ECE 2414: Digital Communications LAB 1

Analysis of Sampling Theorem Using MATLAB or Python

Martin Wafula
Lecturer, ECE Department
Multimedia University of Kenya
Magadi Road

Submission instructions

- This Lab is **20 marks** and will be averaged with the other labs for this course. These marks will be awarded for accuracy, conciseness, elegance and technical details.
- The code file (e.g. *.m* or *.mlx* for MATLAB and *.ipynb* or *.py* for Python) and lab report should be uploaded on Github and access given to my email for me to mark and review the work.
- The LAB report should be typed, preferably on Latex such as overleaf.com. The diagrams you obtain running the simulations should be added to the lab report and explained.
- The MATLAB code should be credited to *Copyright @ Dr. Sudip Mandal, Assistant Professor, Jalpaiguri Government Engineering College, ECE Department, West Bengal, India, PIN-735102, Digital Communication Lab.* Here is Dr. Mandal's Youtube channel where he does explain these Labs on youtube and has links to the MATLAB files. Digital Communication Labs using MATLAB by Dr Mandal. Please reference it if you use the given MATLAB code.
- Lab reports must be submitted by **28 November 2024 at 12.00 noon**.

Objective

To analyze and verify the Sampling Theorem, reconstruct the original signal from sampled data, and perform quantization using MATLAB or Python.

Experiment 1: Analysis of Sampling Theorem

Theory

The Sampling Theorem states that a continuous-time signal can be fully reconstructed from its samples if it is band-limited, and the sampling frequency is at least twice the maximum frequency in the signal (Nyquist rate).

Procedure

1. **Define the Message Signal**: Create a signal with 1 Hz and 3 Hz sinusoidal components.
2. **Plot the Message Signal**: Display the time-domain representation of the message signal.
3. **Compute and Plot the Spectrum**: Use Fast Fourier Transform (FFT) to observe the signal in the frequency domain.
4. **Sample the Message Signal**: Choose a sampling period, for example, 0.02 seconds (50 Hz sampling rate).
5. **Plot the Sampled Signal**: Display the discrete-time version of the sampled signal.
6. **Compute and Plot the Spectrum of the Sampled Signal**: Apply FFT on the sampled signal to analyze frequency components.

Experiment 2: Reconstruction from Sampled Signal

Objective

To reconstruct the original signal from the sampled data using Low Pass Filtering (LPF).

Procedure

1. **Define Parameters and Generate Signal**: Define the message signal and sampling parameters as in Experiment 1.
2. **Upsample and Zero-fill the Sampled Signal**: Increase the rate of the sampled signal by inserting zeros between samples.
3. **Frequency Spectrum of the Sampled Signal**: Calculate the FFT of the upsampled signal to observe its frequency spectrum.
4. **Design a Low Pass Filter (LPF)**: Define the transfer function of an LPF to retain frequencies from -10 Hz to 10 Hz.
5. **Filter the Sampled Signal**: Multiply the FFT of the upsampled signal with the LPF transfer function.
6. **Inverse FFT for Time Domain Representation**: Use inverse FFT to convert the filtered signal back to the time domain and compare with the original signal.

MATLAB Code for Experiment 1: Sampling Theorem Analysis

```
clear all;
close all;
clc;

tot = 1;
td = 0.002;
t = 0:td:tot;
x = sin(2*pi*t) - sin(6*pi*t);

figure;
plot(t, x, 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude');
title('Input Message Signal');
grid on;

L = length(x);
Lfft = 2^nextpow2(L);
fmax = 1/(2*td);
Faxis = linspace(-fmax, fmax, Lfft);
Xfft = fftshift(fft(x, Lfft));
figure;
plot(Faxis, abs(Xfft));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Spectrum of Input Message Signal');
grid on;

ts = 0.02;
n = 0:ts:tot;
x_sampled = sin(2*pi*n) - sin(6*pi*n);

figure;
stem(n, x_sampled, 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude');
title('Sampled Signal');
grid on;

x_sampled_upsampled = upsample(x_sampled, round(ts/td));
Lffu = 2^nextpow2(length(x_sampled_upsampled));
fmaxu = 1/(2*td);
Faxisu = linspace(-fmaxu, fmaxu, Lffu);
Xfftu = fftshift(fft(x_sampled_upsampled, Lffu));
figure;
```

```
plot(Faxisu, abs(Xfftu));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Spectrum of Sampled Signal');
grid on;
```

MATLAB Code for Experiment 2: Reconstruction from Sampled Signal

```
clear all;
close all;
clc;

tot=1;
td=0.002;
t=0:td:tot;
x=sin(2*pi*t)-sin(6*pi*t);

ts=0.02;
Nfactor=round(ts/td);
xsm=downsample(x,Nfactor);
xsmu=upsample(xsm,Nfactor);

Lffu=2^nextpow2(length(xsmu));
fmaxu=1/(2*td);
Faxisu=linspace(-fmaxu,fmaxu,Lffu);
xfftu=fftshift(fft(xsmu,Lffu));

figure(1);
plot(Faxisu,abs(xfftu));
xlabel('Frequency');
ylabel('Amplitude');
title('Spectrum of Sampled Signal');
grid;

BW=10;
H_lpf=zeros(1,Lffu);
H_lpf(Lffu/2-BW:Lffu/2+BW-1)=1;

figure(2);
plot(Faxisu,H_lpf);
xlabel('Frequency');
ylabel('Amplitude');
title('Transfer function of LPF');
grid;

x_recv=Nfactor*((xfftu)).*H_lpf;
```

```
figure(3)
plot(Faxisu,abs(x_recv));
xlabel('Frequency');
ylabel('Amplitude');
title('Spectrum of LPF Output');
grid;

x_recv1=real(ifft(fftshift(x_recv)));
x_recv2=x_recv1(1:length(t));

figure(4)
plot(t,x,'r',t,x_recv2,'b--','LineWidth',2);
xlabel('Time');
ylabel('Amplitude');
title('Original vs. Reconstructed Signal');
grid;
```

Python Code for Experiment 1 and 2: Sampling and Reconstruction

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters for message signal
tot = 1
td = 0.002
t = np.arange(0, tot, td)
x = np.sin(2 * np.pi * t) - np.sin(6 * np.pi * t)

plt.figure()
plt.plot(t, x, linewidth=2)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Input Message Signal')
plt.grid(True)
plt.show()

# Spectrum of the message signal
L = len(x)
Lfft = 2 ** int(np.ceil(np.log2(L)))
fmax = 1 / (2 * td)
Faxis = np.linspace(-fmax, fmax, Lfft)
Xfft = np.fft.fftshift(np.fft.fft(x, Lfft))
plt.figure()
plt.plot(Faxis, np.abs(Xfft))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.title('Spectrum of Input Message Signal')
plt.grid(True)
plt.show()

# Sampling
ts = 0.02
n = np.arange(0, tot, ts)
x_sampled = np.sin(2 * np.pi * n) - np.sin(6 * np.pi * n)
plt.figure()
plt.stem(n, x_sampled, basefmt=" ", use_line_collection=True)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Sampled Signal')
plt.grid(True)
plt.show()

# Reconstruction
Nfactor = int(ts / td)
xsmu = np.zeros(len(t))
xsmu[::Nfactor] = x_sampled
Lffu = 2 ** int(np.ceil(np.log2(len(xsmu))))
fmaxu = 1 / (2 * td)
Faxisu = np.linspace(-fmaxu, fmaxu, Lffu)
Xfftu = np.fft.fftshift(np.fft.fft(xsmu, Lffu))
BW = 10
H_lpf = np.zeros(Lffu)
H_lpf[Lffu // 2 - BW:Lffu // 2 + BW] = 1
x_recv = Nfactor * Xfftu * H_lpf
x_recv_time = np.real(np.fft.ifft(np.fft.fftshift(x_recv)))
plt.figure()
plt.plot(t, x, 'r', t, x_recv_time[:len(t)], 'b--', linewidth=2)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Original vs. Reconstructed Signal')
plt.grid(True)
plt.show()
```

Discussion Questions

1. **Theory**: Explain why the Nyquist rate is important for the sampling process.
2. **Spectrum Analysis**: Describe the frequency spectrum of the sampled signal. How does it change with different sampling rates?
3. **Reconstruction**: Discuss how the low pass filter affects the reconstruction of the sampled signal. What would happen if the filter's bandwidth was reduced or increased beyond the Nyquist limit?

4. **Aliasing**: What is aliasing, and how does it appear in the spectrum of the sampled signal? How can you avoid aliasing in a practical sampling system?

5. **Effects of Undersampling**: If the sampling rate is below the Nyquist rate, how would this affect the reconstruction? Describe how this would look in both the time and frequency domains.

6. **Practical Sampling Rates**: In practical applications, why might we choose a sampling rate higher than the minimum Nyquist rate?

Extension Task: Quantization

For digital communication, signals are not only sampled but also quantized to discrete amplitude levels. Extend the experiment by implementing quantization for the sampled signal using both MATLAB and Python.

Quantization Procedure

1. **Quantization Levels**: Define a number of quantization levels (e.g., 8, 16, or 32 levels).

2. **Apply Quantization**: Quantize the sampled signal by rounding each sample to the nearest discrete level.

3. **Plot Quantized Signal**: Display the quantized signal alongside the original sampled signal to observe quantization effects.

4. **Quantization Error Analysis**: Calculate and plot the quantization error, defined as the difference between the original sampled signal and the quantized signal.

MATLAB Code for Quantization Example

```
% Define quantization levels
levels = 16;
x_min = min(x_sampled);
x_max = max(x_sampled);
step = (x_max - x_min) / levels;

% Quantize the sampled signal
x_quantized = step * round((x_sampled - x_min) / step) + x_min;

% Plot quantized vs. sampled signal
figure;
stem(n, x_sampled, 'r', 'LineWidth', 1.5); hold on;
stem(n, x_quantized, 'b--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Amplitude');
title('Sampled Signal vs. Quantized Signal');
legend('Sampled Signal', 'Quantized Signal');
```

```
grid on;

% Quantization error
quantization_error = x_sampled - x_quantized;
figure;
stem(n, quantization_error, 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Error');
title('Quantization Error');
grid on;
```

Python Code for Quantization Example

```
# Define quantization levels
levels = 16
x_min = np.min(x_sampled)
x_max = np.max(x_sampled)
step = (x_max - x_min) / levels

# Quantize the sampled signal
x_quantized = step * np.round((x_sampled - x_min) / step) + x_min

# Plot quantized vs. sampled signal
plt.figure()
plt.stem(n, x_sampled, 'r', markerfmt='ro', basefmt=" ", use_line_collection=True, la
plt.stem(n, x_quantized, 'b--', markerfmt='bo', basefmt=" ", use_line_collection=True
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Sampled Signal vs. Quantized Signal')
plt.legend()
plt.grid(True)
plt.show()

# Quantization error
quantization_error = x_sampled - x_quantized
plt.figure()
plt.stem(n, quantization_error, basefmt=" ", use_line_collection=True)
plt.xlabel('Time (s)')
plt.ylabel('Error')
plt.title('Quantization Error')
plt.grid(True)
plt.show()
```

Additional Questions

1. **Quantization Error**: How does the quantization error change with the number of quantization levels? Describe the relationship between quantization levels and signal quality.

2. **Signal-to-Noise Ratio (SNR)**: Calculate the Signal-to-Noise Ratio (SNR) for different quantization levels and comment on how quantization affects the SNR.

3. **Bitrate Calculation**: For a given sampling rate and quantization level, calculate the bitrate required to represent the signal in a digital communication system. How does increasing the sampling rate or the number of quantization levels impact the bitrate?

4. **Practical Applications**: Describe how sampling and quantization are used together in practical digital communication systems. Give examples of real-world systems where this is critical.

5. **Trade-offs**: Discuss the trade-offs between sampling rate, quantization levels, and signal quality in digital systems. How might a system designer choose these parameters based on application requirements?