**What is Sampling Rate?**

In signal processing, the **sampling rate** (also called **sampling frequency**) refers to:

**How many samples of the signal are taken per second** during digital recording or processing.

**📐 Unit:**

* Sampling rate is measured in **Hertz (Hz)**.
* Example: 1000 Hz = **1000 samples per second**

**🎵 Example:**

If you're recording a heart sound (PCG signal) at:

text

CopyEdit

Sampling Rate = 1000 Hz

It means that **every second**, the system captures **1000 amplitude values** of that sound wave.

**📊 Why It's Important:**

| **Sampling Rate** | **Use Case** | **Notes** |
| --- | --- | --- |
| 8,000 Hz | Phone calls | Speech only |
| 44,100 Hz | Audio CDs | Music-quality |
| 1,000 Hz | Heart sounds / PCG signals | Biomedical |
| ≥ 50,000 Hz | High-fidelity instruments | Precision needed |

////////////////////////////////////////////////////////////////////////////////////////////

This function is used to **apply a bandpass filter** to a signal, such as a heart sound (PCG), using the **Butterworth filter** design. Let’s break it down step by step:

**🔧 Function Name:**

python

CopyEdit

def bandpass\_filter(signal, sr, lowcut=20, highcut=200):

* signal: The input signal (e.g., heart sound)
* sr: Sampling rate of the signal (samples/second)
* lowcut: Lower cutoff frequency in Hz (default 20 Hz)
* highcut: Upper cutoff frequency in Hz (default 200 Hz)

**🔍 What is a Bandpass Filter?**

A **bandpass filter** allows only the frequencies **between lowcut and highcut** to pass through and removes the rest (e.g., very low-frequency noise and high-frequency interference).

**💡 Explanation of the Code:**

python

CopyEdit

nyq = 0.5 \* sr

* Calculates the **Nyquist frequency**, which is half of the sampling rate.
* In signal processing, you can only represent frequencies up to the Nyquist limit.

python

CopyEdit

low = lowcut / nyq

high = highcut / nyq

* Normalizes the cutoff frequencies (so they are in the range 0 to 1) for use in filter design.

b, a = butter(4, [low, high], btype='band')

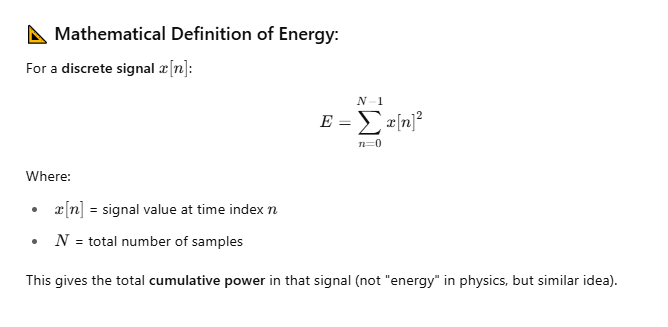
* Designs a **4th-order Butterworth bandpass filter**.
* butter() returns filter coefficients b and a.
* btype='band': means **bandpass filter**.

return filtfilt(b, a, signal)

* Applies the filter **forwards and backwards** using filtfilt(), which avoids phase distortion.
* Returns the **filtered signal**, with only frequencies in the 20–200 Hz range.

**What is Energy in Signal Processing?**

In the context of **audio signals** (like heart sounds or PCG), **energy** represents the **total power or strength** of a signal over time.

**📐 **

**⚙️ How to Calculate Energy in Python:**

Assuming you've already filtered the signal using your bandpass\_filter() function:

python

CopyEdit

from scipy.signal import butter, filtfilt

import numpy as np

def bandpass\_filter(signal, sr, lowcut=20, highcut=200):

nyq = 0.5 \* sr

low = lowcut / nyq

high = highcut / nyq

b, a = butter(4, [low, high], btype='band')

return filtfilt(b, a, signal)

# Example usage

filtered\_signal = bandpass\_filter(signal, sr=1000)

# 🔋 Energy of the entire filtered signal

energy = np.sum(filtered\_signal \*\* 2)

print(f"Total signal energy: {energy:.2f}")

**🪜 For Windowed Energy (e.g., per 200ms window):**

This is common in heart sound analysis for **detecting murmurs** or **S1/S2 segmentation**.

python

CopyEdit

sr = 1000 # Sampling rate

win\_len = int(0.2 \* sr) # 200 ms window

windowed\_energy = [

np.sum(filtered\_signal[i:i + win\_len] \*\* 2)

for i in range(0, len(filtered\_signal) - win\_len, win\_len)

]

This gives a list of energy values per window.