EECE 442 - Communication Systems
American University of Beirut, Fall 2025–2026

September 22, 2025
Due date: October 19, 2025

Course Project

Project Requirements

- Work in a groups of two (except one group of three in case odd).
- Matlab coding is recommended due to its robust communication system toolbox.
- Avoid using data that require special approval for animal or human subjects.
- The final grade will be based on the accuracy and completeness of your work, the efficiency and organization of the code, the quality of the report, and the oral presentation. Ensure that the code is testable, user-friendly, and well-documented. The percentage distribution of the grade on the different components is detailed in the course syllabus.

Description

This course project aims to strengthen your research and programming skills by guiding you through the **design** and **analysis** of digital communication systems. The project is structured, over two parts, into several modules, each covering essential baseband operations and offering hands-on insights into the foundations of digital communications.

By completing the modules, you will construct a full physical-layer communication simulation chain, along with an application for experimentation. As illustrated in Fig. 1, a continuous-time message signal $m(t)$, representing a target application, is first sampled into $m(nT)$ or expressed through a finite set of Fourier series coefficients (assuming communication of time-limited chunks of $m(t)$). The sampled sequence is then quantized into a vector $\mathbf{v}$, mapped into bits $\mathbf{b}$, channel-coded into a codeword $\mathbf{c}$, and modulated into complex symbols $\mathbf{x}$ for transmission. These symbols are affected by the channel $\mathbf{h}$ and additive noise $\mathbf{n}$, producing a received vector $\mathbf{y}$. On the receiver side, the inverse operations are performed: after equalization and detection, an estimate $\hat{\mathbf{x}}$ is obtained, followed by demodulation and decoding into $\hat{\mathbf{c}}$, $\hat{\mathbf{b}}$, and $\hat{\mathbf{v}}$, eventually reconstructing an approximate message signal $\hat{m}(t)$.

Project – Part I

In this first stage of the project, you will focus on building and testing a source encoder and decoder.

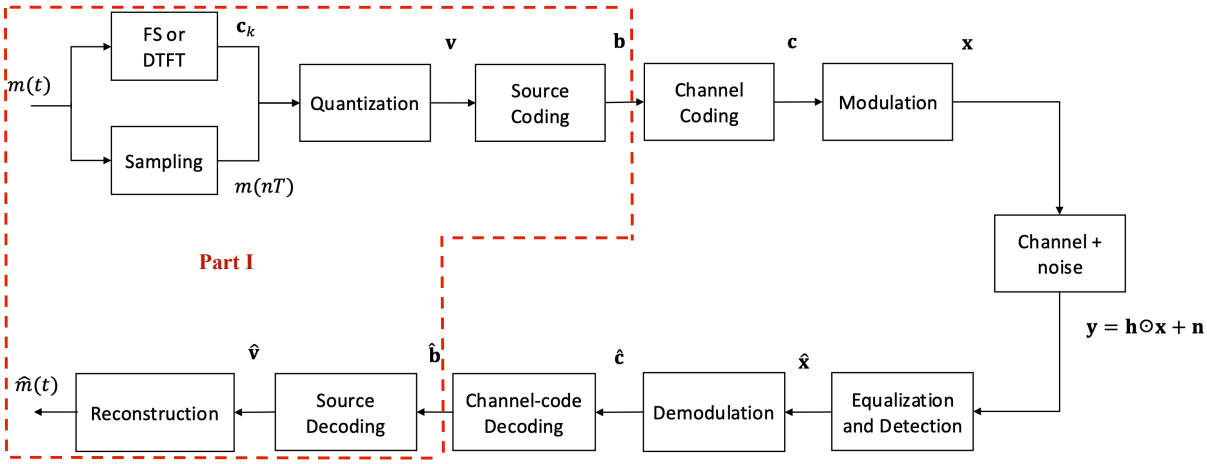

Figure 1: A physical-layer baseband communication system chain.

1 Signal Representation through Sampling

1.1 Generic Sampling

Write a function to sample a given signal $x(t)$ at a rate $f_s = \frac{1}{T_s}$. The Nyquist frequency is f_N. The sampling function should have the following signature:

```
function [t_sample, x_sample] = sample(t, xt, fs)
```

- `t`, `xt`: row vectors representing the time axis and signal values
- `fs`: scalar sampling rate
- `t_sample`, `x_sample`: the sampled time indices and corresponding signal values

Sampling below and above nyquist: Sample $x(t) = \cos(2\pi f_0 t)$ at two different rates: $0.5f_N$ (below Nyquist, leading to aliasing), and $2f_N$ (above Nyquist, proper sampling).

Next, implement a reconstruction function with the signature:

```
function xrcon = reconstruct(t, x_sample, fs)
```

- `t`: row vector of time indices for reconstruction

- `x_sample`: sampled signal
- `fs`: sampling rate

Reconstruction and analysis: Reconstruct the signals from both sampling rates and plot them alongside the original.

- Which reconstruction more closely matches the original signal?
- Explain your observations in light of the sampling theorem.

1.2 Sampling through Fourier Series

Implement a function that generates a **finite Fourier series approximation** of a time-limited signal $x(t)$:

$$\hat{x}(t) = \sum_{k=-n}^{n} c_k e^{j\frac{2\pi}{T}kt}, \quad \text{where} \quad c_k = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\, e^{-j\frac{2\pi}{T}kt}\, dt.$$

Your function must **not** use MATLAB's built-in Fourier series or transform functions. It should have the following signature:

```
function [xhat, ck] = ffs(xt, t, n, T)
```

- n and T are scalars.
- `xt` and `t` are row vectors of $x(t)$ and the corresponding time samples.
- `xhat`: row vector containing $\hat{x}(t)$.
- `ck`: row vector containing the Fourier coefficients c_k.

Fourier series approximation

(a) For a chunked version, $x(t)$, of a message signal, $m(t)$, choose an appropriate value of T and compute the Fourier series approximation $\hat{x}(t)$ for different values of n.

(b) Plot $x(t)$ along with several approximations $\hat{x}(t)$.

(c) Repeat the experiment by varying T while keeping n sufficiently large.

(d) Plot the approximation error energy, $E(n, T)$, as a function of n (for fixed T):

$$E(n, T) = \int_{-\infty}^{\infty} |x(t) - \hat{x}(t)|^2\, dt.$$

(e) Plot $E(n, T)$ as a function of T (for fixed, sufficiently large n). Comment on the tradeoffs observed between n and T.

Choose your message signal carefully to ensure it can be sampled with reasonable complexity; it should also represent an interesting application. By analyzing the frequency content of your message, you can determine suitable input parameters for the sampling functions.

Before moving to the next steps, ensure that you can accurately reconstruct your message signal, assuming an ideal remainder of the communication chain. Additionally, consider segmenting the frequency domain of your message and applying the discrete-time Fourier transform as a third sampling method.

2 Quantization

In this second module, you will implement and experiment with **quantization techniques** applied to the sampled analog sequences obtained from the previous module.

Use the following function to quantize a sampled sequence:

```
function xq = quan(x, thr, lvl)
```

- `x`: input sequence of sampled values (row vector)
- `thr`: vector of quantization thresholds
- `lvl`: vector of quantization levels
- `xq`: quantized sequence (same size as `x`)

2.1 Tasks

Two-level quantization

(a) Apply two-level quantization to the sampled sequence.

(b) Plot both the original and quantized sequences; comment on the distortion observed.

Uniform multi-level quantization

Divide the dynamic range of the input sequence into M equal intervals, and set the quantization levels as the midpoints of each interval.

(c) Experiment with low-rate (e.g., $M = 4$) and high-rate (e.g., $M = 16$ or $M = 32$) uniform quantization.

(d) Report the thresholds and levels used in each case.

(e) Compare the mean square error (MSE) between different quantization rates.

Near-optimal quantization and overall assessment

(f) From the obtained empirical distribution of the input signal, build a Lloyd-Max quantizer.

(g) Integrate your quantization methods into the end-to-end simulation chain.

(h) Compare the reconstructed signals with different quantization methods.

(i) Discuss trade-offs between rate (number of levels), distortion (MSE), and source statistics.

3 Source Coding and Decoding

In this module you will compress the *quantized* symbol streams produced in the previous experiments. You will: (i) start with coding under *unknown* symbol distribution, (ii) learn the *empirical* distribution and design an *optimal* compressor (Huffman), (iii) measure average code lengths and relate them to entropy, and (iv) integrate the source-coding stage into the end-to-end simulation chain to assess overall performance across quantizers and coding strategies.

3.1 Inputs and Setup

- Quantized sequences from each quantization method/rate (uniform, Lloyd–Max, etc.). State alphabet $\mathcal{A}$ and quantizer parameters.
- Keep the same seeds between methods when comparing.

3.2 Tasks

Baseline Huffman Coding (symbol-wise)

(a) **Unknown distribution pass:** code the stream with an *initial* generic code (e.g., fixed-length $\lceil \log_2 |\mathcal{A}| \rceil$ bits/symbol). This establishes a no-model baseline.

(b) **Empirical modeling:** estimate $p(a) = \frac{\#\{a\}}{N}$ for $a \in \mathcal{A}$ from the full sequence (or a held-out prefix). Report entropy

$$H(\mathsf{A}) = -\sum_{a \in \mathcal{A}} p(a) \log_2 p(a).$$

(c) **Optimal instantaneous code:** build a *Huffman* code for $p(\cdot)$; encode and decode the stream. Verify lossless reconstruction.

(d) **Analysis:** compare $\bar{\ell} = \sum_a p(a)\,\ell(a)$ vs. $H(\mathsf{A})$ and vs. fixed-length. Explain why some quantizers/rates yield shorter codes (e.g., skewed histograms, dead-zones) while others do not.

Block Source Coding (concatenated symbols)

(e) Form length-k blocks $A^k = (A_1, \ldots, A_k)$; estimate $p(a^k)$ from data (choose modest k, e.g., $k \in \{2, 3, 4\}$ to avoid sparsity).

(f) **Encode twice:** (i) fixed-length with $\lceil \log_2 |\mathcal{A}|^k \rceil = k \lceil \log_2 |\mathcal{A}| \rceil$ bits/block; (ii) *Huffman* on $p(a^k)$ (variable length).

(g) **Compare efficiency:** report $\bar{\ell}_k$ (bits per *symbol*) and relate to the empirical *block* entropy rate H_k/k, where $H_k = -\sum_{a^k} p(a^k) \log_2 p(a^k)$.

(h) **Discussion:** comment on gains from short-range dependencies (if any) introduced by quantizer memory or signal statistics.

Integration into the Full Chain (Uncoded Channel): Insert the lossless compressor *after* quantization and the decompressor *before* de-quantization (if applicable). For each quantizer:

(i) Measure: total bits, throughput (bits/symbol), and latency (if you buffer to learn p).

(j) Compare the end-to-end performance between quantizers and coding strategies with identical SNR and block length.

3.3 (Bonus) Extension: Dictionary Coding via Lempel–Ziv

Yes—include a *model-free* compressor such as **LZ77/LZ78/LZW**:

(k) Run LZ on the same sequences *without* prior modeling; report achieved bits/symbol.

(ℓ) Compare to Huffman(*symbol*) and Huffman(*block*) at the same conditions. Discuss scenarios where LZ performs better (captures longer patterns) vs. worse (small blocks, highly uniform symbols).