Netaji Subhas University of Technology

LAB REPORT

DATA COMMUNICATIONS

Name	**Kushagra Lakhwani**
Roll No.	***2021UCI8036***
Semester	**4th**
Course	***CICPC12***

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

April 11, 2023

Abstract

The practical lab report *"Data Communications"* is the original and unmodified content submitted by *Kushagra Lakhwani* (Roll No. 2021UCI8036).

The report is submitted to *Mr. Pattetti*, Department of Computer Science and Engineering, NSUT, Delhi, for the partial fulfillment of the requirements of the course (CICPC12).

Index

1 Fourier Transform

We plot a Rectangular Pulse Signal $x(t)$ in *Matlab* and explore its magnitude and phase spectrum of its Fourier Transform.

1.1 Matlab Code

```
close all;

% parameters of a rectangular pulse signal
w = 10;                      % width
A = 1;                       % amplitude
t = -10:0.01:10;             % time vector
xt = A * rectpuls(t, w);     % rectangular pulse signal

% plot the rectangular pulse signal in the first subplot
subplot(2, 2, 1)
plot(t, xt)
xlabel('Time')
ylabel('Amplitude')
title('Rectangular pulse')

% define a range of frequencies and compute the Fourier transform at each frequency
w = -8 * pi:0.01:8 * pi;  % range of frequencies
for i = 1:length(w)
    xw(i) = trapz(t, xt .* exp(-1i * w(i) .* t));  % Fourier transform
end

% plot the Fourier transform in the second subplot
subplot(2, 2, 2)
plot(w, xw)
title('Fourier transform of rect pulse: Sampling signal')
xlabel('Frequency')
ylabel('Amplitude')

% plot the magnitude spectrum of the Fourier transform in the third subplot
subplot(2, 2, 3)
plot(w, abs(xw))
title('Magnitude spectrum')
xlabel('Frequency')
ylabel('Amplitude')

% plot the phase spectrum of the Fourier transform in the fourth subplot
subplot(2, 2, 4)
plot(w, angle(xw))
title('Phase spectrum')
xlabel('Frequency')
ylabel('Amplitude')
```

1.2 Output

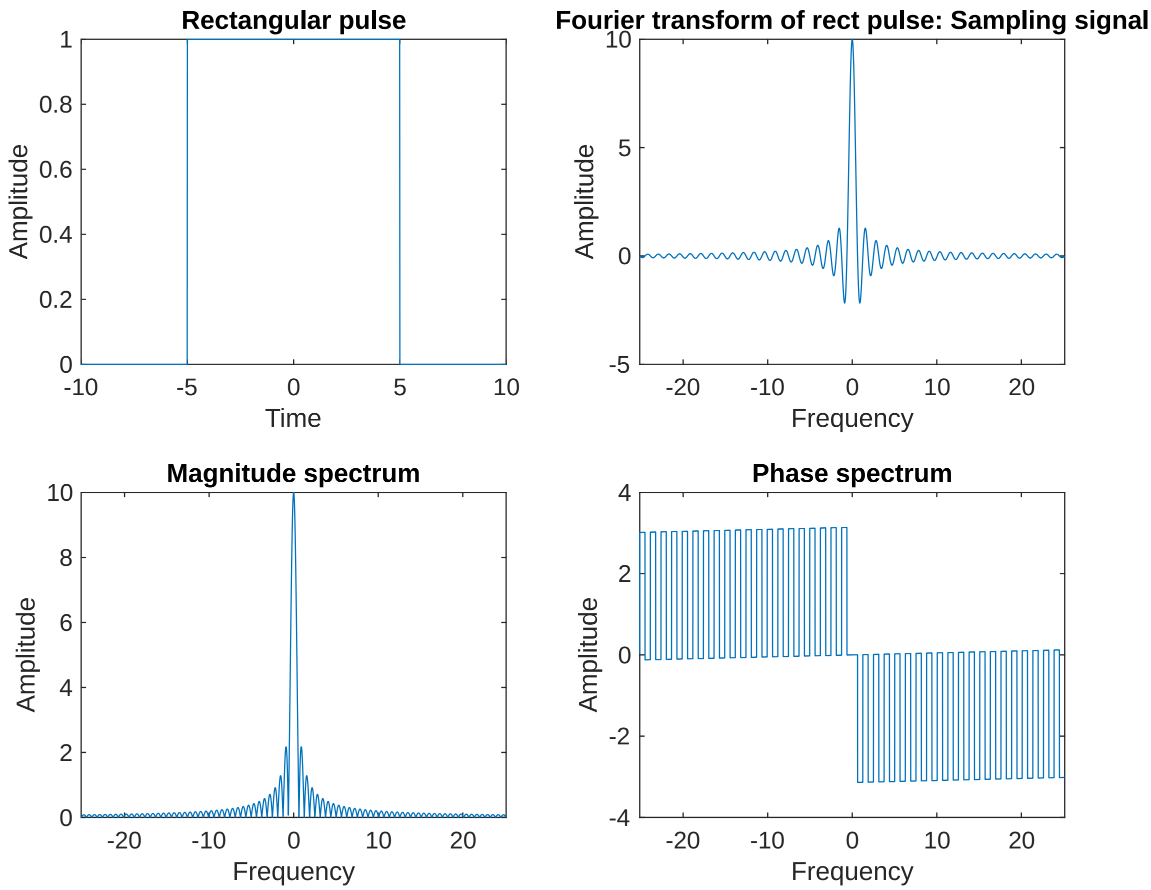

Figure 1: Fourier Transform

2 Uniform Distribution

Generate uniform random numbers and plot their density function. Find the mean and variance

2.1 Matlab Code

```
% Define the parameters of the uniform distribution
a = 1; % Lower bound
b = 6; % Upper bound

% Generate 1000 random numbers from the uniform distribution
rng(1); % Set the random seed for reproducibility
X = a + (b - a) * rand([1, 1000]);

% Compute the mean and variance of the generated numbers
```

```
mu = mean(X);
sigma2 = var(X);

% Define the range of x values to plot
x = linspace(a - 1, b + 1, 1000);

% Compute the uniform distribution density function
f = ones(size(x)) ./ (b - a);

% Plot the uniform distribution density function
plot(x, f, 'LineWidth', 2);
hold on;

% Plot a vertical line at the mean value
ymin = 0;
ymax = max(f) * 1.5;
line([mu mu], [ymin ymax], 'Color', 'r', 'LineStyle', '--', 'LineWidth', 2);

% Set the plot limits and labels
xlim([a - 2, b + 2]);
ylim([ymin, ymax]);
xlabel('x');
ylabel('Probability density');
title('Uniform distribution');
legend(sprintf('Mean = %.2f\nVariance = %.2f', mu, sigma2));
```

2.2 Output

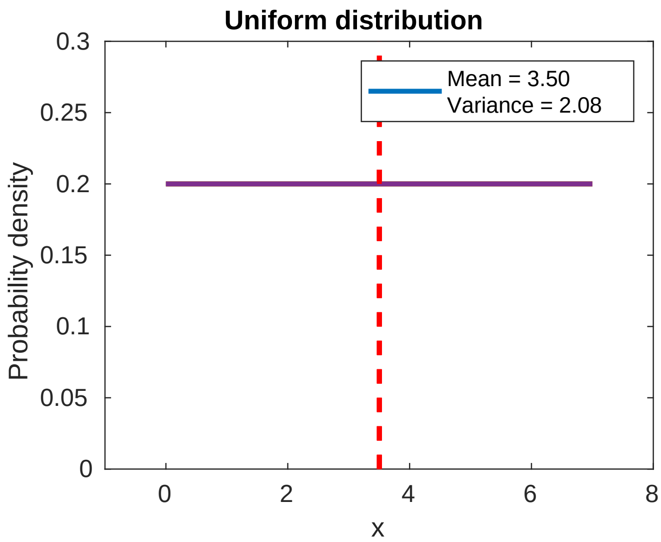

Figure 2: Uniform Distribution

3 Normal Distribution

Using the Gaussian random numbers we find the mean and variance.

3.1 Matlab Code

```
data = randn(1000, 1); % Generate random numbers

histogram(data, 20, 'Normalization', 'pdf');
hold on;

mu = mean(data);
sigma = std(data);

x = linspace(min(data), max(data), 100); % Define x values for Gaussian curve
y = normpdf(x, mu, sigma); % Calculate y values for Gaussian curve

% Overlay Gaussian curve
plot(x, y, 'LineWidth', 2);

% Add title and labels
title('Histogram of Random Data with Gaussian Fit');
xlabel('Data Value');
ylabel('Probability Density');

hold off;
```

3.2 Output

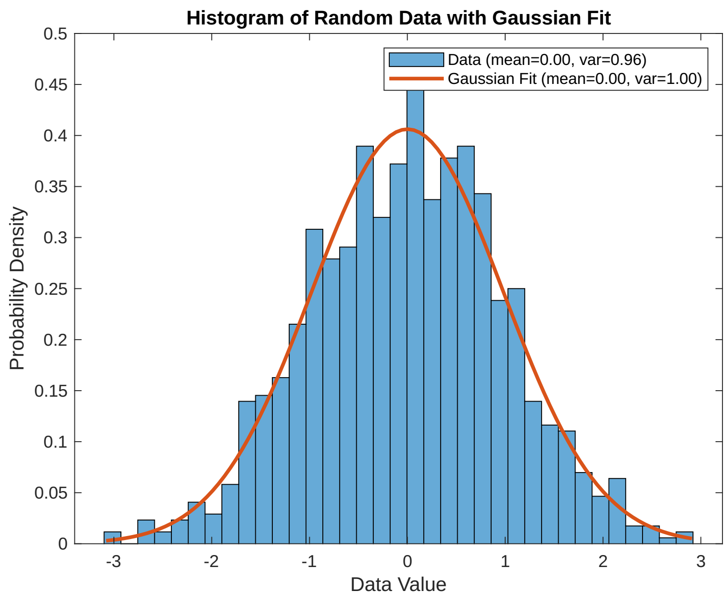

Figure 3: Gaussian Distribution

4 Quantization: Uniform

Computing the Signal to quantization Noise ratio of Uniform Quantization. Plot SNQR vs. Quantization levels.

4.1 Matlab Code

```
close all; clc;

% Define the message signal
t = linspace(0, 1, 1000);
fm = 1; % message signal frequency
Am = 1; % message signal amplitude
m = Am * sin(2 * pi * fm * t);

% Define the maximum number of quantization levels
n_max = 4;

% Initialize vectors to store SQNR and number of quantization levels
sqnr = zeros(1, n_max);
levels = 1:n_max;

% Compute the SQNR for each quantization level
for i = 1:n_max
    L = 2 ^ i;
    delta = (max(m) - min(m)) / (L - 1);
    m_quantized = delta * round(m / delta);
    noise = m - m_quantized;
    power_m = sum(m .^ 2) / length(m);
    power_noise = sum(noise .^ 2) / length(noise);
    sqnr(i) = power_m / power_noise;
end

% Plot the message signal and the quantized signal for n=4
subplot(2, 1, 1);
plot(t, m, 'b', 'LineWidth', 2);
hold on;
plot(t, m_quantized, 'r', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude');
title('Message signal and Quantized signal');
legend('Message signal', 'Quantized signal');

% Plot the number of quantization levels vs. the SQNR
subplot(2, 1, 2);
plot(sqnr, levels, 'LineWidth', 2);
ylabel('Quantization levels');
xlabel('Signal to Quantisation Noise Ratio (dB)');
title('Number of quantization levels vs. SQNR');
```

4.2 Output

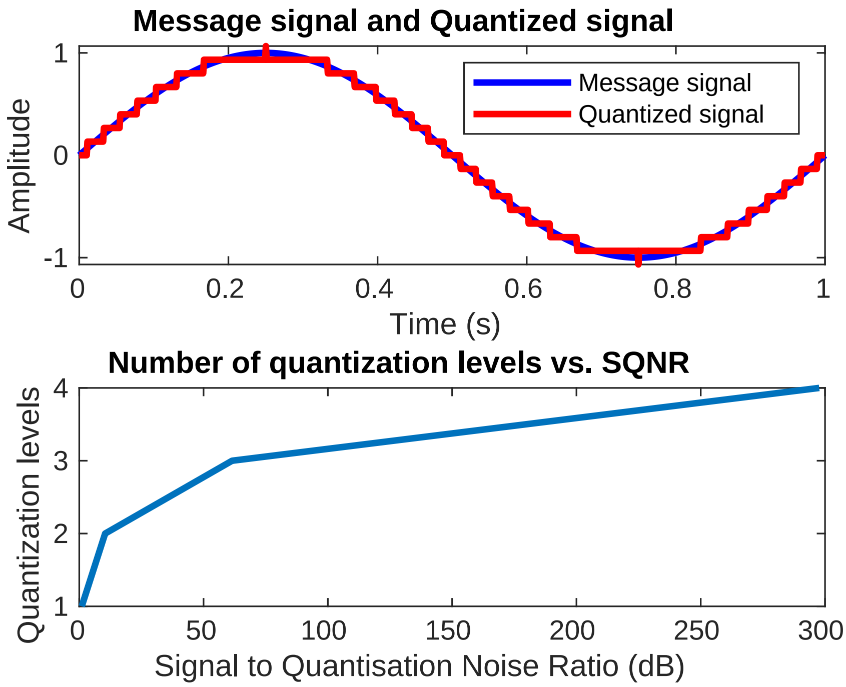

Figure 4: SQNR vs Quantization

5 Quantization: Non-Uniform

Computing SNR of Non-Uniform Quantization and Plot SNR vs. Quantization Levels

5.1 Matlab Code

```
% Program to Compute SNR of Non-Uniform Quantization and Plot the SNR vs. Quantization Levels
close all; clc;

% Signal Parameters
N = 10000;                  % Number of samples in the signal
f = 1;                      % Signal frequency
Fs = 1000;                  % Sampling frequency
t = (0:N - 1) / Fs;         % Time vector
x = sin(2 * pi * f * t);    % Signal

% Quantization Parameters
L = 2:20;                       % Number of quantization levels to try
b = log2(L);                    % Number of bits to represent each level
Delta = 2 ./ (L - 1);           % Step size of the quantization levels
SQNR = zeros(length(L), 1);     % To store the Signal to Quantization Noise Ratio (SQNR) for each qu
```

```
% Non-Uniform Quantization
for i = 1:length(L)
    q = zeros(size(x));
    % Compute quantization levels
    V = [- (L(i) - 1) / 2:1:(L(i) - 1) / 2] * Delta(i);
    % Quantize the signal
    for j = 1:N
        [val, index] = min(abs(x(j) - V));
        q(j) = V(index);
    end

    % Compute the SQNR
    noise = x - q;
    signal_power = sum(x .^ 2) / N;
    noise_power = sum(noise .^ 2) / N;
    SQNR(i) = 10 * log10(signal_power / noise_power);
end

% Plot the SNR vs. Quantization Levels
figure;
plot(b, SQNR, 'b-o', 'LineWidth', 2);
xlabel('Number of Bits');
ylabel('Signal to Quantization Noise Ratio (dB)');
grid on;
```

5.2 Output

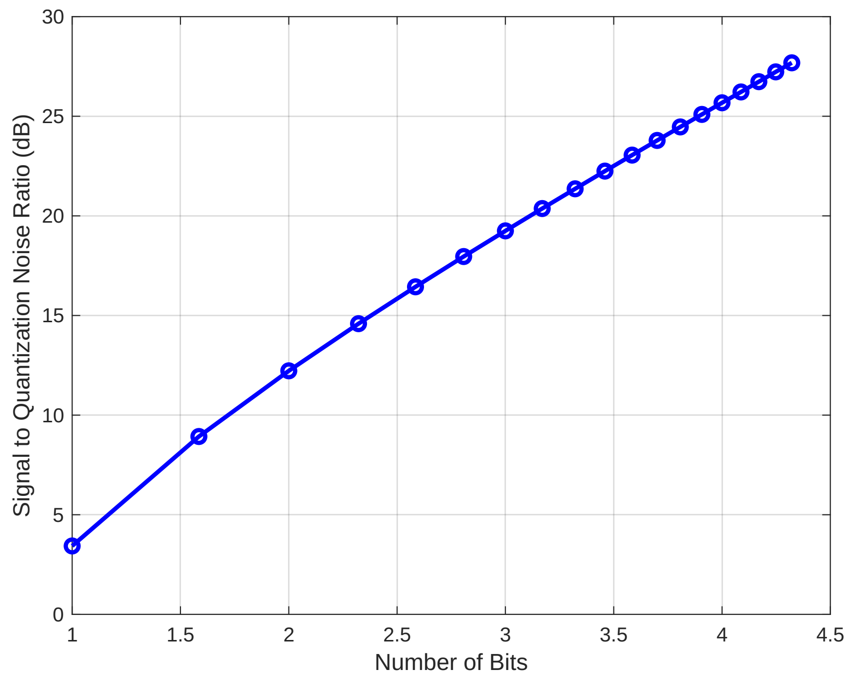

Figure 5: SQNR vs Quantization (non-uniform)

6 Pulse Code Modulation (PCM)

6.1 Theory

Pulse Code Modulation (PCM) is a technique used to digitize analog signals. The process involves three main steps: sampling, quantization, and encoding.

1. In the first step, the analog signal is sampled at regular intervals[1]. The resulting sequence of samples represents the signal in a discrete-time domain.
2. In the second step, the samples are quantized into a finite number of levels. This reduces the number of possible amplitude values that each sample can take on, resulting in a loss of information compared to the original analog signal. However, quantization allows for the signal to be represented using a fixed number of bits, which is necessary for digital storage and transmission.
3. In the third step, the quantized samples are encoded into binary code words. Each code word represents a quantization level and is assigned a unique binary code based on the number of bits used to represent it. This is typically done using a lookup table that maps each quantization level to a binary code.

To demodulate the signal, the process is reversed.

6.2 Matlab Code

```
% Define parameters
fs = 100; % Sampling frequency
f = 10; % Signal frequency
A = 1; % Signal amplitude
bits = 8; % Number of bits per sample

% Generate sinusoidal signal
t = 0:1 / fs:1 - 1 / fs; % Time vector
x = A * sin(2 * pi * f * t); % Original signal

% Sample the signal
Ts = 1 / fs; % Sampling interval
n = 0:Ts:1 - Ts; % Sample times
xs = A * sin(2 * pi * f * n); % Sampled signal

% Encode signal
L = 2 ^ bits; % Number of quantization levels
partition = linspace(-A, A, L + 1); % Quantization levels
codebook = linspace(-A + A / L, A - A / L, L); % Codebook
index = zeros(1, length(xs)); % Preallocate index vector
```

[1]The Nyquist-Shannon sampling theorem states that a signal can be perfectly reconstructed from its samples if the sampling rate is at least twice the maximum frequency of the signal.

```
for i = 1:length(xs)
    [~, ind] = min(abs(xs(i) - partition)); % Find closest quantization level
    index(i) = ind - 1; % Subtract 1 to get 0-based index
end

code = dec2bin(index, bits); % Convert to binary

% Decode signal
index_hat = bin2dec(code); % Convert binary to decimal
xq_hat = codebook(index_hat + 1); % Reconstructed quantized signal
t_hat = 0:1 / fs:1 - 1 / fs; % Time vector for reconstructed signal
x_hat = interp1(n, xq_hat, t_hat, 'linear'); % Reconstructed signal

% Demodulate signal
demod = zeros(1, length(code) * bits); % Preallocate demodulated signal

for i = 1:length(code)
    demod((i - 1) * bits + 1:i * bits) = str2double(code(i, :)); % Convert to serial binary stream
end

demod = reshape(demod, bits, length(demod) / bits)'; % Reshape into matrix
demod = bin2dec(num2str(demod)); % Convert binary to decimal
demod = demod - A; % Convert to original range

% Plot signals
subplot(5, 1, 1)
plot(t, x)
title('Original Signal')
xlabel('Time (s)')
ylabel('Amplitude')

subplot(5, 1, 2)
stem(n, xs)
title('Sampled Signal')
xlabel('Time (s)')
ylabel('Amplitude')

subplot(5, 1, 3)
stairs(1:length(code), index)
title('Encoded Signal')
xlabel('Sample')
ylabel('Quantization Index')

subplot(5, 1, 4)
plot(t_hat, x_hat)
title('Demodulated Signal')
xlabel('Time (s)')
ylabel('Amplitude')

subplot(5, 1, 5)
plot(n, xs, 'b-', n, xq_hat, 'r--')
```

```
title('Encoded and Reconstructed Signal')
xlabel('Time (s)')
ylabel('Amplitude')
legend('Original Signal', 'Reconstructed Signal', 'Location', 'south')

% Adjust spacing between subplots
set(gcf, 'Units', 'normalized', 'Position', [0.2 0.2 0.5 0.6])
set(gcf, 'DefaultAxesLooseInset', [0.1, 0.1, 0.1, 0.1])

% Save figure
saveas(gcf, 'pcm_no_quantization.pdf')
```

6.3 Output

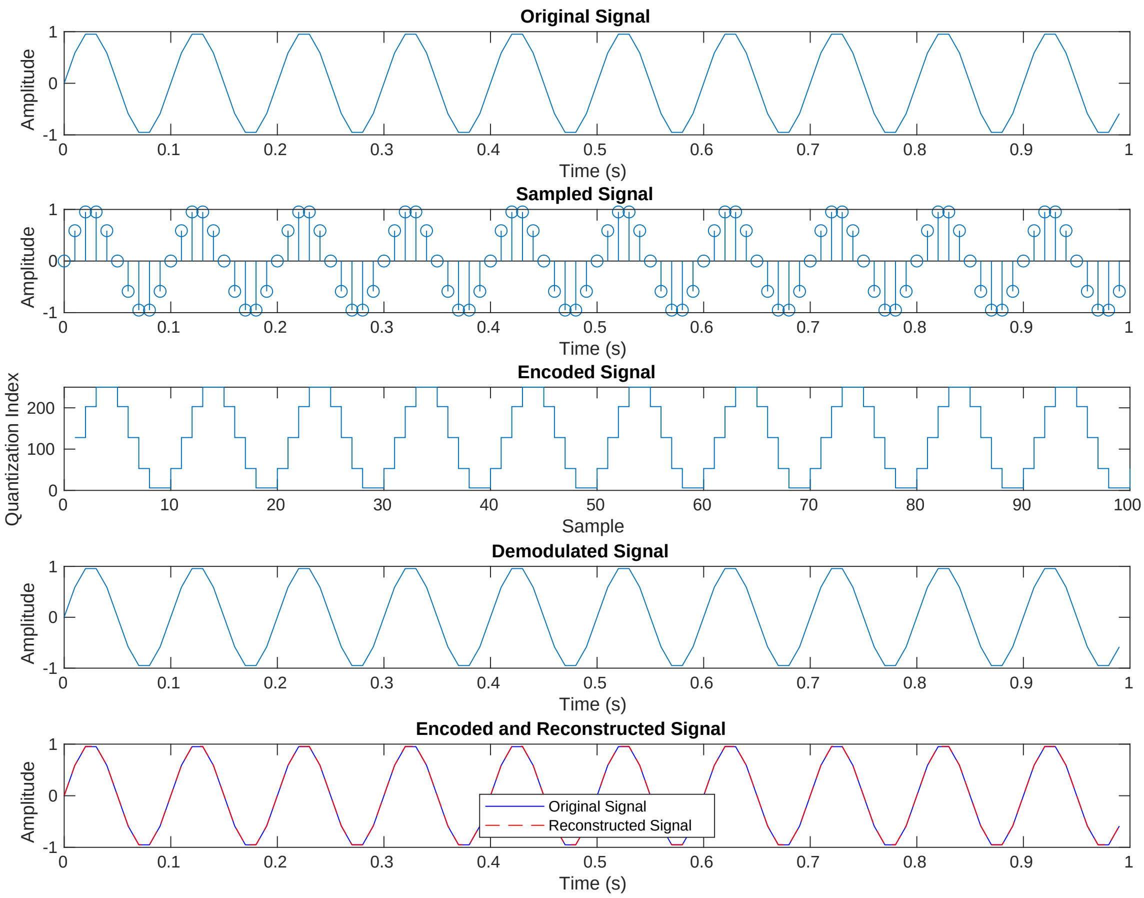

Figure 6: PCM