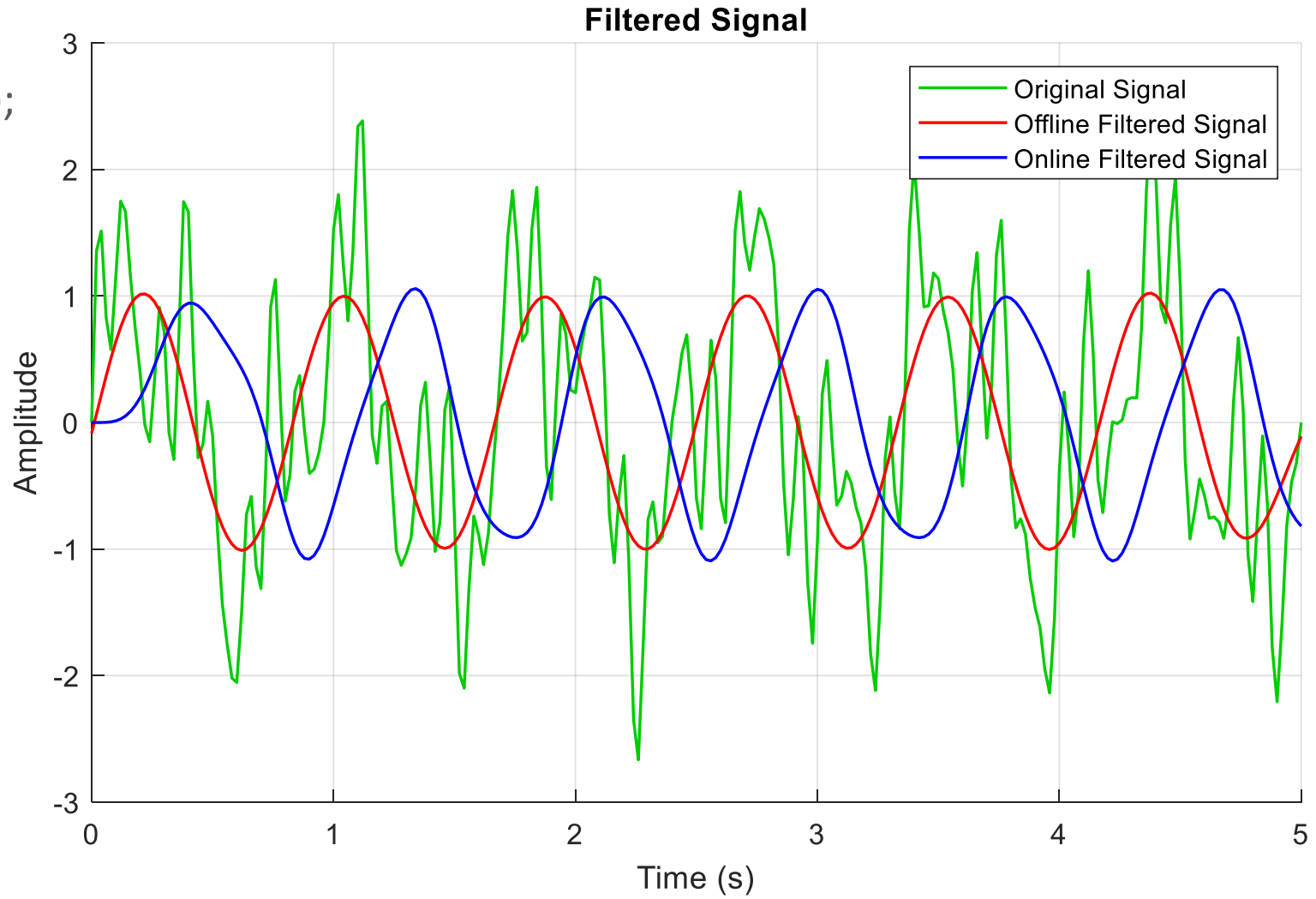
Matlab code example for *data buffer*

```
buffer_size = 10;  % Size of the buffer
data_buffer = [];  % Empty buffer
% Online processing loop
for i = 1:100
    new_data = get_new_data();  % Sample new data point
    % Store new data in buffer
    data_buffer = [data_buffer new_data];
    % Ensure buffer size is maintained
    if length(data_buffer) > buffer_size
        data_buffer = data_buffer(2:end);  % Remove oldest data
    end
    % Process the data in the buffer
    processed_data = process_buffer(data_buffer);
    % Perform further computations or actions using
processed_data
    perform_action(processed_data);
end
```

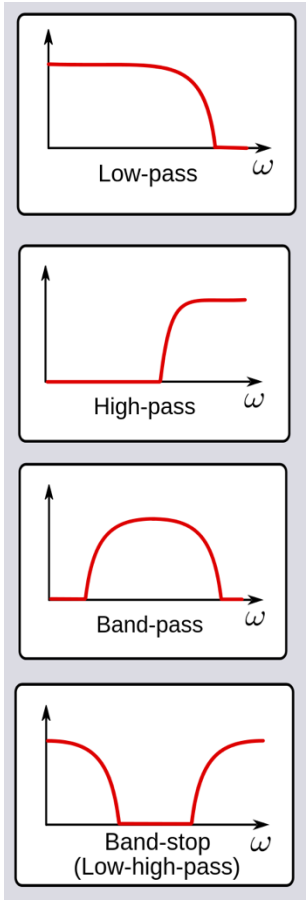


Online Butterworth filter

```
% Initialize the filter's state
z = zeros( max(length(b),length(a))-1, 1);
data_onlinefilter=[];
% Process data in an online fashion
for n = 1:length(data_sampled)
        % Add new sample to input buffer
        x = data_sampled(n);
        % Filter the data
        [y, z] = filter(b, a, x, z);
        % Output the filtered data
        data_onlinefilter(end+1)=y;
end
```

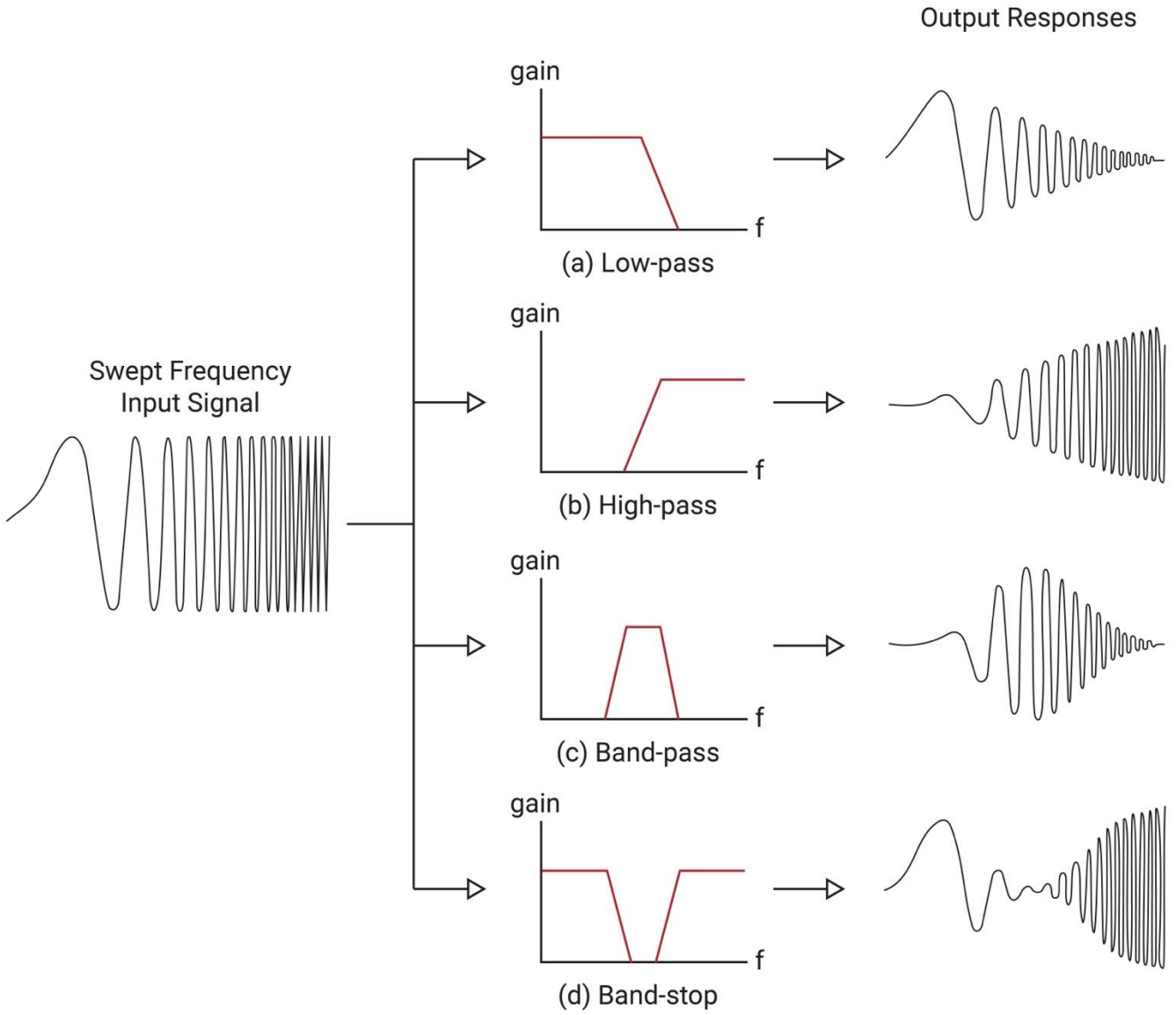


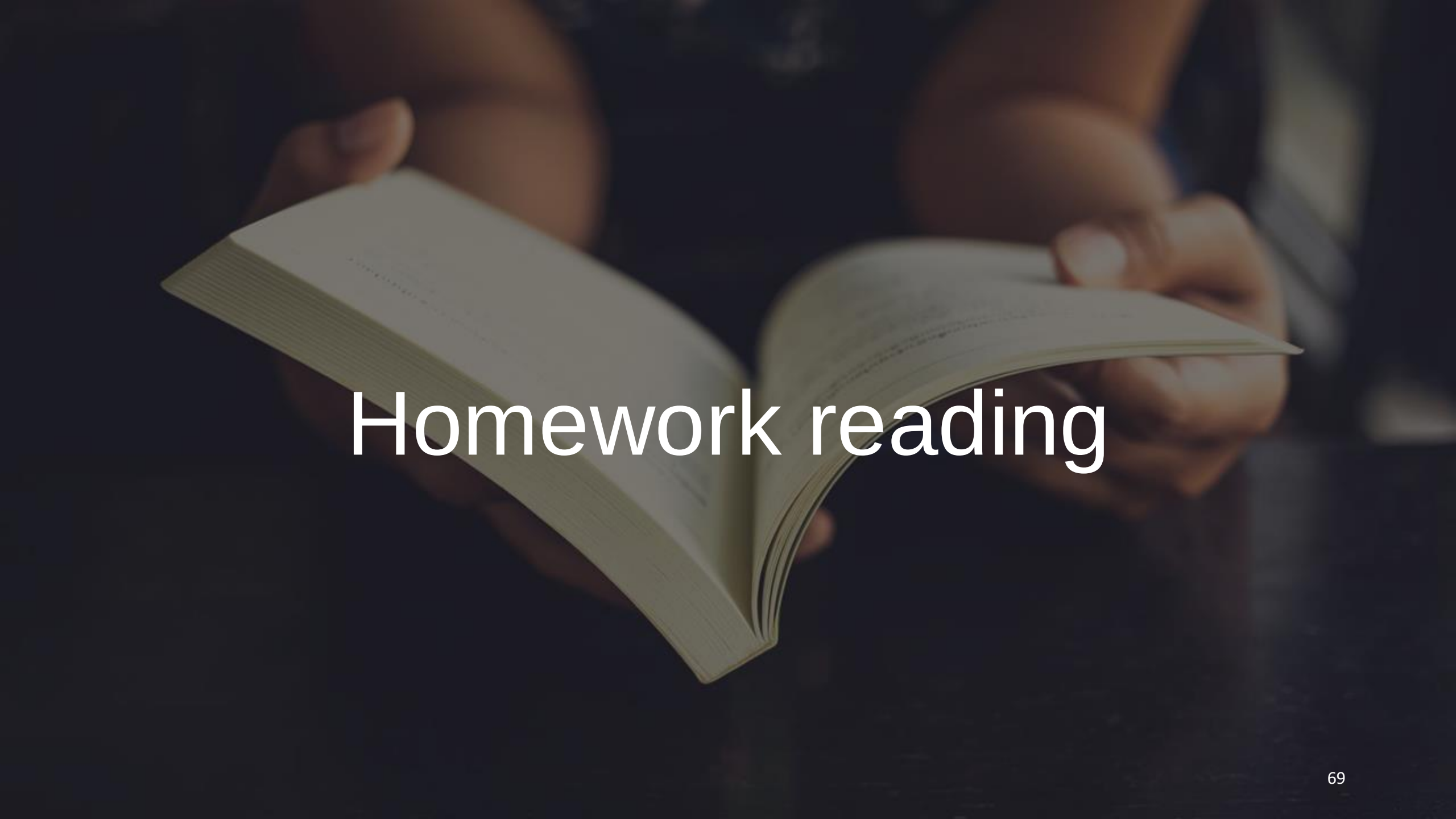
Takeaway messages: commonly used filters



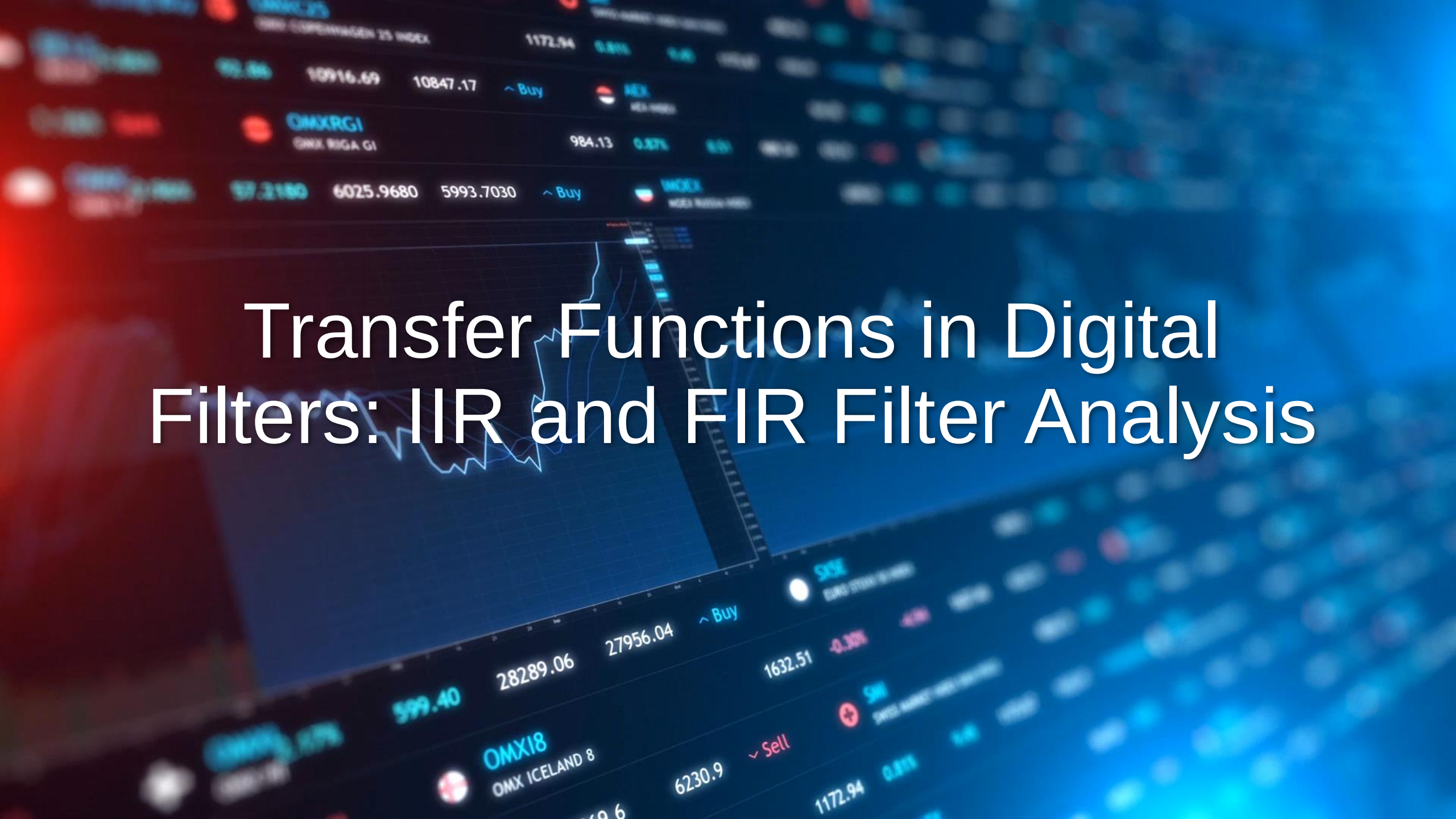
- Low-pass filter: frequencies lower than cut-off frequency are passed
- High-pass filter: frequencies higher than cut-off frequency are passed
- Band-pass filter: only frequencies within a frequency band are passed
- Band-stop filter: only frequencies in a frequency band are attenuated

Takeaway messages: frequency response of digital filters



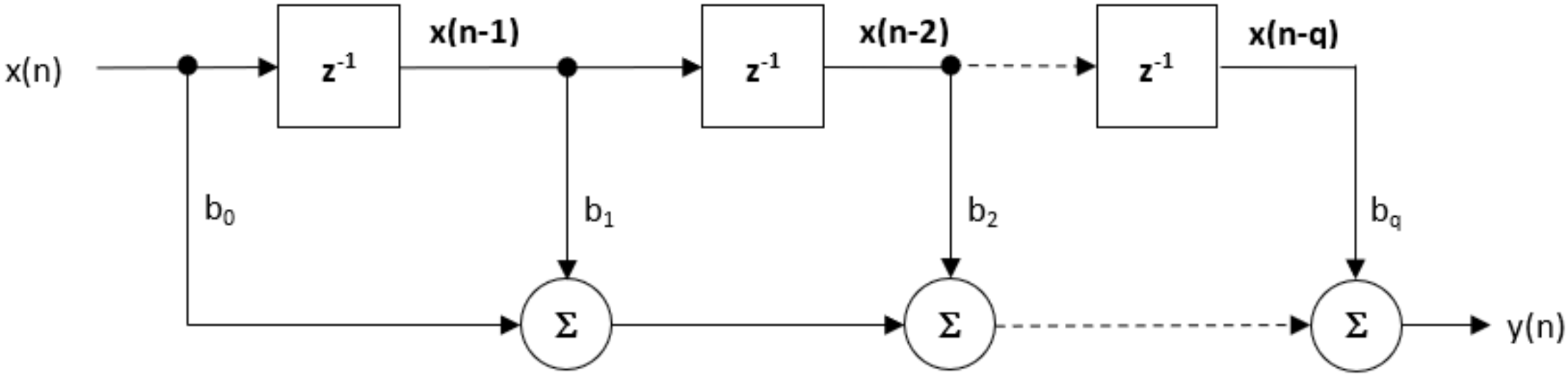


Homework reading


Transfer Functions in Digital Filters: IIR and FIR Filter Analysis

Z-transform

- Concept: Z-transform converts a discretized signal, into a complex frequency-domain (z-domain or z-plane) representation, ie a discrete-time equivalent of the Laplace transform (s-domain). Z-transform is used to analyze and process discrete-time signals, such as stability.
- Within our scope, we only focus on the **basic notion needed practical implementation**.
- Definition & notion: **z**^(-1), sampled data delayed by 1 sampling time



Input-output relation

$$\frac{Y(z)}{X(z)}$$

- y = the output, or filtered value
- x = the input, or incoming raw value
- n = the sample number, iteration number, or time period number

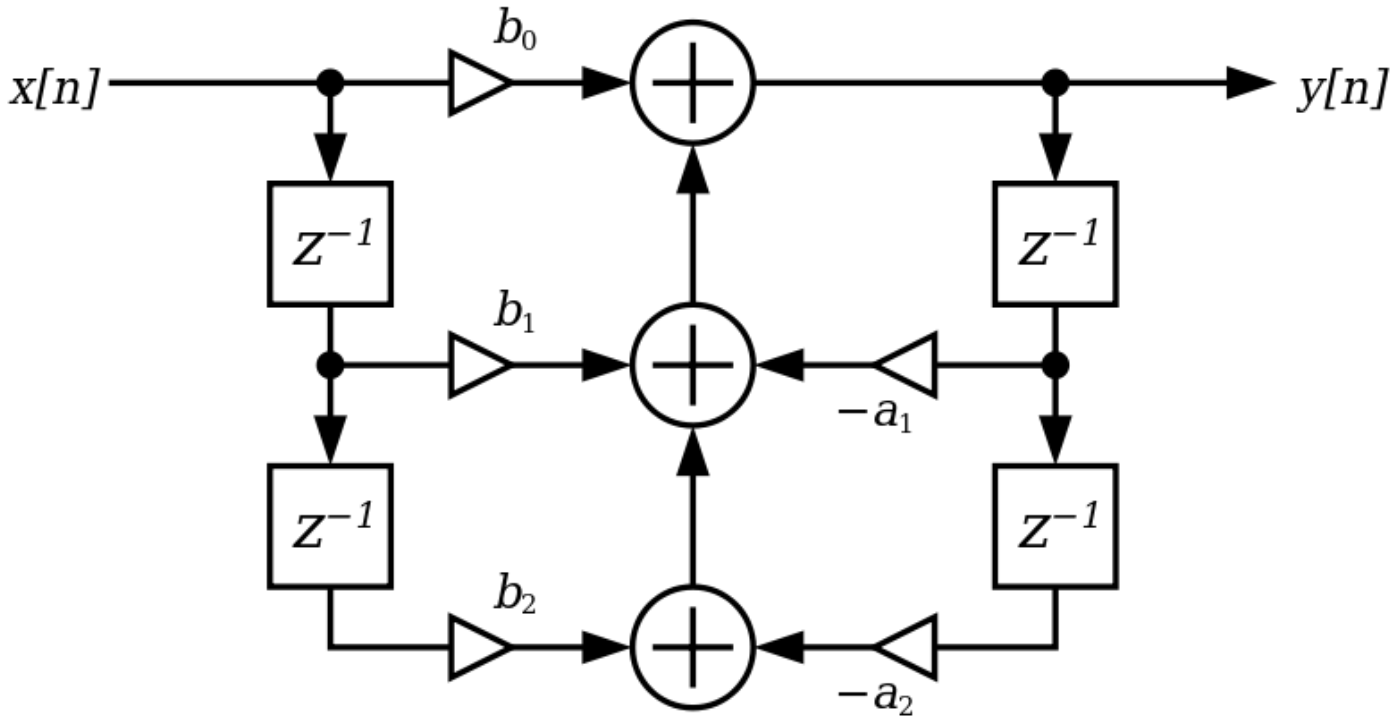
- y[n] = the current filtered (output) value
- y[n-1] = the last filtered (output) value
- y[n-2] = the 2nd-to-last filtered (output) value

- x[n] = the current raw input value
- x[n-1] = the last raw input value
- x[n-2] = the 2nd-to-last raw input value

Direct form

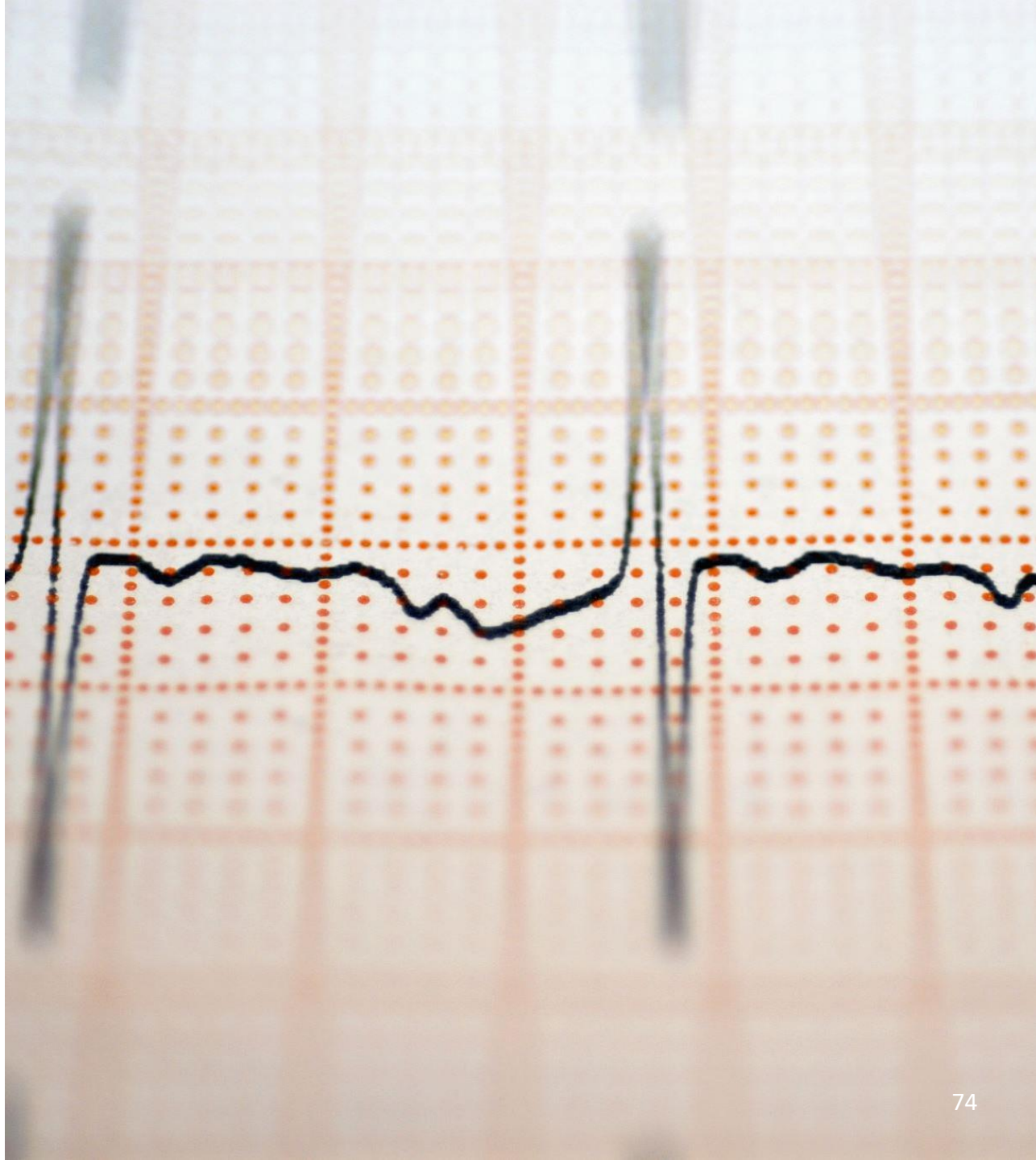
Y(z)

X(z)

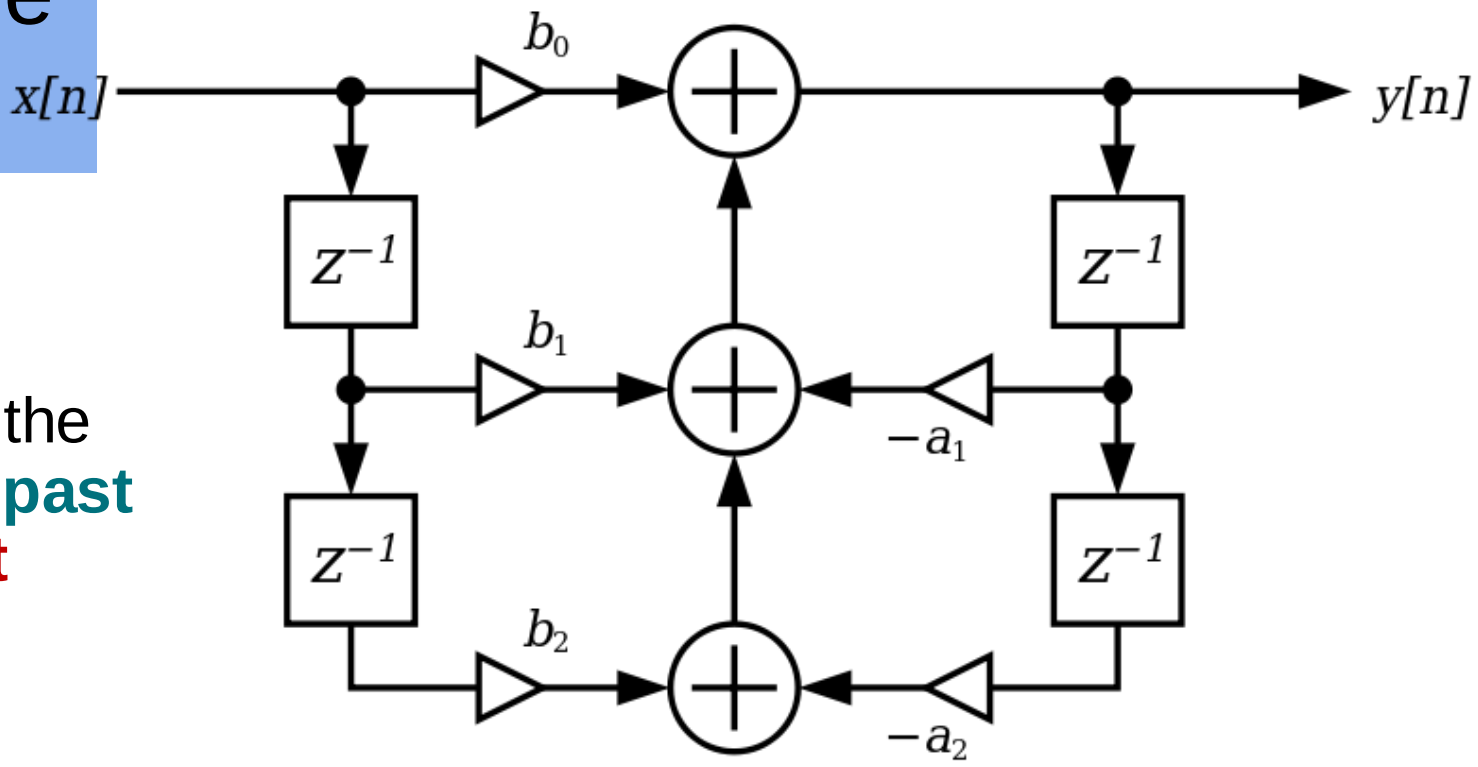


Digital filters: types

- Digital filters are divided into two categories:
- Infinite impulse response (**IIR**): the impulse response exists indefinitely.
- Finite impulse response (**FIR**): the impulse response of the filter falls to zero after a finite period of time.

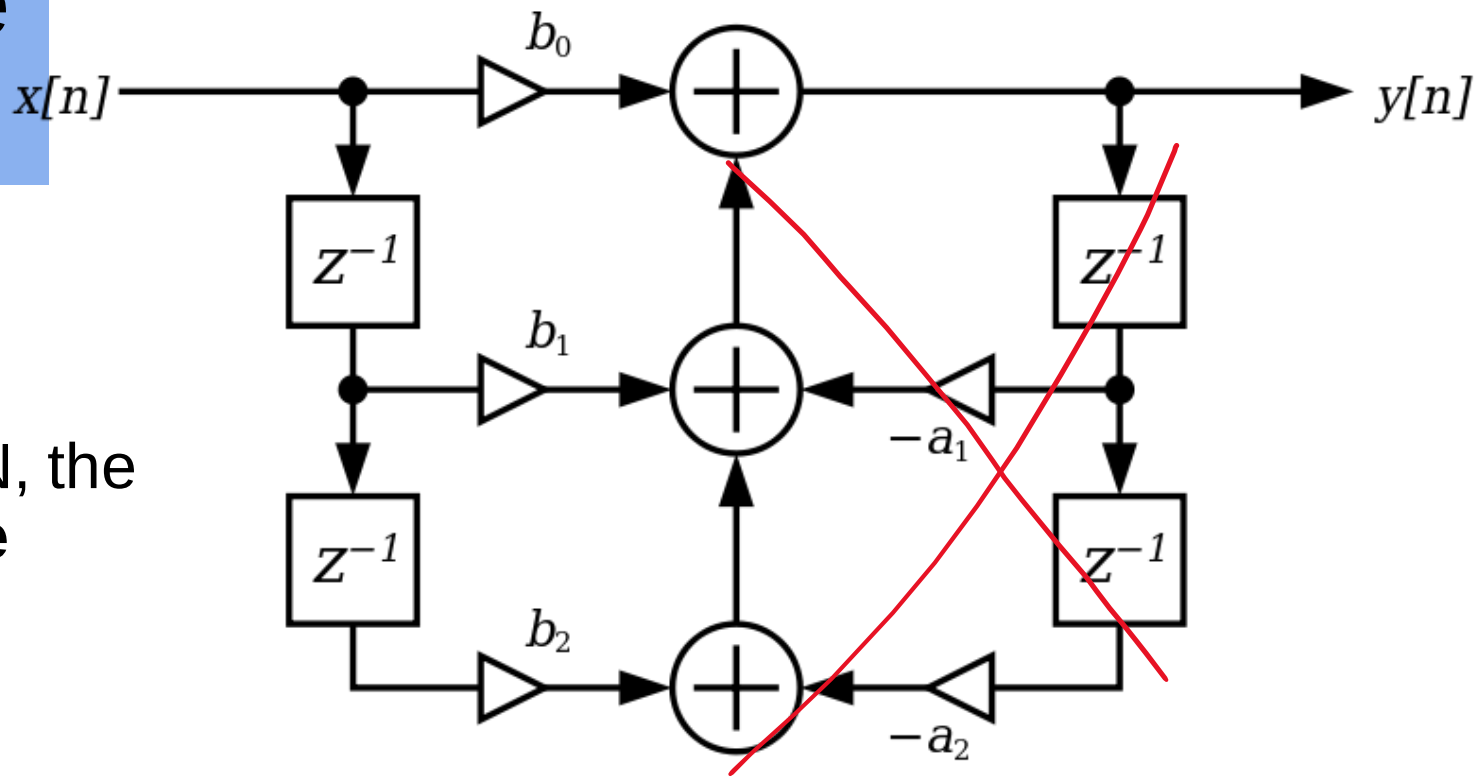


Infinite impulse response - IIR



- Discrete time IIR filter of order N, the output is **a weighted sum of the past and the current inputs and past outputs**:

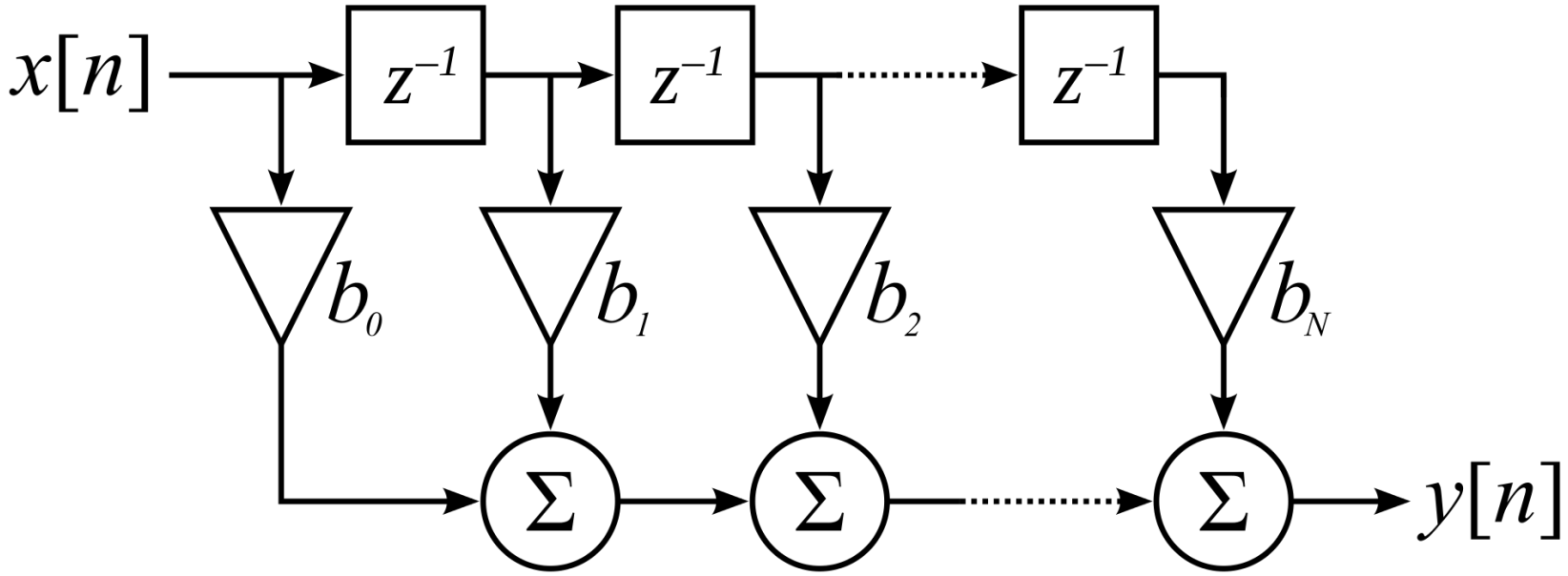
Finite impulse response - FIR



- Discrete time FIR filter of order N, the output is **a weighted sum of the <u>past and the current</u> inputs** :

Finite impulse response - FIR

- Discrete time FIR filter of order N, the output is **a weighted sum of the past and the current inputs** :



Python implementation

- SciPy (https://scipy.org/): SciPy is a free and open-source Python library used for scientific computing and technical computing. scipy.signal is the signal processing toolbox that currently contains some filtering functions and filter design tools.

- We can use scipy.signal.butter to make Matlab-style filter design.

scipy.signal.butter(N, Wn, btype='low', analog=False, output='ba', fs=None)

- Design an Nth-order digital or analog Butterworth filter and return the filter coefficients.

Python implementation

- scipy.signal.butter(N, Wn, btype='low', analog=False, output='ba', fs=None)

Parameters:

- N : int
  The order of the filter
- Wn : array_like
  For a Butterworth filter, this is the point at which the gain drops to 1/sqrt(2) that of the passband (the “-3 dB point”). For digital filters, Wn are in the same units as fs. By default, fs is 2 half-cycles/sample, so these are normalized from 0 to 1, where 1 is the Nyquist frequency.
- btype : {'lowpass', 'highpass', 'bandpass', 'bandstop'} (optional)
- output : {'ba', 'zpk', 'sos'} (optional)
  Type of output: numerator/denominator ('ba'), pole-zero ('zpk'), or second-order sections ('sos'). Default is 'ba' for backwards compatibility, but 'sos' should be used for general-purpose filtering.
- fs : float (optional)
  The sampling frequency of the digital system.

Returns:

- b,a : ndarray, ndarray
  Numerator (*b*) and denominator (*a*) polynomials of the IIR filter. (mostly frequently used)
- z, p, k : ndarray, ndarray, float
  Zeros, poles, and system gain of the IIR filter transfer function.
- sos: ndarray
  Second-order sections representation of the IIR filter.

Docs: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html#scipy.signal.butter

Low pass filter design – python

- Example: We first generate a signal with three frequency component, 1.2 Hz, 9 Hz and 12 Hz. Then we design a digital low pass filter with cutoff frequency at 3 Hz to recover the 1.2 Hz component.

```
data = np.sin(1.2*2*np.pi*t) + 1.5*np.cos(9*2*np.pi*t)
  + 0.5*np.sin(12.0*2*np.pi*t)
order = 6
fs = 30.0       # sample rate, Hz
cutoff = 3      # desired cutoff frequency of the filter, Hz

b, a = butter(order, cutoff/(fs/2), 'low', analog='False')
w, h = freqz(b, a, worN=8000)

plt.subplot(2, 1, 1)
plt.plot(0.5*fs*w/np.pi, np.abs(h), 'b')
plt.plot(cutoff, 0.5*np.sqrt(2), 'ko')
plt.axvline(cutoff, color='k')
filtered_data = lfilter(b, a, data)
```