REPORT

Subject: Digital Signal Processing
Lecturer: prof. dr hab. Vasyl Martsenyuk

Laboratory #4 Date: 21.12.2024 Topic: Solve the tasks for: - sampling and reconstruction - coding and decoding. Second variant (2)	Bartosz Bieniek IT Science II degree, 1 semester, gr.A

1. Task:

Solve the tasks for:
- sampling and reconstruction
- coding and decoding

Task 1: Sampling and Reconstruction, variant two
Task: Analyze a cosine wave with frequency f = 20 Hz, sampled at fs = 25 Hz.
Task 2: Delta Encoding and Decoding, variant two
Task: Encode and decode the signal [8, 10, 12, 12, 14] using delta coding.

2. Code description [Github Repository]

Solve the tasks for:

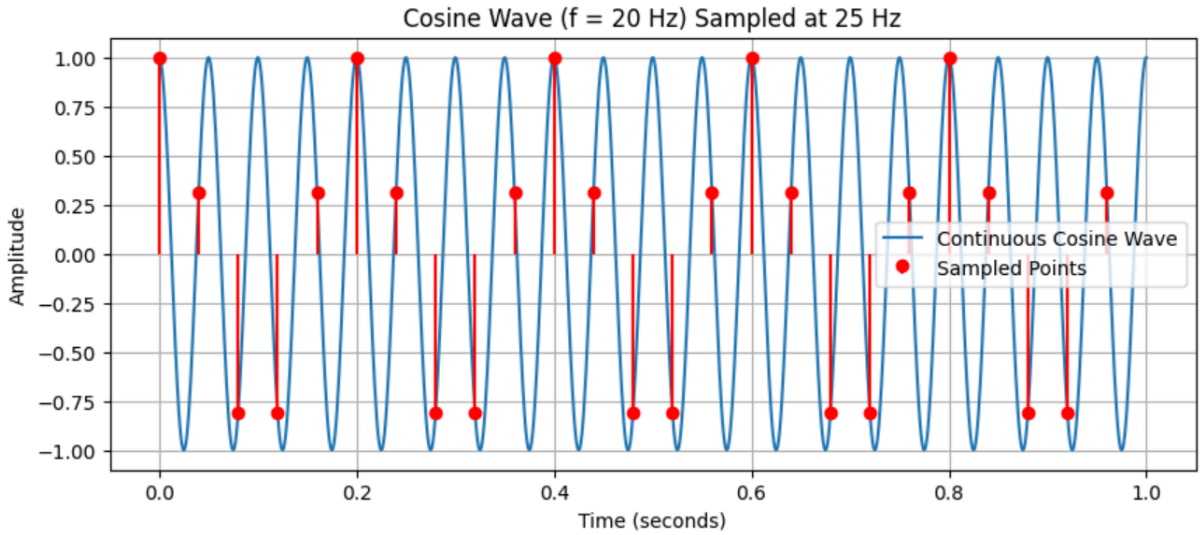
- sampling and reconstruction
- coding and decoding Variant 2

```
import numpy as np ···

# Task 1: Sampling and Reconstruction, variant two
# Task: Analyze a cosine wave with frequency f = 20 Hz, sampled at fs = 25 Hz.

f = 20
fs = 25
duration = 1

# Generate time vectors
t_continuous = np.linspace(0, duration, 1000)  # continuous time points
t_sampled = np.arange(0, duration, 1/fs)  # sampled time points

# Generate signals
x_continuous = np.cos(2 * np.pi * f * t_continuous)  # continuous signal
x_sampled = np.cos(2 * np.pi * f * t_sampled)  # sampled signal

plt.figure(figsize=(10, 4))
plt.plot(t_continuous, x_continuous, label='Continuous Cosine Wave')
plt.stem(t_sampled, x_sampled, linefmt='r', markerfmt='ro', basefmt=' ', label='Sampled Points')
plt.title(f'Cosine Wave (f = {f} Hz) Sampled at {fs} Hz')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid(True)
plt.show()
```

T.D. 1. Source code

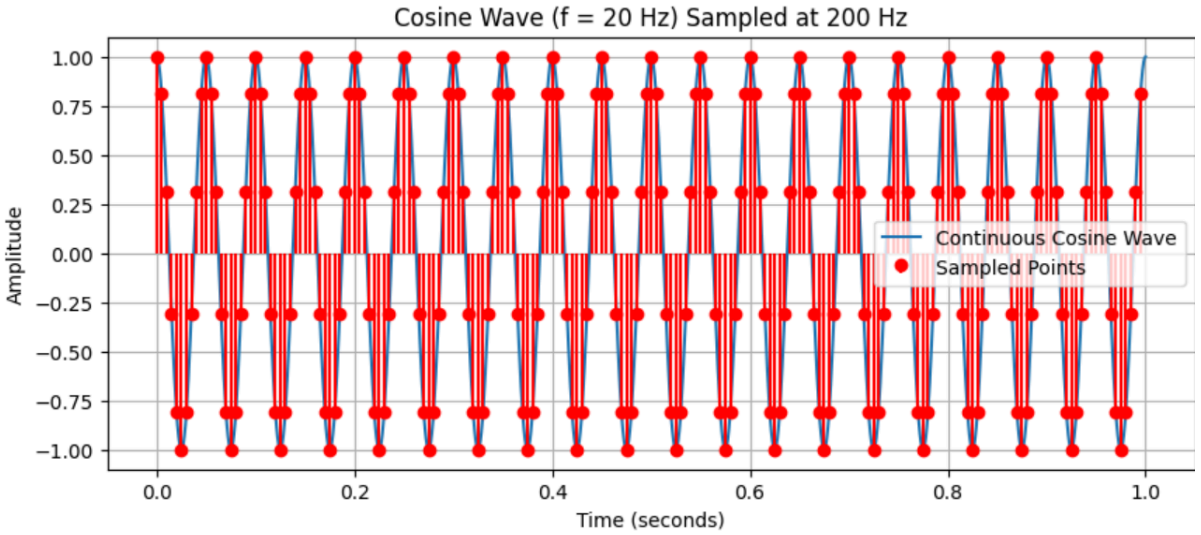
The application was created to tackle two areas of digital signal processing: sampling and reconstruction, as well as delta algorithm encoding and decoding.

A cosine wave of 20 Hz was analyzed as the signals in the first portion. The signal itself was sampled at 25 Hz. A continous and sampled version of the signal was created, and the later was generated over one second while being digitized at specific time intervals. To demonstrate the straightforward problem of sampling, both signals were drawn out to do side by side comparison with the waveform.



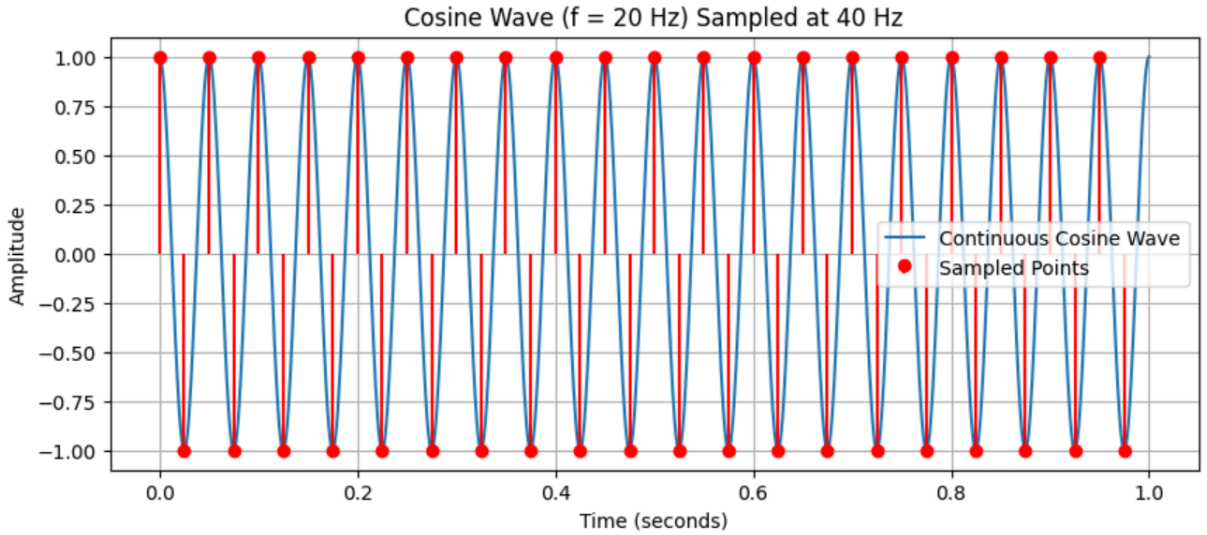
T. D. 2. Cosine wave analysis with a frequency of f = 20Hz and 25Hz samples.

Some signal information loss is visible, as the sampled points fail to track every crest and trough of the cosine wave. The red points and vertical stems represent the sampled data at 25 Hz. Due to the sampling frequency being only slightly higher than the signal frequency (20 Hz), the plot illustrates clear undersampling, Increasing the fs value cause drawing more sampled points:



T. D. 3. Cosine wave analysis with a frequency of f = 20Hz and 200Hz samples.

The sampling frequency (25 Hz) does not adequately capture all the nuances of the 20 Hz wave, as per the Nyquist theorem, which recommends sampling at least at twice the signal frequency (40 Hz in this case). Sampling frequency equal 400Hz causes oversampling.



T. D. 4. Cosine wave analysis with a frequency of f = 20Hz and 40Hz samples.

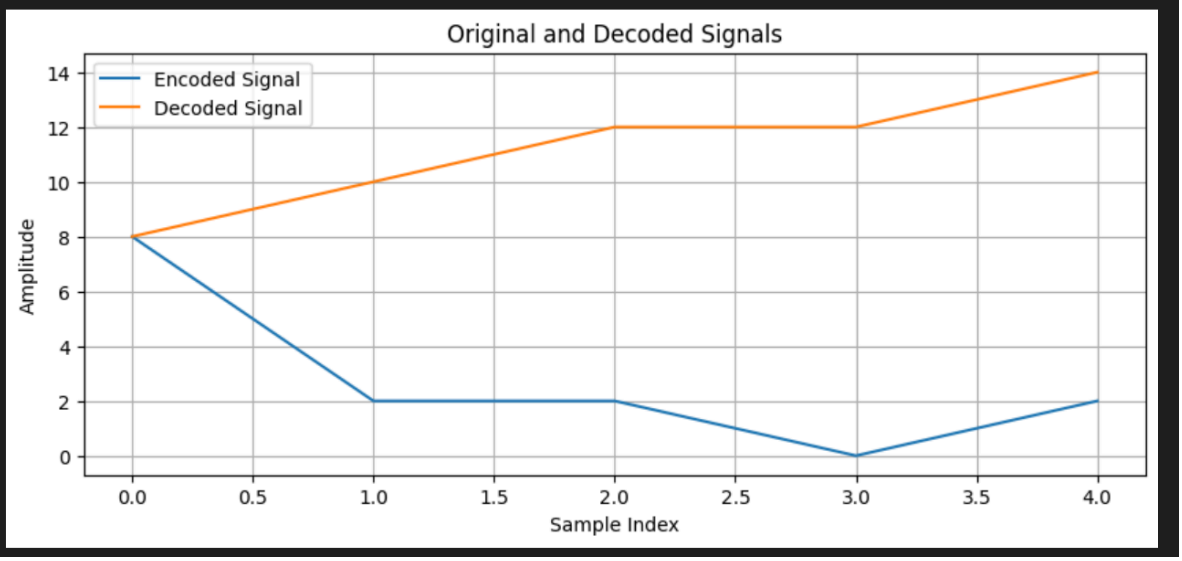
The second part of the program undertook an different focus, delta encoding and decoding. The input values [8, 10, 12, 12, 14] were altered by encoding the signal, the encoding being constructed by differencing every two adjacent values. The first term was simply kept and all other terms were the differences. The original signal contained in the encoded differences will be reconstructed through the decoding process that equals the differcons by description. An assertion was employed to test if the constructed signal matched the original one, which certainly proved right.

```python
signal = [8, 10, 12, 12, 14]

# Delta Encoding
encoded_signal = [signal[0]]
for i in range(1, len(signal)):
    encoded_signal.append(signal[i] - signal[i - 1])

print("Delta Encoded Signal:", encoded_signal)

# Delta Decoding
decoded_signal = [encoded_signal[0]]
for i in range(1, len(encoded_signal)):
    decoded_signal.append(decoded_signal[i - 1] + encoded_sig

print("Delta Decoded Signal:", decoded_signal)

assert decoded_signal == signal, "Decoding failed: Decoded si

plt.figure(figsize=(10, 4))
plt.plot(encoded_signal, label='Encoded Signal')
plt.plot(decoded_signal, label='Decoded Signal')
plt.title('Original and Decoded Signals')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')
plt.legend()
plt.grid(True)
plt.show()
```

T. D. 5. Delta encoding and decoding source code

Computation and visualization were the means within which fundamental concepts of signal processing and data compression were illustrated and successfully demonstrated within this delicate program.

```
Delta Encoded Signal: [8, 2, 2, 0, 2]
Delta Decoded Signal: [8, 10, 12, 12, 14]
```



T. D. 6. Delta encoding and decoding source code

3. Conclusions

The analysis of the cosine wave demonstrated the majority of selecting an specified sampling frequency. The sampling rate of 25 Hz for a 20 Hz signal was insufficient to fully capture the waveform's details, approaching aliasing conditions. This highlights the need to adhere to the Nyquist criterion, which

requires sampling at least two times the signal frequency for accurate reconstruction. Greater value than that may lead to oversampling.

The delta coding task successfully demonstrated an efficient method for signal compression by encoding only the differences between consecutive values. The decoding process verified the effectiveness of this technique, as the original signal was accurately reconstructed. Delta coding proves to be a simple yet effective approach for data compression in digital signal processing.