**COMM502 - Communication Theory**

**Final Lab Project 2024**

**Simulation of a Communication Block**

**Introduction**

This report outlines the implementation and results of simulating a modified communication block, as required by the COMM502 final lab project. The key objectives were to analyze and simulate signal sampling, quantization, encoding, and reconstruction while exploring their impact on the signal's accuracy and compression efficiency. Additionally, an error-channel simulation was implemented as a bonus task.

**Project Details**

**Input Signal Definition**

The input sinusoidal signal is defined as:

x(t)=Asin(2πft

Where:

* A=9 (calculated as the sum of 1 and 8).
* f=1000Hz.
* Duration: 0.01 seconds.

**Tasks and Results**

**1. Sampling the Input Signal**

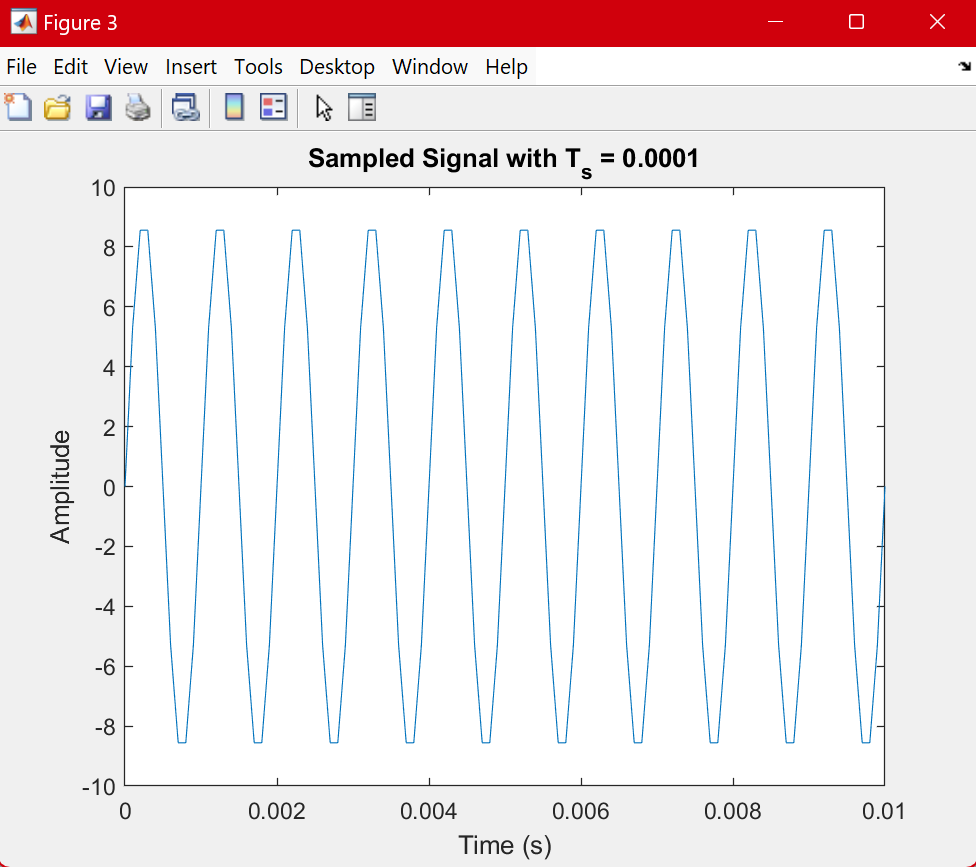
The input signal was sampled at intervals of Ts=0.001,0.0005,and 0.0001 seconds.

* **Observation**: As Ts​ decreased, the sampled signal better approximated the continuous waveform. Insufficient sampling (Ts>1/2fs) resulted in aliasing.
* **Figure 1**: Sampled signal for various Ts​ values.

A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated



**2. Quantization**

* **Uniform Quantizer**: Quantized the sampled signal using L=4,8,16. Higher L values reduced quantization errors.
* **Non-Uniform Quantizer (μ\muμ-law)**: Applied μ=255 for compressing and expanding the signal. Compared to the uniform quantizer, μ-law effectively reduced quantization noise for low-amplitude regions.

**3. Quantization Error Analysis**

Mean absolute error (MAE) and variance were computed for uniform quantization at different L values:

| **L** | **MAE** | **Variance** |
| --- | --- | --- |
| 4 | 1.70 | 4.04 |
| 8 | 0.89 | 0.93 |
| 16 | 0.34 | 0.15 |

**Figure 2**: MAE vs. L and variance vs L.

A graph with a line

Description automatically generated A graph with a line

Description automatically generated

**4. Signal-to-Quantization Noise Ratio (SQNR)**

The SQNR was calculated for uniform quantization:

SQNR=10log(Signal PowerNoise Power) [the log is base 10]

| L | **SQNR (dB)** |
| --- | --- |
| 4 | 9.98 |
| 8 | 16.32 |
| 16 | 24.08 |

**Figure 3**: SQNR vs. L.

*A screen shot of a graph

Description automatically generated*

**5. Encoding and Decoding**

* **Huffman Encoding**: Applied Huffman coding to compress quantized values.
* **Decoding**: Used the Huffman dictionary to decode the compressed signal.

**6. Signal Comparison**

* **Cross-correlation**: 0. 99907, indicating very high similarity between input and reconstructed signals.
* **Plot**: Overlapping input and reconstructed signals highlighted minor distortions due to quantization.

**Figure 4**: Input vs. reconstructed signal comparison*.*

A diagram of a signal

Description automatically generated

**7. Compression Metrics**

* **Compression Rate**:

Compression Rate=(Original Bits−Compressed Bits)/Original Bits

Compression efficiency increased with L, as Huffman encoding better exploited patterns in quantized signals.

**8. Error Channel Simulation**

A Binary Symmetric Channel (BSC) with p=0.01 was simulated.

* **Observations**:
  + Random bit flips introduced minor distortions.
  + Decoded signals were noisier, which resulted in changing the cross correlation into around -0.0008

**Figure 5**: Input vs. noisy reconstructed signal comparison.

A screen shot of a graph

Description automatically generated

**Conclusion**

1. **Key Observations**:
   * Increasing L improved SQNR and reduced quantization errors.
   * μ-law quantization effectively handled low-amplitude signals.
   * Huffman encoding provided efficient compression, particularly at higher L.
2. **Enhancements**:
   * Increase quantization levels for finer resolution.
   * Employ advanced encoding techniques like arithmetic coding.
   * Use error-correcting codes for noisy channels.

**Code Appendix**

* **Code**:
* % Parameters
* A = 9; % Signal Amplitude [13001578 --> 1 + 8]
* f = 1000; % Frequency in Hz
* duration = 0.01; % Signal duration in seconds
* T\_s = 0.0001; % Sampling interval
* t = 0:T\_s:duration; % Time vector
* L = 32; % Number of quantization levels
* mu = 255; % For μ-law quantization
* p = 0.01; % Error probability in Binary Symmetric Channel
* %% Sampling
* function samples = sample\_signal(A, f, T\_s, duration)
* t = 0:T\_s:duration; % Time vector
* samples = A \* sin(2 \* pi \* f \* t); % Generate sinusoidal signal
* end
* T\_s\_values = [0.001, 0.0005, 0.0001]; % Sampling intervals
* for T\_s = T\_s\_values
* samples = sample\_signal(A, f, T\_s, duration);
* % figure;
* % plot(0:T\_s:duration, samples);
* % title(['Sampled Signal with T\_s = ', num2str(T\_s)]);
* % xlabel('Time (s)');
* % ylabel('Amplitude');
* end
* %% Quantization
* % Uniform
* % function [quantized\_values, levels] = uniform\_quantizer(signal, L, A)
* % levels = linspace(-A, A, L); % Define L quantization levels
* % delta = levels(2) - levels(1); % Step size
* % quantized\_values = round((signal + A) / delta) \* delta - A; % Quantize
* % end
* function [quantized\_values, levels] = uniform\_quantizer(signal, L, A)
* % Define L quantization levels linearly spaced between -A and A
* levels = linspace(-A, A, L);
* % Quantize the signal: find the closest level to each sample
* quantized\_values = zeros(size(signal)); % Initialize output array
* for i = 1:length(signal)
* % Find the closest quantization level
* [~, idx] = min(abs(signal(i) - levels)); % Find closest level index
* quantized\_values(i) = levels(idx); % Assign the quantized value
* end
* end
* % meow :3
* function quantized\_values = mu\_law\_quantizer(signal, L, mu)
* normalized\_signal = signal / max(abs(signal)); % Normalize
* compressed\_signal = sign(normalized\_signal) .\* log(1 + mu \* abs(normalized\_signal)) / log(1 + mu);
* quantized\_signal = round((compressed\_signal + 1) \* (L - 1) / 2); % Uniform quantization
* quantized\_values = 2 \* quantized\_signal / (L - 1) - 1; % Decompress
* quantized\_values = quantized\_values \* max(abs(signal)); % Restore scale
* end
* %% Quantization Error Analysis
* function [mae, var\_error] = quantization\_error\_analysis(signal, quantized\_signal)
* error = signal - quantized\_signal;
* mae = mean(abs(error)); % Mean Absolute Error
* var\_error = var(error); % Variance of Error
* end
* %% Signal-to-Quantization Noise Ratio (SQNR)
* function sqnr = calculate\_sqnr(signal, quantized\_signal)
* signal\_power = mean(signal .^ 2);
* error = signal - quantized\_signal;
* noise\_power = mean((error) .^ 2);
* sqnr = 10 \* log10(signal\_power / noise\_power);
* end
* %% Huffman Encoding
* function [encoded\_signal, dict] = huffman\_encode(signal)
* symbols = unique(signal);
* probabilities = histcounts(signal, [symbols, max(symbols) + 1], 'Normalization', 'probability');
* dict = huffmandict(symbols, probabilities);
* encoded\_signal = huffmanenco(signal, dict);
* end
* %% Huffman Decoding
* function decoded\_signal = huffman\_decode(encoded\_signal, dict)
* decoded\_signal = huffmandeco(encoded\_signal, dict);
* end
* %% Signal Comparison
* % no functions here :)
* %% Compression Metrics
* function compression\_rate = calculate\_compression\_rate(original\_bits, compressed\_bits)
* compression\_rate = (original\_bits - compressed\_bits) / original\_bits;
* end
* %% Error Channel Simulation
* function noisy\_signal = bsc\_channel(encoded\_signal, p)
* noisy\_signal = xor(encoded\_signal, rand(size(encoded\_signal)) < p);
* noisy\_signal = double(noisy\_signal);
* end
* %% Simulation time :3
* % Step 1: Input Signal and Sampling
* input\_signal = sample\_signal(A, f, T\_s, duration);
* % Step 2: Quantization
* [uniform\_quantized, uniform\_levels] = uniform\_quantizer(input\_signal, L, A);
* mu\_quantized = mu\_law\_quantizer(input\_signal, L, mu);
* % Step 3: Quantization Error Analysis
* L\_Values = [4, 8, 16];
* mae\_Values = [0, 0, 0];
* var\_Values = [0, 0, 0];
* i = 1;
* for LValue = L\_Values
* [uniform\_quantized, uniform\_levels] = uniform\_quantizer(input\_signal, LValue, A);
* [mae\_uniform, var\_uniform] = quantization\_error\_analysis(input\_signal, uniform\_quantized);
* % [mae\_uniform, var\_uniform] = quantization\_error\_analysis(input\_signal, mu\_quantized);
* % disp(['Mean Absolute Error (Uniform): ', num2str(mae\_uniform), 'For the value of L:', num2str(LValue)]);
* % disp(['Variance of Error (Uniform): ', num2str(var\_uniform), 'For the value of L:', num2str(LValue)]);
* mae\_Values(1, i) = mae\_uniform;
* var\_Values(1, i)= var\_uniform;
* i = i+1;
* end
* figure;
* plot(mae\_Values, L\_Values);
* title('Mean values vs L values');
* xlabel('Mean Values');
* ylabel('L values');
* figure;
* plot(var\_Values, L\_Values);
* title('Variance values vs L values');
* xlabel('Variance Values');
* ylabel('L values');
* % Step 4: SQNR
* % sqnr\_uniform = calculate\_sqnr(input\_signal, uniform\_quantized);
* % sqnr\_uniform = calculate\_sqnr(input\_signal, mu\_quantized);
* % disp(['SQNR (Uniform): ', num2str(sqnr\_uniform), ' dB']);
* sqnr\_Values = [0, 0, 0];
* i = 1;
* for LValue = L\_Values
* [uniform\_quantized, uniform\_levels] = uniform\_quantizer(input\_signal, LValue, A);
* sqnr\_uniform = calculate\_sqnr(input\_signal, uniform\_quantized);
* % sqnr\_uniform = calculate\_sqnr(input\_signal, mu\_quantized);
* sqnr\_Values(1, i) = sqnr\_uniform;
* i = i+1;
* end
* figure;
* plot(sqnr\_Values, L\_Values);
* title('SQNR values vs L values');
* xlabel('SQNR Values (db)');
* ylabel('L values');
* % Step 5: Encoding (Huffman Encoding)
* [encoded\_signal, huffman\_dict] = huffman\_encode(uniform\_quantized);
* % [encoded\_signal, huffman\_dict] = huffman\_encode(mu\_quantized);
* % Step 6: Simulate a Channel (Noiseless)
* % Huffman decoding to reconstruct the signal
* decoded\_signal = huffman\_decode(encoded\_signal, huffman\_dict);
* % Step 7: Signal Comparison
* % Rescale decoded signal for comparison
* quantization\_step = uniform\_levels(2) - uniform\_levels(1);
* reconstructed\_signal = decoded\_signal \* quantization\_step;
* % Cross-correlation between input and reconstructed signal
* correlation = corrcoef(input\_signal, reconstructed\_signal);
* disp(['Cross-correlation: ', num2str(correlation(1, 2))]);
* % Plotting Input vs Output
* figure;
* plot(t, input\_signal, 'b', 'DisplayName', 'Original Signal');
* hold on;
* plot(t, reconstructed\_signal, 'r--', 'DisplayName', 'Reconstructed Signal');
* legend;
* title('Input Signal vs Reconstructed Signal');
* xlabel('Time (s)');
* ylabel('Amplitude');
* % Step 8: Compression Efficiency and Rate
* original\_bits = length(input\_signal) \* ceil(log2(L));
* compressed\_bits = length(encoded\_signal);
* compression\_rate = calculate\_compression\_rate(original\_bits, compressed\_bits);
* disp(['Compression Rate: ', num2str(compression\_rate \* 100), '%']);
* % Step 9: Error Channel Simulation (Bonus)
* bsc\_signal = bsc\_channel(encoded\_signal, p);
* try
* noisy\_decoded = huffman\_decode(double(bsc\_signal), huffman\_dict);
* noisy\_reconstructed\_signal = noisy\_decoded \* quantization\_step;
* % Plot noisy reconstructed signal
* % Ensure the lengths match for plotting
* n = length(noisy\_reconstructed\_signal);
* min\_length = min(length(input\_signal), n); % Ensure matching lengths
* t\_reconstructed = linspace(0, duration, min\_length); % Adjust time vector length
* % display the cross-correlation in the erroneous :) channel
* % commenting the code because it produces an error when the signals have
* % different lengths
* % normalized\_xcorr = xcorr(input\_signal, noisy\_reconstructed\_signal, 'coeff');
* % disp(['Cross-correlation in the erroneous channel: ', num2str(normalized\_xcorr(1, 2))]);
* figure;
* plot(t\_reconstructed, input\_signal(1:min\_length), 'b', 'DisplayName', 'Original Signal');
* hold on;
* plot(t\_reconstructed, noisy\_reconstructed\_signal(1:min\_length), 'g--', 'DisplayName', 'Noisy Reconstructed Signal');
* legend('Location', 'best')
* title('Input Signal vs Noisy Reconstructed Signal');
* xlabel('Time (s)');
* ylabel('Amplitude');
* catch e
* disp('Error during huffman decoding due to an invalid encoded sequence')
* rethrow(e)
* end
* %% Enhancing Signal Approximation
* % one could increase the number of quantization levels but that would be
* % resource-extensive if implemented irl
* % more elegantly we could resort to more advanced encoding techniques
* % such as arithmetic coding :)