# REPORT

Zajęcia: Analog and digital electronic circuits
Teacher: prof. dr hab. Vasyl Martsenyuk

**Lab**
**Date: 09.1.2025**
**Topic: 7. Sampling and Reconstruction of Signals: Analysis of Aliasing Effects and Proper Signal Reconstruction. 8. Coding and Decoding Digital Signals**
**Variant DSP 4**

Tomasz Pietrzyk
Informatyka II stopień,
niestacjonarne,
1 semestr,
Gr.A

## 1. Problem statement:

The laboratory focuses on two key areas: **sampling and reconstruction** of signals, and **coding and decoding** of digital signals. In the first part, students analyze the effects of aliasing and perform proper signal reconstruction to explore the fundamental principles of the Nyquist-Shannon sampling theorem. In the second part, students implement coding and decoding techniques, including delta encoding, quantization, and transform-based compression, to optimize digital signal representation and analyze trade-offs between compression ratio and distortion. These tasks aim to enhance understanding of digital signal processing concepts through practical Python implementations.

## 2. Input data:

Sampling and Reconstruction (Variant 4)

- Signal Type: Triangular wave
- Frequency of Signal (f): 8 Hz
- Sampling Frequency (fs): 12 Hz
- Duration: 1 second
- Time Vector: t = linspace(0, 1, 1000) (continuous)
- Sampling Period: T = 1/fs
- Wave Parameters: Use the scipy.signal.sawtooth function with a width parameter of 0.5 to generate a triangular wave.

Coding and Decoding (Variant 4)

- Problem to Solve: Quantize the signal and calculate the quantization error.
- Signal Data: [1.2, 2.3, 3.1, 4.5, 5.7]
- Quantization Levels: 3
- Required Outputs:
    - Quantized signal.
    - Quantization error for each signal value.
- Quantization Formula: Quantized Value = round((Signal - min) / Step Size) * Step Size + min Where Step Size = (max - min) / Levels.

## 3.Commands used (or GUI):

jupyter Untitled Last Checkpoint: 42 seconds ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

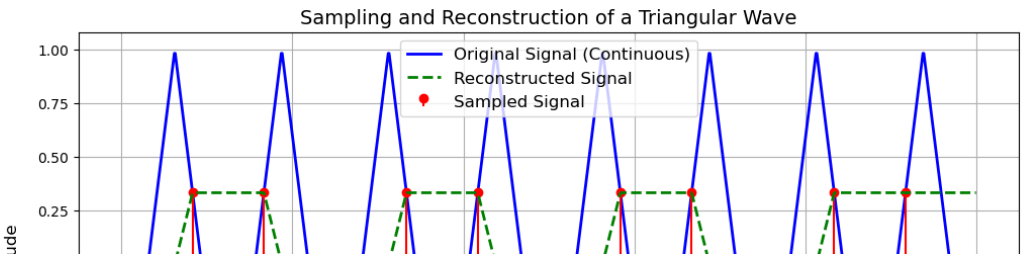
[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import sawtooth

# Define parameters for the original signal
signal_frequency = 8  # Frequency of the triangular wave (Hz)
sampling_frequency = 12  # Sampling frequency (Hz)
duration = 1  # Duration of the signal (seconds)

# Generate the time vector for the continuous signal
time_continuous = np.linspace(0, duration, 1000, endpoint=False)

# Generate the triangular wave
original_signal = sawtooth(2 * np.pi * signal_frequency * time_continuous, width=0.5)

# Generate the time vector for the sampled signal
sampling_period = 1 / sampling_frequency
time_sampled = np.arange(0, duration, sampling_period)

# Sample the triangular wave
sampled_signal = sawtooth(2 * np.pi * signal_frequency * time_sampled, width=0.5)

# Reconstruct the signal using linear interpolation
reconstructed_signal = np.interp(time_continuous, time_sampled, sampled_signal)

# Plot the continuous signal, sampled signal, and reconstructed signal
plt.figure(figsize=(12, 6))

# Plot the original continuous signal
plt.plot(time_continuous, original_signal, label='Original Signal (Continuous)', linewidth=2, color='blue')

# Plot the sampled signal
plt.stem(time_sampled, sampled_signal, label='Sampled Signal', linefmt='r-', markerfmt='ro', basefmt=' ')

# Plot the reconstructed signal
plt.plot(time_continuous, reconstructed_signal, label='Reconstructed Signal', linestyle='--', linewidth=2, color='green')
```

jupyter Untitled Last Checkpoint: 1 minute ago

File Edit View Run Kernel Settings Help Trusted

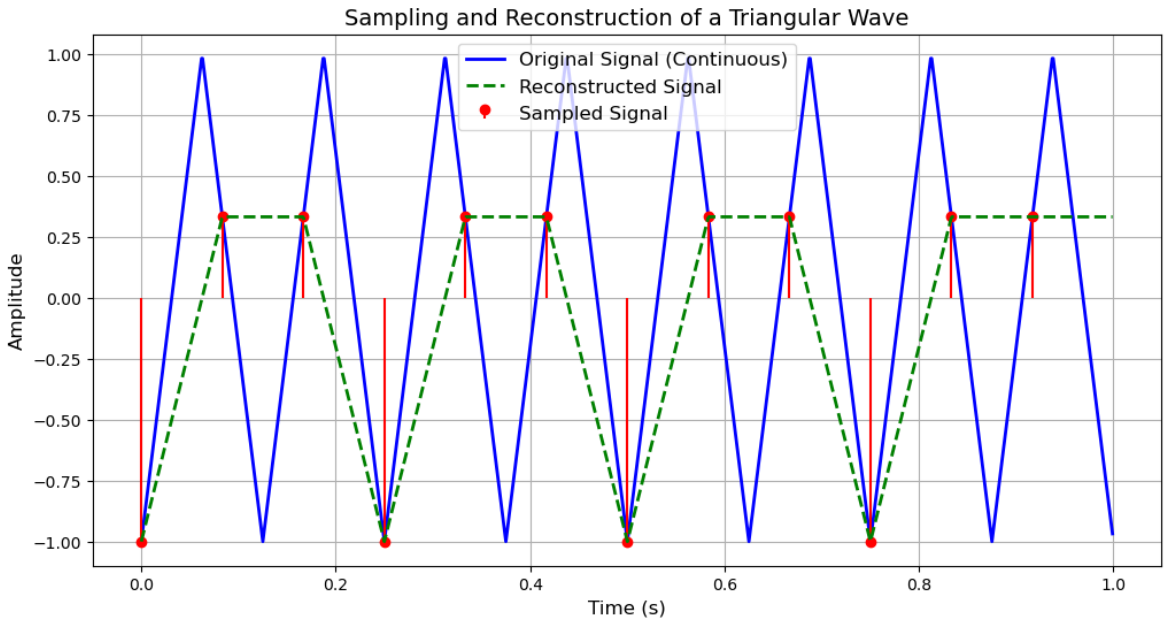
Code JupyterLab Python 3 (ipykernel)

```
# Reconstruct the signal using linear interpolation
reconstructed_signal = np.interp(time_continuous, time_sampled, sampled_signal)

# Plot the continuous signal, sampled signal, and reconstructed signal
plt.figure(figsize=(12, 6))

# Plot the original continuous signal
plt.plot(time_continuous, original_signal, label='Original Signal (Continuous)', linewidth=2, color='blue')

# Plot the sampled signal
plt.stem(time_sampled, sampled_signal, label='Sampled Signal', linefmt='r-', markerfmt='ro', basefmt=' ')

# Plot the reconstructed signal
plt.plot(time_continuous, reconstructed_signal, label='Reconstructed Signal', linestyle='--', linewidth=2, color='green')

# Add labels, legend, and grid
plt.title('Sampling and Reconstruction of a Triangular Wave', fontsize=14)
plt.xlabel('Time (s)', fontsize=12)
plt.ylabel('Amplitude', fontsize=12)
plt.legend(fontsize=12)
plt.grid(True)

# Show the plot
plt.show()
```




Link to remote repozytorium (e.g. GitHub)
https://github.com/TomekPietrzyk/DSP_2024_NS

## 4. Outcomes:


Sampling and Reconstruction of a Triangular Wave
Original Signal (Continuous)
Reconstructed Signal
Sampled Signal
Amplitude
Time (s)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

**5. Conclusions:**

The experiment successfully demonstrated the processes of signal sampling and reconstruction using a triangular wave with a frequency of 8 Hz, sampled at 12 Hz. The sampling frequency adhered to the Nyquist criterion, ensuring no aliasing occurred, as evidenced by the absence of distortion in the sampled data. Reconstruction through linear interpolation effectively restored the original signal's characteristics, closely resembling the continuous waveform. Visualization of the original, sampled, and reconstructed signals confirmed the theoretical principles, highlighting the preservation of the triangular wave's periodicity and shape throughout the process. The results validate the adequacy of the chosen sampling frequency and emphasize the utility of interpolation for signal reconstruction in practical applications