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).

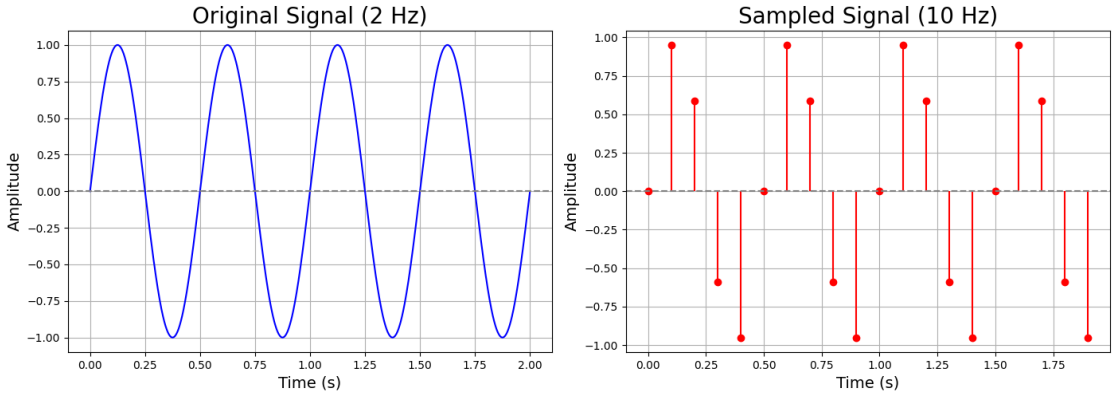

Figure 1: Representation of sampling of $\sin(4\pi t)$ in the time domain, sampled at a frequency of 10 Hz.

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, and 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

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

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.

To recover the original signal from the sampled signal, we apply an ideal band pass filter,

$$H(f) = \begin{cases} 1 \text{ if } f_l < f < f_h \\ 0 \text{ otherwise} \end{cases}$$

which essentially only passes the frequencies between f_l and f_h Hertz and ignores all other frequencies outside of this range. We then perform an inverse Fourier Transform on the signal and we will have recovered the original signal.

Nonetheless, the frequency domain representation of the sampled signal highlights the need for the Shannon-Nyquist theorem. When our sampling frequency is sufficiently large, the process of copying and shifting the frequencies occurs at larger intervals, meaning the copied frequencies are more spread out. As a result, it is possible to apply an ideal filter to recover the original frequencies. However, if the sampling frequency decreases, the interval between frequencies is not as large and they may overlap. It is therefore no longer possible to apply an ideal filter to recover the original frequencies, as the original signal and the duplicated frequencies overlap and interfere with each other. This phenomenon is known as aliasing: the act of different frequency components becoming indistinguishable in the sampled signal due to overlapping spectra. As long as we set $f_s > 2B$, however, the signal can be perfectly recovered.

3 Implementation

To implement the process of sampling in Python, I begin by defining a function `generate_signal` which generates a sine wave (a continuous-time signal) of 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, and `plot_two` to `True` if they wish to just visualize the sampled points.

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. 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. The sinc function, defined as $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$, is utilized as it is the time domain representation of the ideal band pass filter $H(f)$. The sinc function is the result of taking the inverse Fourier Transform of the ideal band pass filter:

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{2\pi i f t} df$$

$H(f)$ is even and real; therefore, $e^{2\pi i f t} = \cos(2\pi f t) + i \sin(2\pi f t) = \cos(2\pi f t)$.

$$h(t) = \int_{-\infty}^{\infty} H(f) \cos(2\pi f t) df$$

Evaluating this integral,

$$h(t) = \frac{1}{\pi t} \left[\sin(2\pi f_h t) - \sin(2\pi f_l t)\right]$$

This is equivalent to the difference of two sinc-like functions:

$$h(t) = 2f_h \cdot \text{sinc}\left(2f_h t\right) - 2f_l \cdot \text{sinc}\left(2f_l t\right)$$

In a lowpass filter, we pass all frequencies below a specified frequency f_l. This is essentially like using a bandpass filter with boundaries $f_l = 0$ and $f_h = f_l$:

$$h(t) = 2f_l \cdot \text{sinc}\left(2f_l t\right) - 2 \cdot 0 \cdot \text{sinc}\left(2 \cdot 0 \cdot t\right) = 2f_l \cdot \text{sinc}\left(2f_l t\right)$$

As previously explained, sampling a signal in the frequency domain corresponds to taking copies of the Fourier Transform of the signal, and shifting them to multiples of the sampling frequency. The baseband signal is going to lie around 0, as this is the first "copy" of the signal

The signal $x(t)$ can therefore be recovered in the following way:

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

which is the approach implemented in the `reconstruction` function. Because it is not possible to perform an infinite sum, I must loop over a finite number of iterations. Mathematically, this looks like

$$x(t) = \sum_{n} x[n] \,\text{sinc}\left(F_s t - n\right)$$

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

4 Discussion

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.