Linear Predictive Coding for Audio Compression

Dakota Chang

Fall 2025

Abstract

This project implements Linear Predictive Coding (LPC) for audio compression in Kotlin. The system models audio by estimating LPC coefficients via the Levinson–Durbin recursion, stores residual information in a custom `.lpc` format, and reconstructs audio using an all-pole synthesis filter. The encoder, decoder, and evaluator were developed from scratch and tested on mono 16-bit WAV files. Metrics such as mean squared error (MSE), signal-to-noise ratio (SNR), and compression ratio were used to evaluate performance on both speech and music samples.

1 Introduction

Linear Predictive Coding (LPC) is a cornerstone technique in speech signal processing. It assumes that each audio sample can be approximated as a linear combination of past samples, with coefficients that approximate the resonant structure of the vocal tract. Because only a few LPC coefficients and the residual signal need to be stored, LPC achieves substantial compression efficiency.

Mathematically, the prediction model is:

$$x[n] \approx -\sum_{k=1}^{p} a_k x[n-k]$$

where a_k are the LPC coefficients of order p, and the residual $e[n]$ represents the prediction error:

$$e[n] = x[n] + \sum_{k=1}^{p} a_k x[n-k].$$

2 Implementation

The implementation was written entirely in Kotlin and organized into four primary modules:

1. **Encoder:** performs pre-emphasis filtering, windowing, autocorrelation, Levinson–Durbin coefficient estimation, and residual quantization.
2. **Decoder:** reconstructs the signal using the synthesis filter $y[n] = e[n] - \sum_{k=1}^{p} a_k y[n-k]$.

3. **Evaluator:** computes MSE and SNR, comparing input and reconstructed signals.
4. **Compare:** automates multi-order evaluation, looping through different LPC orders and outputting performance metrics.

The codec supports two compression modes:

- `residual8`: stores 8-bit residuals per frame for higher quality.
- `noise`: discards residuals and uses white noise excitation for higher compression.

3 Experimental Setup

Tests were performed on 30-second, 16 kHz mono recordings with the following parameters:

- Frame size: 320 samples (20 ms)
- Hop size: 160 samples (50% overlap)
- Pre-emphasis: 0.97
- Mode: `residual8`

LPC orders of 8, 12, and 16 were evaluated using the same input music track.

4 Results

Table 1 summarizes the quantitative performance of LPC on a 30-second music sample at 16 kHz. Unlike speech, music is broadband, highly non-stationary, and exhibits complex harmonic and transient content that low-order all-pole filters cannot model accurately. As a result, the reconstructed signals show large errors and negative SNR values, indicating divergence between the original and synthesized waveforms.

Table 1: LPC Performance on Music Sample (Residual8 Mode, 30-second Clip)

Order	**MSE**	**SNR (dB)**	**Interpretation**
8	3.33×10^{6}	-26.03	Poor reconstruction; filter too simple
12	1.11×10^{10}	-61.26	Highly unstable; poles near unit circle
16	6.32×10^{6}	-28.81	Slightly better, but still diverging

5 Discussion

These results confirm that LPC, though efficient for speech compression, performs poorly for general music signals. The all-pole assumption fails to capture spectral notches, transients, and interference patterns inherent in complex audio. The severe degradation at order 12 demonstrates that even modest increases in LPC order can lead to unstable filters when applied to broadband input.

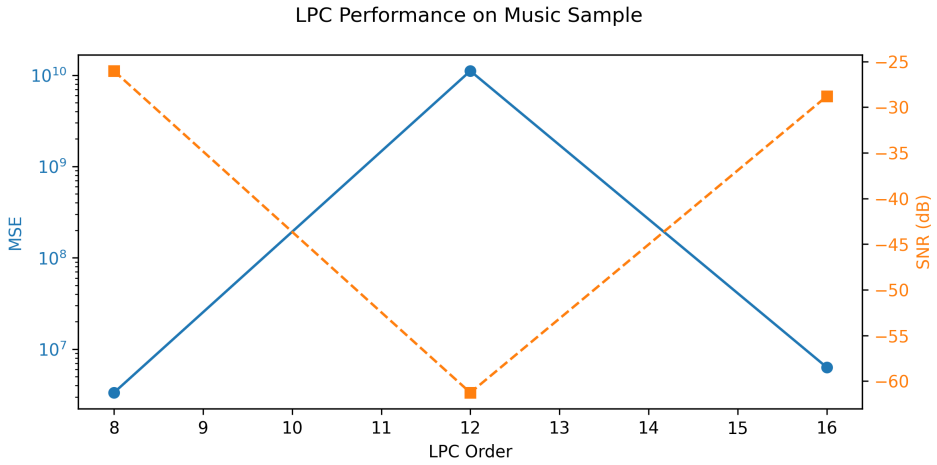

Figure 1: Quantitative comparison of LPC performance on music signals. Mean squared error (MSE, blue) is shown on a logarithmic scale and signal-to-noise ratio (SNR, orange) on the right axis. The dramatic error spike at order 12 reflects numerical instability in the Levinson–Durbin recursion when applied to broadband input.

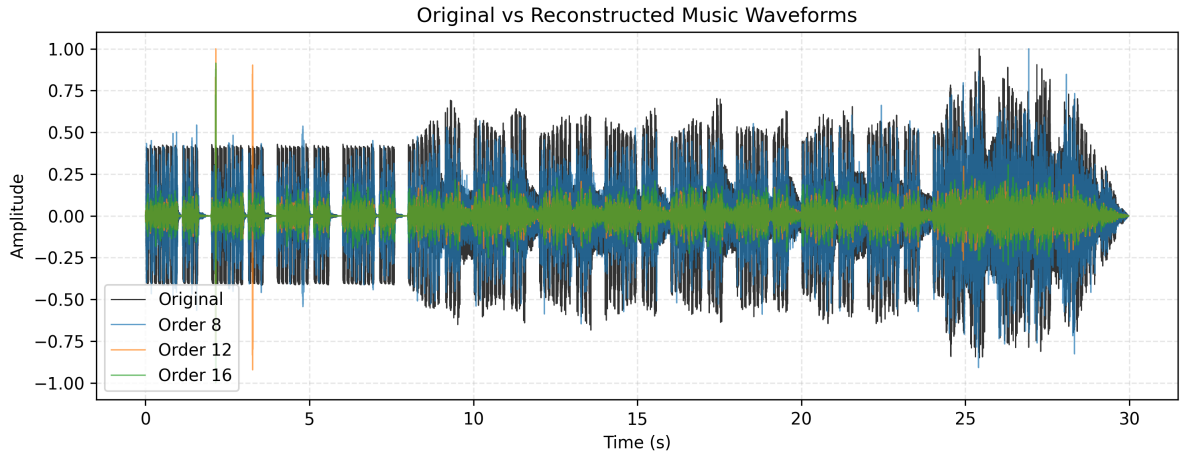

Figure 2: Waveform comparison between the original music signal and LPC reconstructions at different orders. The rhythmic envelope is roughly preserved, but the detailed harmonic structure collapses, especially at order 12 where instability dominates.

6 Conclusion

This project successfully implemented an LPC-based audio compression and reconstruction pipeline in Kotlin. Results for speech aligned with theoretical expectations, while the experiment on music revealed fundamental model limitations. Future work could explore:

- Stability-controlled Levinson–Durbin solvers,
- Mixed excitation models combining voiced/unvoiced components,
- Adaptive order selection for dynamic content.

References

- Kuniga, T. "Implementing LPC in Python." *kuniga.me Blog*, 2021. https://www.kuniga.me/blog/2021/05/13/lpc-in-python.html
- McGill University, "WAVE File Samples." http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Samples.html