Exploring Aliasing and the Sampling Theorem

Margherita Tonon

April 2025

1 Introduction

In signal processing, computers analyze and process discrete time signals. However, most signals of interest are continuous time signals, as these are the signals most commonly found in nature – e.g. sound waves. We therefore use sampling, allowing us to convert a continuous-time signal into a discrete signal that a computer is able to process. Nonetheless, one of the factors determining the quality of sampled signals is the sampling rate. Given that the Shannon-Nyquist theorem, a condition relating to the sampling rate, is met, a signal can be perfectly recovered from its sampled signal. However, if this condition is not met, a signal can no longer be perfectly recovered due to a phenomenon named aliasing [1]. Here, we will explore the process of recovering a sampled signal, looking at different sampling rates that meet and do not meet the conditions for perfect signal recovery.

2 Background

In signal processing, signals are commonly categorized as analog signals and digital signals. An analog signal refers to a signal that varies continuously over time. The complexity of analog signal processing, their susceptibility to noise and signal degradation over time, as well as their limited reproductibility and scalability makes them inconvenient to work with in practice. Therefore, digital signals are used – signals that vary discretely over time and can take only a finite number of distinct values.

Sampling refers to the process of converting an analog signal into a digital signal. If we let $x(t)$ be a continuous time signal, the sampled signal $x[n]$ is defined as

$$x[n] = x(nT_s)$$

where n represents discrete time sampling points and T_s represents the sampling period, such that the sampling frequency $f_s = \frac{1}{T_s}$.

Sampling can also be represented as

$$x_s(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT_s)$$

where $x_s(t)$ is the sampled points, and δ is the Dirac Delta distribution, taking value 1 if $t = nT_s$, and 0 otherwise. Therefore, it is clear that when $t = nT_s$, $x_s(t) = x(t)$; otherwise, $x_s(t) = 0$.

In practice, sampling allows us to discretize a continuous input, facilitating the handling of signals. After sampling is done, it is natural that the signal must be reconstructed in order to recover the original (time continuous) signal. However, recovery is not always perfect – the Shannon-Nyquist condition must be met.

The Shannon-Nyquist theorem states that a signal can be perfectly reconstructed if the sampling frequency is greater than two times the maximum frequency B of the continuous signal:

$$f_s > 2B$$

For example, if we consider a sine wave with frequency 2 Hertz (Hz), $\sin(2\pi \cdot 2 \cdot t)$, once sampled it can be perfectly reconstructed if the sampling frequency is greater than 4 Hz.

In the time domain, a sampled signal looks like discrete spikes (Figure 1).

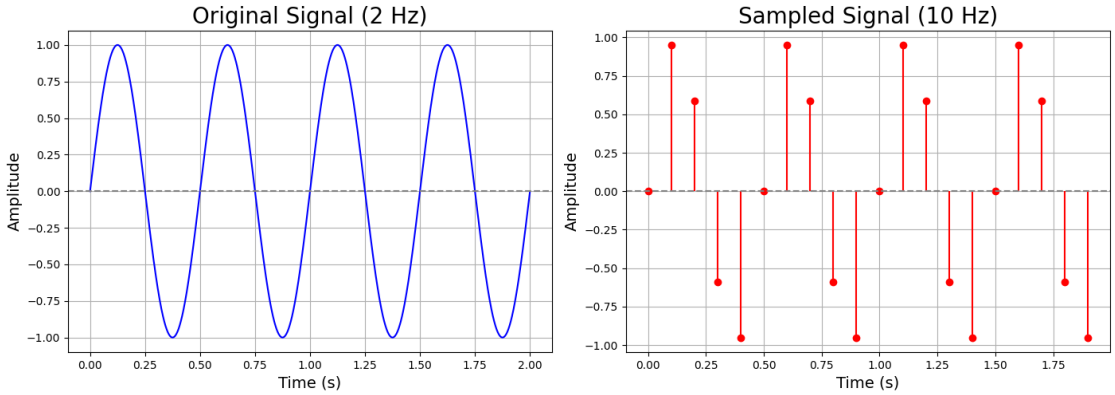

Figure 1: Representation of sampling of $\sin(4\pi t)$ in the time domain, sampled at a frequency of 10 Hz.

The Fourier Transform, defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi f t}\, dt$$

allows us to transition from the time domain into the frequency domain. In order to visualize what sampling looks like in the frequency domain, we make use of the Inverse Convolution Theorem.

The Inverse Convolution Theorem states that if we have two functions $x(t)$ and $h(t)$ whose Fourier Transforms $\mathcal{F}(x(t))$ and $\mathcal{F}(h(t))$ are absolutely integrable in the frequency domain, then

$$\mathcal{F}^{-1}\left(X(f) * H(f)\right) = x(t) \cdot h(t)$$

where $*$ denotes the convolution operator. Consequently, if $x(t)$ and $h(t)$ are absolutely integrable and the Fourier Transform of their product exists,

$$X(f) * H(f) = \mathcal{F}\left(x(t) \cdot h(t)\right)$$

Therefore, because we can express sampling as $x_s(t) = x(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT_s)$ in the time domain, the Inverse Convolution Theorem tells us that

$$\mathcal{F}\left(x(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT_s)\right) = \mathcal{F}(x(t)) * \mathcal{F}\left(\sum_{-\infty}^{\infty} \delta(t - nT_s)\right)$$

To visualize sampling in the frequency domain we must therefore convolve the Fourier Transform of the input signal with the Fourier Transform of $\sum_{n=-\infty}^{\infty} \delta(t - nT_s)$, known as the Dirac comb. Using properties of the Fourier Transform, Dirac Delta distribution, Fubini's Theorem of exchanging the order of integration and summation, and the Poisson summation formula, we obtain that

$$\mathcal{F}\left(\sum_{-\infty}^{\infty} \delta(t - nT_s)\right) = \tfrac{1}{T_s} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{T_s}\right)$$

We therefore have

$$\mathcal{F}(x(t)) * \mathcal{F}\left(\sum_{n=-\infty}^{\infty} \delta(t - nT_s)\right) = X(f) * \tfrac{1}{T_s} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{T_s}\right)$$

Convolution of a function with a delta function shifts the function by the offset/position of the delta function, and therefore

$$\mathcal{F}(x(t)) * \mathcal{F}\left(\sum_{n=-\infty}^{\infty} \delta(t - nT_s)\right) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X\left(f - \frac{k}{T_s}\right)$$

We can therefore observe how the representation of a sampled signal in the frequency domain is simply the Fourier Transform of the signal, duplicated and shifted to multiples of the sampling frequency.

When recovering the original signal, we assume that the Shannon-Nyquist condition is met. Because $f_s > 2B$, the original signal has to be contained within the range of frequencies $\frac{-f_s}{2}$ and $\frac{f_s}{2}$, as $\frac{-f_s}{2} < -B$ and $\frac{f_s}{2} > B$. Because the frequency domain representation of a signal has the frequency of the signal as well as the negative frequency of the signal, we only need to consider the positive frequencies: 0 to $\frac{f_s}{2}$ Hz. We therefore apply a low pass filter with cutoff frequency $f_l = \frac{f_s}{2} = \frac{1}{2T}$. In the frequency domain, we would multiply this low pass filter with the sampled signal, and then perform an inverse Fourier Transform on the signal.

Nonetheless, the frequency domain representation of the sampled signal and the low pass filter highlight the need for the Shannon-Nyquist theorem. When filtering a sampled signal, we effectively only see and select the frequencies between $\frac{-f_s}{2}$ to $\frac{f_s}{2}$. If $f_s = 2B$, this range becomes $[-B, B]$, and therefore all of the frequency components of the original signal are included. However, when $f_s < 2B$, the filter will select frequencies on a range $[-a, a]$, where $a < B$. Clearly, B is not included within this range and we end up missing part of the original spectrum. We recall that sampling in the frequency domain replicates the original spectrum $X(f)$ at every integer multiple of f_s. If f_s is too small, some copied frequencies will shift into in the interval $\left[\frac{-f_s}{2}, \frac{f_s}{2}\right]$ – this is known as folding. When filtering, these folded frequencies will be wrongly interpreted as the original signal's frequency, and therefore the original signal will not be reconstructed correctly. The process of different frequency components becoming indistinguishable in the sampled signal due to overlapping spectra is known as aliasing. On the other hand, if $f_s > 2B$, the range $\left[\frac{-f_s}{2}, \frac{f_s}{2}\right]$ fully contains $[-B, B]$. Because f_s is sufficiently large, the shifted copies will not land inside $\left[\frac{-f_s}{2}, \frac{f_s}{2}\right]$, and the original signal can be reconstructed perfectly.

3 Implementation

To implement the process of sampling in Python, I begin by defining a function `generate_signal` which generates a continuous time signal. If the parameter `signal_type` is set to `"single"`, a sine wave of desired frequency is created. If this parameteris set to `"multiple"`, a sine wave of desired frequency is added to a sine wave with half of the desired frequency. Here, I simply create a time array with $5000t_e$ elements, where t_e is the duration in seconds of the signal, and then apply the sine function with the given frequency to the $5000t_e$ time values. Users can set the `plot` parameter to `True` if they wish a plot of the original continuous-time signal to be displayed.

Next, I define the `sample_signal` function, which samples the continuous signal at a specified sampling frequency. In this code block, I create a new time array `t_sampled` for the sampled signal using the sampling period, representing the new set of discrete time points. Next, I use `np.interp` to perform the sampling of the original signal `x` at the time points in the `t_sampled` array. Users can set `plot_one` to `True` if they wish to visualize the original signal together with the sampled points, `plot_two` to `True` if they wish to just visualize the sampled points, and `plot_three` to `True` to visualize both plots side by side.

Thirdly, I create the `sampled_fourier_transform` function, which takes as inputs the sampled amplitudes, the sampling frequency, and the number of times to repeat the spectrum (k), and performs the Fast Fourier Transform of the sampled amplitudes using the `scipy` module [2]. The Fourier Transform integrates a signal over all time (from $-\infty$ to $+\infty$) and requires a signal to be continuous. However, in many practical situations, such as our current scenario, we work with finite and discrete signals. The Discrete Fourier Transform allows us to numerically compute the Fourier Transform for a finite time interval, and the Fast Fourier Transform is a way of

computing the Discrete Fourier Transform in a more computationally efficient way. Therefore, in the `sampled_fourier_transform` function, `yf` is a complex array representing the sampled signal `x_sampled` in the frequency domain, and `xf` is the frequency values, in Hertz, that each element in `yf` corresponds to. If `plot` parameter is set to `True`, a plot of the frequencies of the sampled signal in the frequency domain is shown.

Finally, I define the `reconstruction` function, which reconstructs the original signal from the sampled signal. This function takes as inputs the array of the amplitudes of the sampled signal `x_sampled` and the time array of the sampled signal `t_sampled`. If users wish to plot the reconstructed signal together with the original signal, they must set the `plot` parameter to `True` and provide the original signal `x_continuous` as well as the continuous time array `t_s`.

In the code, I reconstruct the signal in the time domain using the sinc function. Previously, we mentioned that we must apply a low pass filter with cutoff frequency $f_l = \frac{f_s}{2}$ to recover the signal. The convolution theorem states that if $x(t)$ and $h(t)$ are two absolutely integrable functions in $\mathcal{L}^1$, and if their Fourier transforms $X(f) = \mathcal{F}(x(t))$ and $H(f) = \mathcal{F}(h(t))$ exist and are well defined,

$$\mathcal{F}\{x(t) * h(t)\} = X(f) \cdot H(f)$$

We therefore can convolve the time-domain representation of the lowpass filter with the sampled signal, and we will obtain the filtered signal.

The low pass filter has an impulse response defined as

$$h(t) = \frac{1}{T} \operatorname{sinc}\left(\frac{t}{T}\right)$$

where $\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$.

Recalling that our sampled signal is $x(nT)$, their convolution is

$$\int_{-\infty}^{\infty} \left[\sum_{n=-\infty}^{\infty} x(nT)\delta(\tau - nT)\right] \cdot h(t-\tau)d\tau$$

After some simplifications, we obtain that the recovered signal can be found using

$$\sum_{n=-\infty}^{\infty} x(nT) \cdot \operatorname{sinc}\left(\frac{t-nT}{T}\right)$$

which is the approach implemented in the `reconstruction` function.

However, because it is not possible to compute an infinite sum numerically, I must loop over a finite number of iterations. Mathematically, this looks like

$$x(t) \approx \sum_n x[n] \operatorname{sinc}\left(F_s t - n\right)$$

where $F_s = 1/T$ and $x[n]$ is the value of the signal at the n-th sampling point.

4 Discussion

We begin by observing a continuous time signal: $x(t) = \sin(2\pi \cdot 5t) + \sin(\pi \cdot 5t)$ (Figure 2).

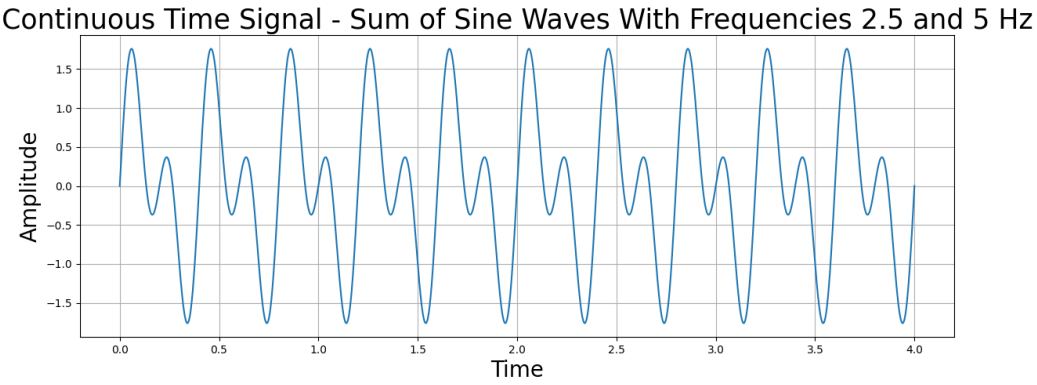

Figure 2: Continuous time signal $x(t) = \sin(2\pi \cdot 5t) + \sin(\pi \cdot 5t)$.

The highest frequency component of this signal is $B = 5$. According to the Shannon-Nyquist theorem, if we sample the signal at a frequency greater than $2 \cdot 5 = 10$ Hz, we can perfectly reconstruct the signal. Sampling the wave with sampling frequencies of 12 Hz (above the Shannon-Nyquist threshold) and 7 Hz (below the Shannon-Nyquist threshold) is seen in Figure 3.

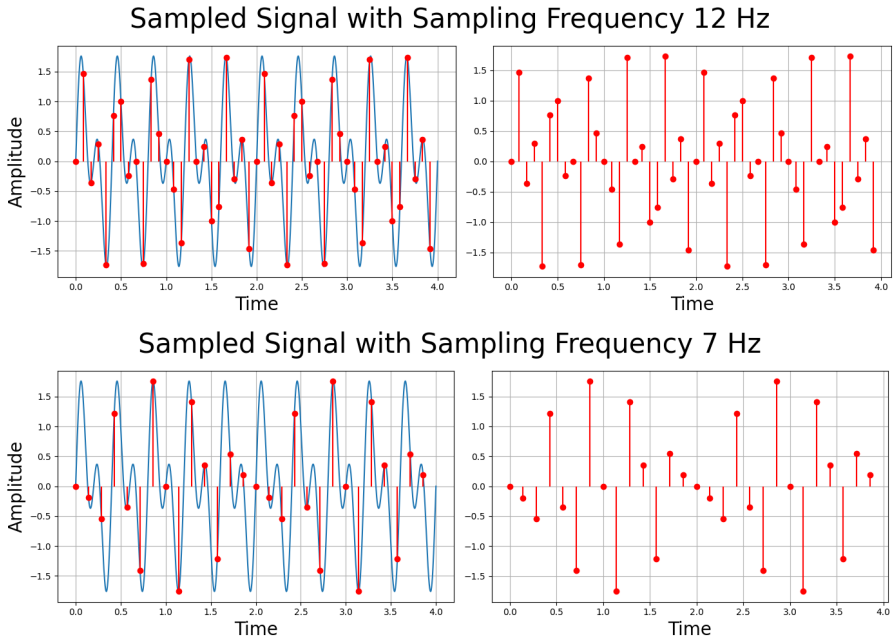

Figure 3: Visualization of the sampling a continuous time signal with a sampling frequency 12 Hz (top) and 7 Hz (bottom). The original signal is in blue on the left plots, and the red spikes are the sampled amplitudes.

As expected, the time domain representation of the signal sampled at 12 Hz has more spikes than the signal sampled at 7 Hz, as a higher sampling frequency implies a signal is sampled more times within a given time interval.

In the frequency domain, the original continuous time signal looks like four spikes at 5, -5, 2.5, and -2.5 Hz (Figure 4).

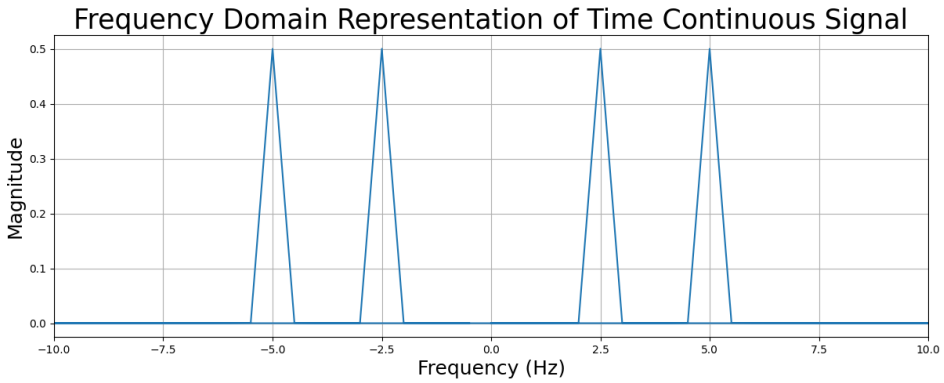

Figure 4: Frequency domain representation of the continuous time signal.

Figure 5 shows the frequency domain representation of the signal sampled at 12 Hz and 7 Hz.

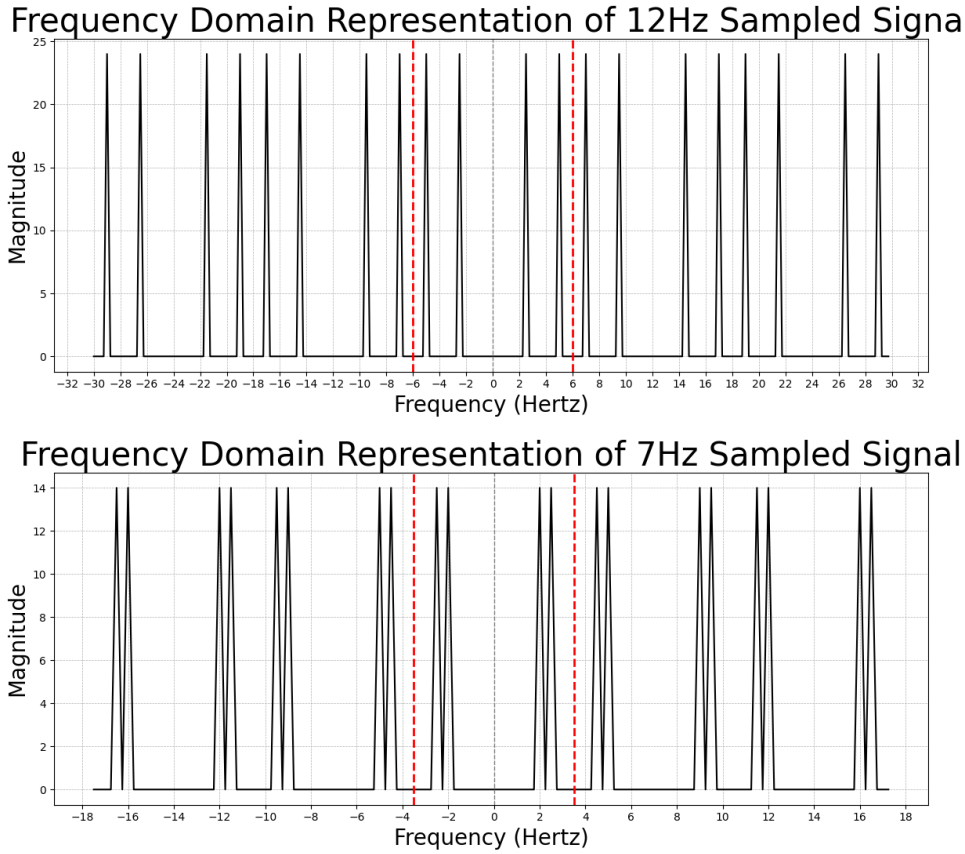

Figure 5: Frequency domain representation of sampled signal at 12 Hz (top) and 7 Hz (bottom). The red dotted line represents the low pass filter cutoff frequencies $\frac{f_s}{2}$.

When the sampling frequency is above the Shannon-Nyquist threshold, we can see that both of the original frequency components of the original signal (2.5 and 5 Hz) are contained within the filtering bound, which in this case is at $\frac{12}{2} = 6$ Hz.

However, when the sampling frequency is below the Shannon-Nyquist threshold,

5 Conclusion

References

[1] Fiveable. "Signal Digitization – Biomedical Engineering II." Edited by Becky Bahr, Fiveable, 2024, https://fiveable.me/key-terms/biomedical-engineering-ii/signal-digitization. Accessed 22 Apr. 2025.

[2] MacLeod, Cameron. "Fourier Transforms with Scipy.Fft: Python Signal Processing." Real Python, Real Python, 1 Sept. 2022, realpython.com/python-scipy-fft/. Accessed 13 Apr. 2025.

6 Appendix

1. Access the GitHub repository with all code here.