# SATFD - lab 02 report

Dawid Karpiński, 18.03.2024

## 1 Comparison of signals in time and frequency domain.

In figure 1 one can see the generated plots of different signals in time and frequency domain. The signals include an infinite signal, a rectangular window, a finite signal (result of multiplying the infinite signal with the rectangular window), a non-rectangular window, and a smoothed finite signal (result of multiplying the infinite signal with the non-rectangular window).

The multiplication of two signals in the time domain results in the convolution of their frequency domain representations.

Listing 1: **Code snippet for generating the signals.**

```
N = 1000
A = 5
f = 10
fs = 1000

dt = 1 / fs
time = dt * np.arange(N)

infi = A * np.sin(2 * np.pi * f * time)
window = time[time < 5 / f]
rect = np.pad(np.full(window.size, 1), ((time.size - window.size) // 2))
nonrect = scipy.signal.windows.gaussian(N, std=1 / f * N)

spectrum = np.abs(spectrum) ** 2
freqs = np.fft.fftfreq(signal.size, 1 / fs)
```

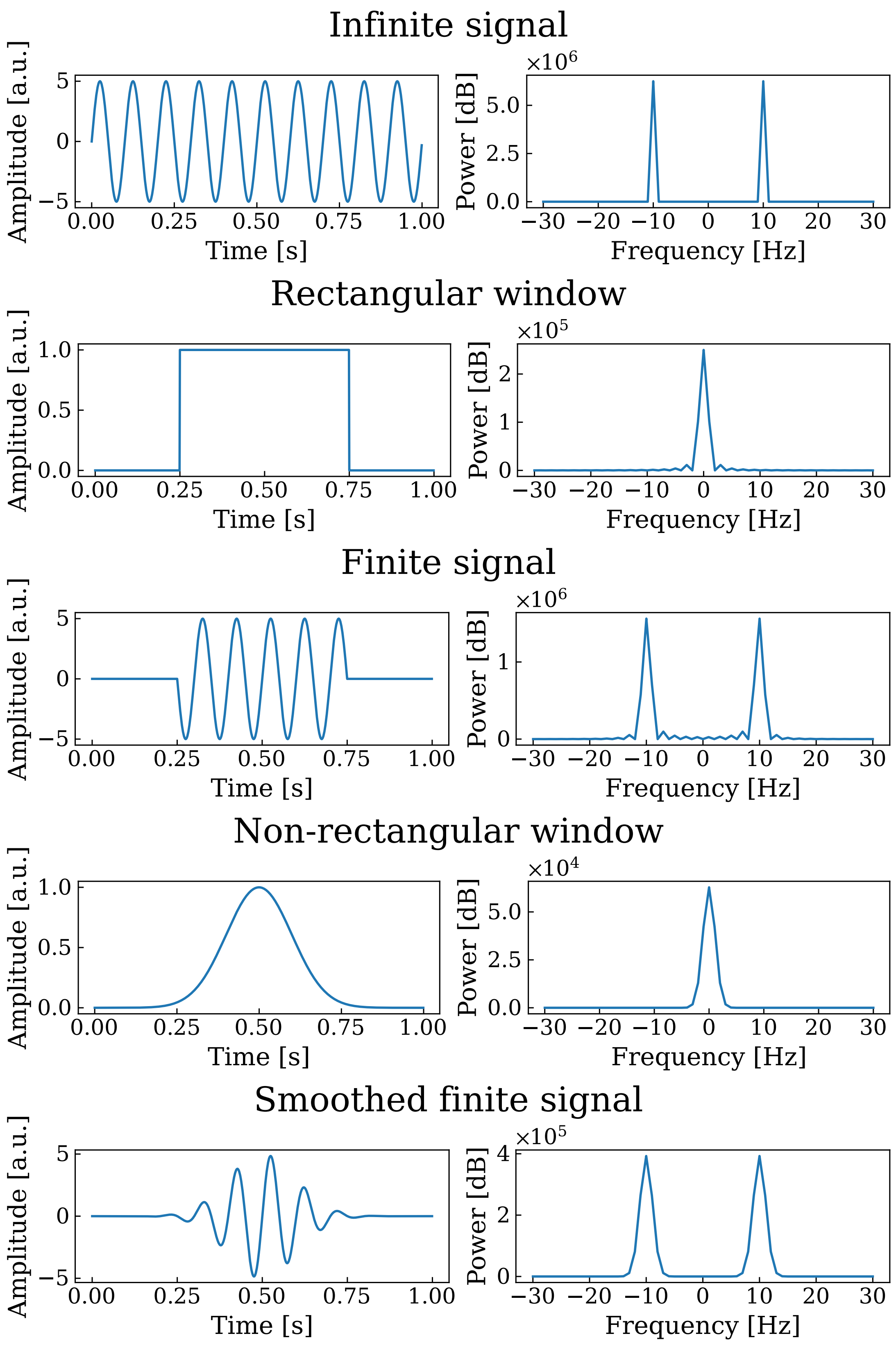

Figure 1: **Signals comparison**.

## 2 Sampling theorem and aliasing.

According to the sampling theorem, to perfectly reconstruct a signal, the sampling frequency $f_s$ must be at least twice the maximum frequency of the signal. This minimum sampling frequency is called the Nyquist frequency.

The analysis has been performed on a Windows startup sound (figure 2), with the original sample rate of $f_s = 22.05$ [kHz]. The following plots in figure 3 show a gradually decimated signal, with the removal of every other sample, where the impact of sampling frequency on the frequency domain representation of the signal can be observed.

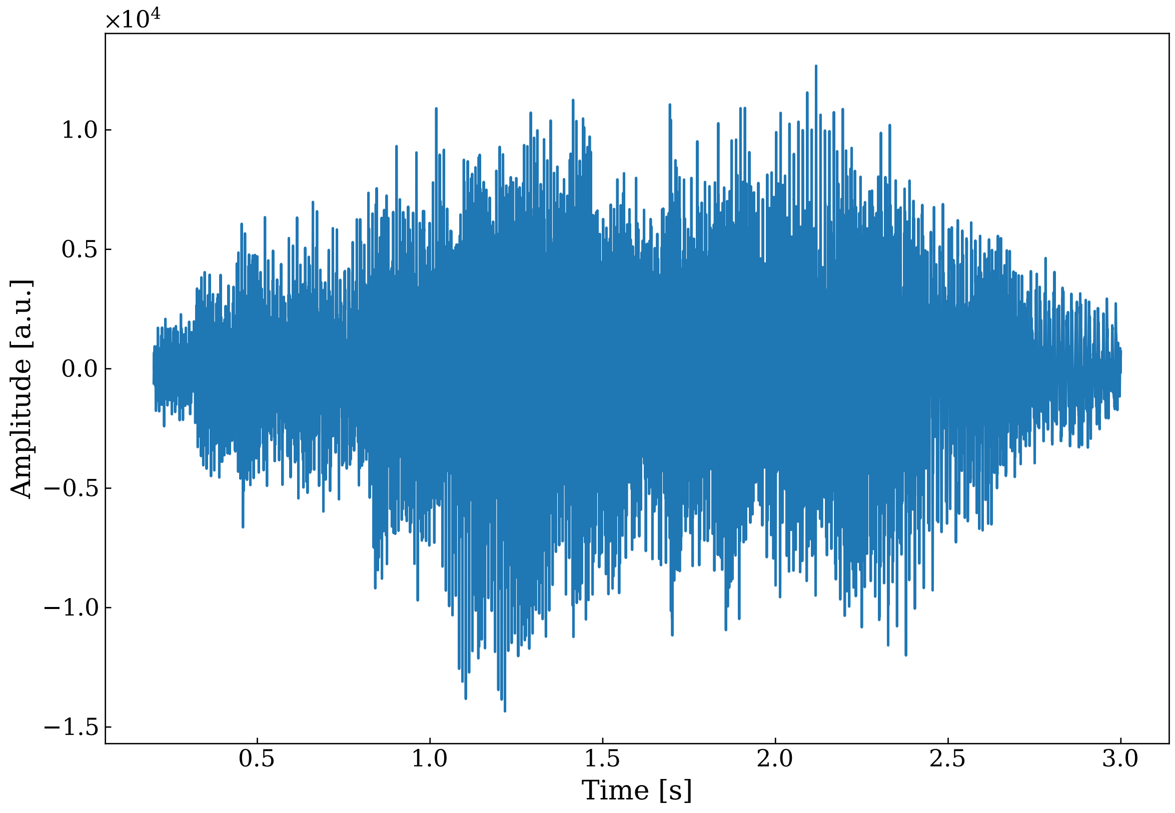

Figure 2: **Windows startup sound**.

At each decimation, the range of signal's frequencies moved towards larger values. Additionally, some of the frequencies are lost, probably due to occurring aliasing.

Listing 2: **Code snippet for gradual decimation of the signal**.

```
fss = fs // np.array([1, 1.5, 2, 5, 10])

for curr_fs in fss:
    samples = round(wave.size * curr_fs / fs)
    rwave, rtime = signal.resample(wave, samples, t=time)
```

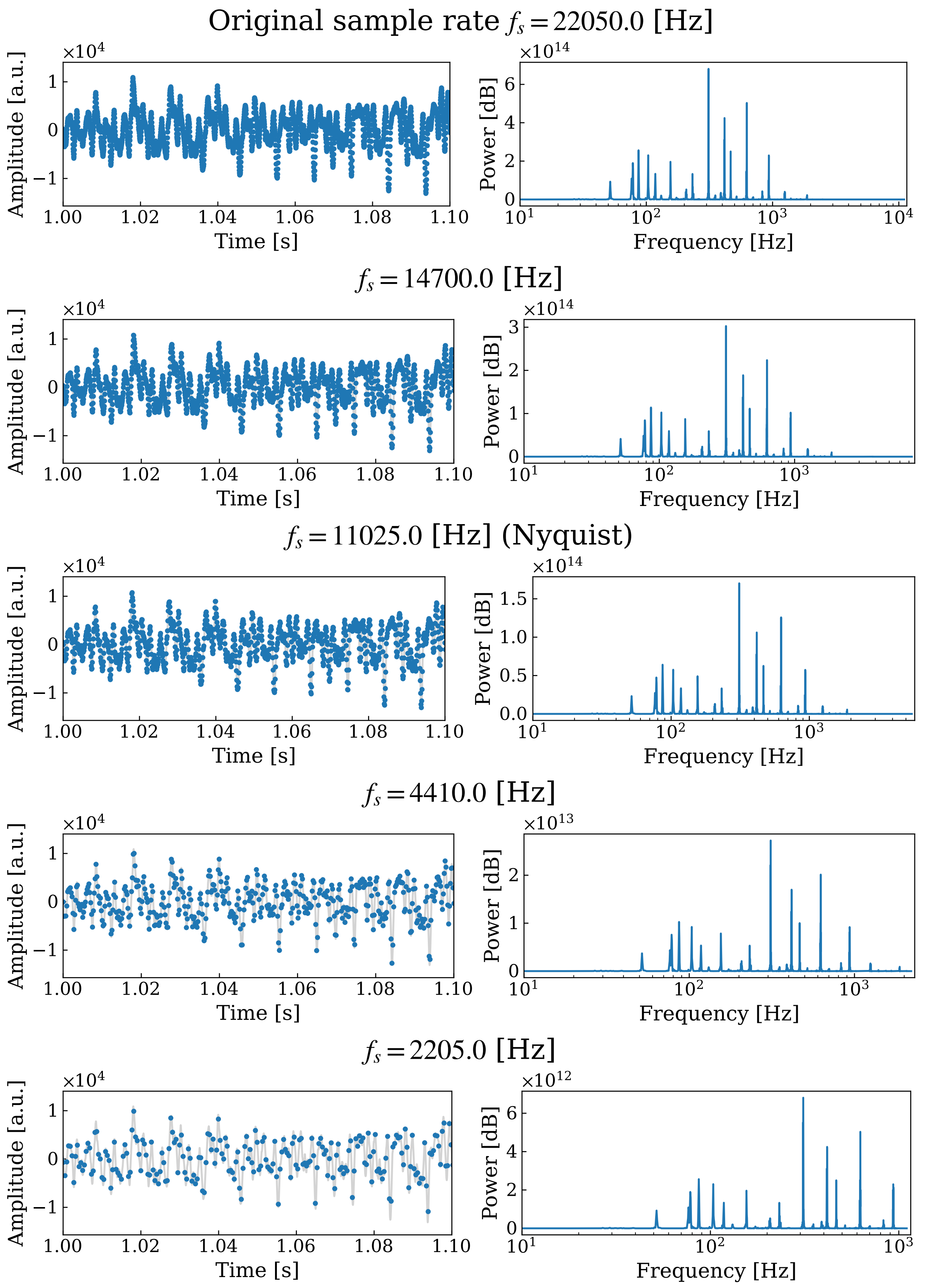

Figure 3: **Signals comparison. For better readability, only 0.1 [s] of the signal in the time domain is shown.**

## 3 Zero-padding the signal.

Zero padding is a technique used to increase the resolution of the frequency domain representation of a signal. The signal is therefore extended by adding zeros to its end, effectively increasing its length.

Listing 3: **Code snippet for adding the padding to infinite signal.**

```
wave = np.pad(wave, (0, wave.size * 3))
time = np.arange(wave.size) / fs
```

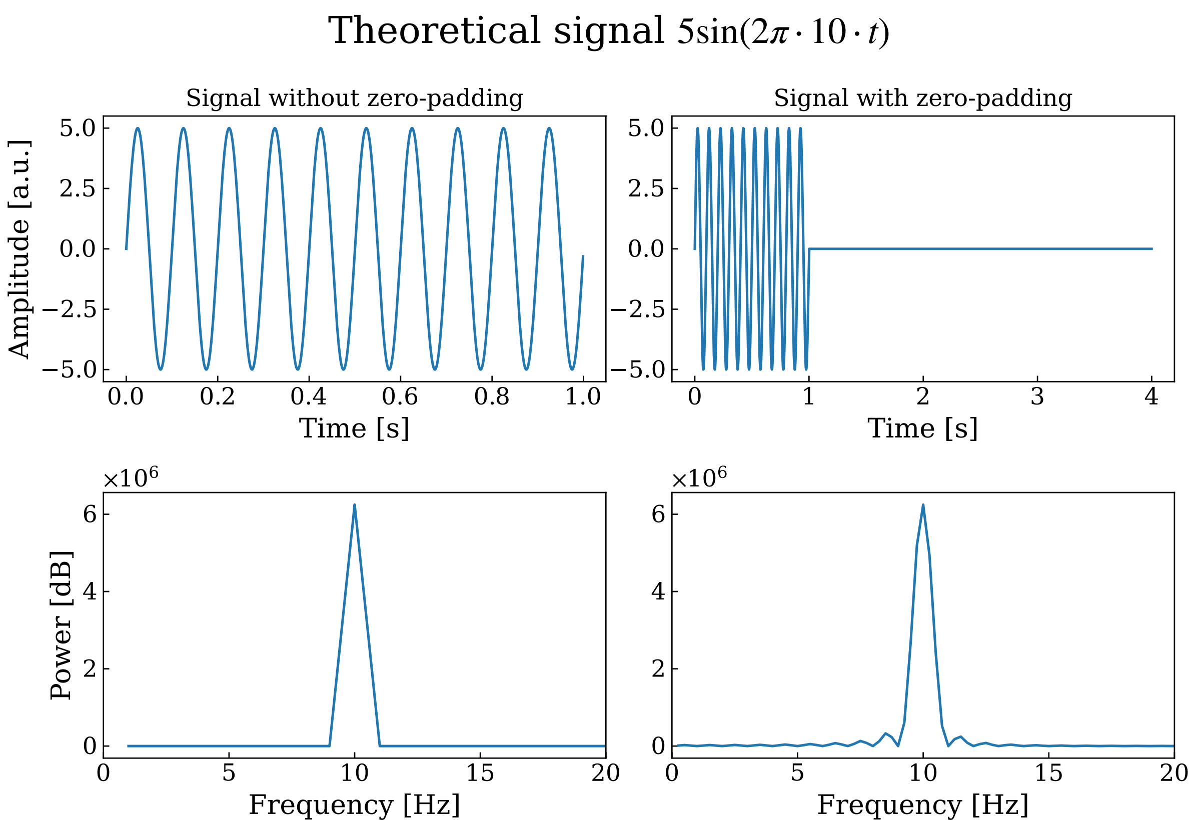

Figure 4: **Zero padding.**

## 4 Signal upsampling.

In upsampling, the signal is interpolated by inserting zeros between the samples, which results in a higher sampling rate.

Listing 4: **Code snippet for upsampling the signal**.

```
upsampled = np.zeros(wave.size * 2)
upsampled[::2] = wave

new_fs = 2 * fs
time = np.arange(wave.size * 2) / new_fs
```

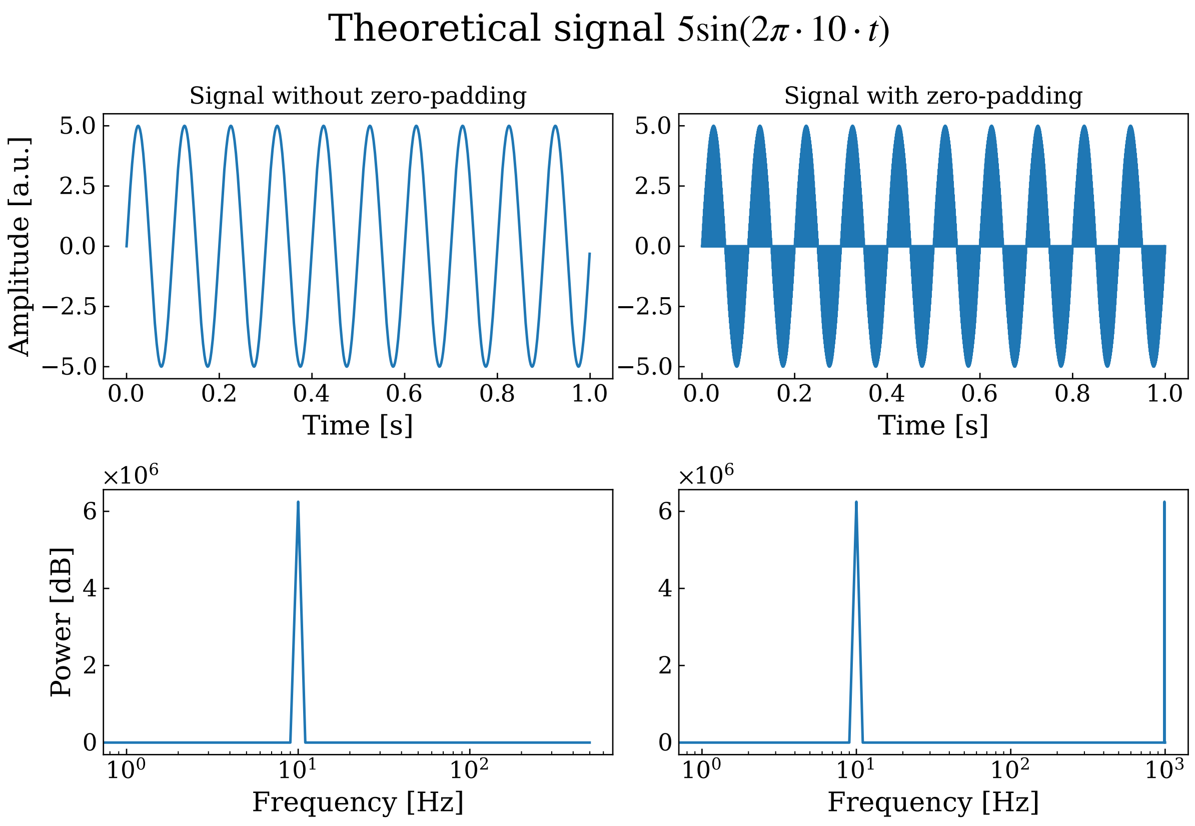

Figure 5: **Signal upsampling**.

Moreover, after performing the upsampling, a new frequency appeared in the power spectrum.

## 5 Conclusion.

In conclusion, the task in lab 02 has explored various aspects of signal processing, including the comparison of windows applied to signals in both time and frequency domains, the application of the sampling theorem and its implications on aliasing, zero-padding techniques to enhance frequency resolution, and signal upsampling by zero interpolation.

The entire code for generating the data and plots can be found at:

`https://github.com/davkk/signal-analysis/tree/main/sat/lab02`