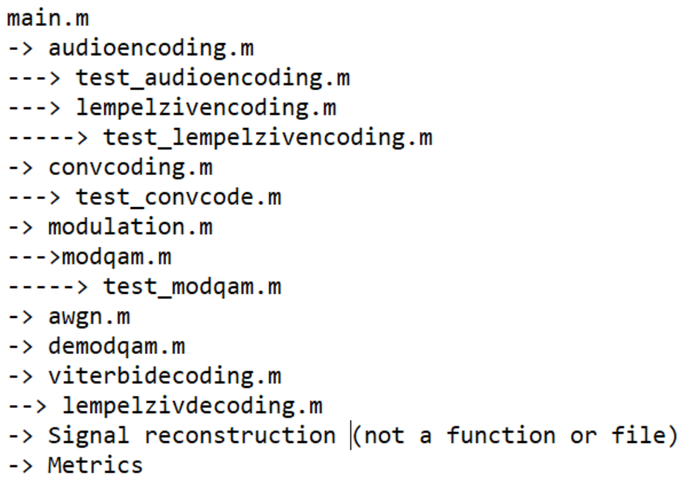
Audio Communication Systems

Contents

The goal of this miniproject is to incorporate the details of an end-end communication systems via simulations. An audio file is read into the program that passes through the transmitter system. In here, it undergoes source coding using Lempel-Ziv and channel coding using convolutional code. From here, it is modulated using quadrature amplitude modulation then passes through the channel which is additive white Gaussian noise. The signal then goes through the receiver system where it begins to decode the noisy signal into bit streams that passes through Viterbi decoding and Lempel-Ziv decoding to eventually be reconstructed at the recipient's end. The goal is to observe the effects on Mean Square Error (MSE), BER, Compression Ratio, and Qualitative analysis.

The audio file compressed and transmitted in this communication system is of a well-known piano piece of Fur Elise recorded on Youtube [2]. The audio was read as sampled quantized double values, 64 bits per sample. Recorded at 44kHz, only few random seconds snippets were utilized to observe and analyze the results due to computational complexity. Majority of the programs were not optimally implemented for speed based on data structure, accessibility etc. They operate on a functionality standpoint. Each iteration of the program takes roughly 2 hours to process, so it may be best to view the figures and tables of this report.

The overarching program is ran using main.m. An outline of the hierarchy is shown here:



```
main.m
-> audioencoding.m
---> test_audioencoding.m
---> lempelzivencoding.m
-----> test_lempelzivencoding.m
-> convcoding.m
---> test_convcode.m
-> modulation.m
--->modqam.m
-----> test_modqam.m
-> awgn.m
-> demodqam.m
-> viterbidecoding.m
--> lempelzivdecoding.m
-> Signal reconstruction (not a function or file)
-> Metrics
```

Figure 0: Program Hierarchy

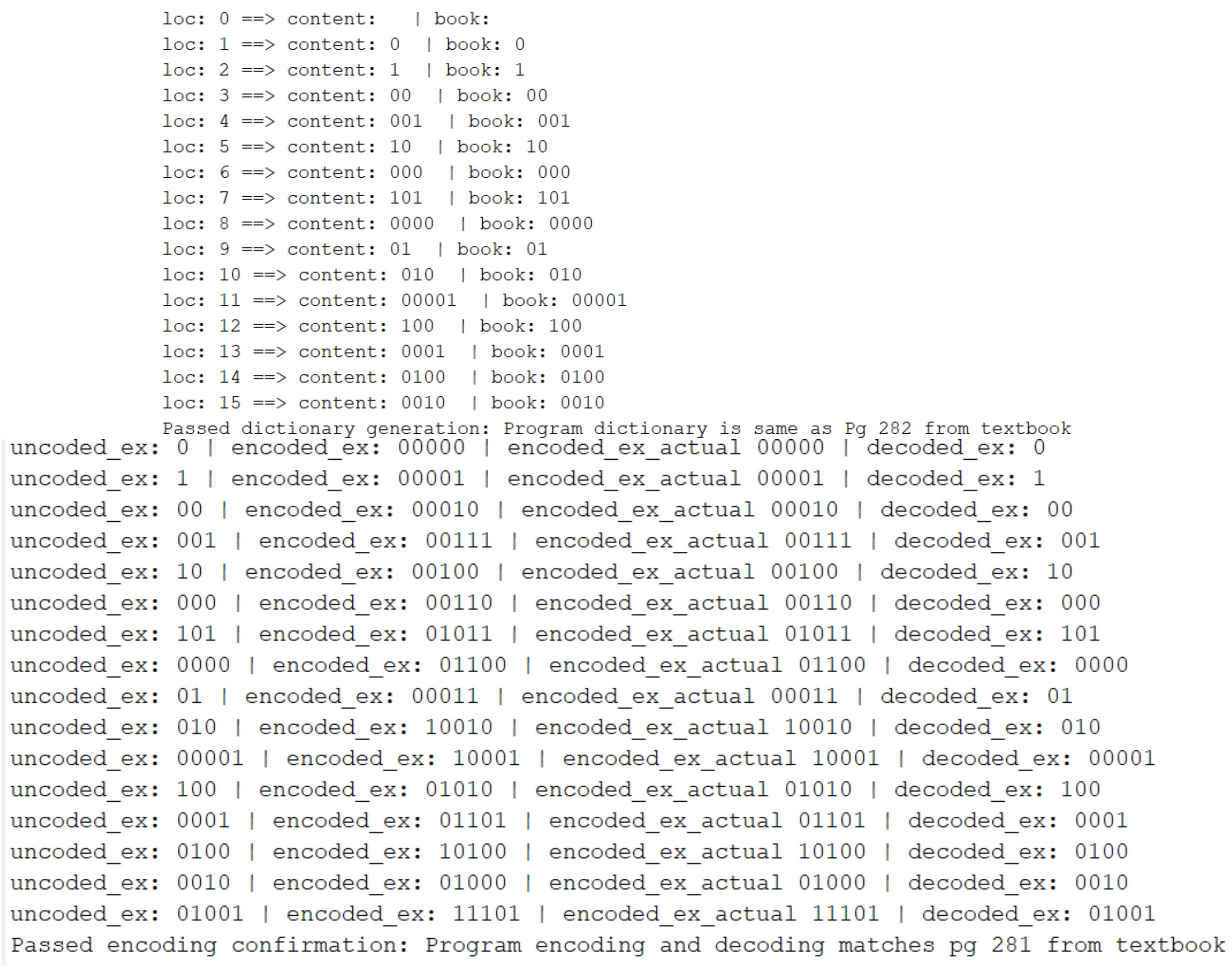
1 Lempel-Ziv Encoding

The lossless compression scheme utilized is the Lempel-Ziv encoding scheme that does not require source probabilities (Pg 280-282). This maps variable-length codes to a fixed-length code. It accomplishes this by populating a dictionary of 2^b entries for *b* bits with new unique sequences as they appear in the data. For example, for the sequence of '0100001100001010000010100000110000010100001001001', the first few entries are '0','1','00','001' etc. After filling up all of the dictionary entry space, the data appends continues to grab sequential bits until the first largest sequence of data not in the dictionary is obtained. Then the last bit becomes the codeword and the previous bits are replaced by a series of *b* bits for the *location* in the dictionary. If the sequence to encode is '01010' with the fifth dictionary entry of '0101' of a 3-bit compression, then the compression would be 101 0. The 0th entry is reserved for the empty character set. In the compression mapping to 2^b dictionary entries results in each sequence being mapped to $b + 1$ bits to include the codeword as well.

Table 1.1: LZ Bits Per Phrase and Compression Rate (Lower is better)

Bits Per Phrase	Code Rate
16	17.77%
12	38.12%
8	45.23%
4	91.63%

The compression rate, defined as the ratio between number of bits after compression and before compression, was found to be roughly 43.12% for a 12-bit mappings. As shown from table, an increase in bits per phrase decreases the code rate. Larger bits per phrase has more space to map longer sequences to only 16 bits.



```
loc: 0 ==> content:   | book:
loc: 1 ==> content: 0  | book: 0
loc: 2 ==> content: 1  | book: 1
loc: 3 ==> content: 00  | book: 00
loc: 4 ==> content: 001  | book: 001
loc: 5 ==> content: 10  | book: 10
loc: 6 ==> content: 000  | book: 000
loc: 7 ==> content: 101  | book: 101
loc: 8 ==> content: 0000  | book: 0000
loc: 9 ==> content: 01  | book: 01
loc: 10 ==> content: 010  | book: 010
loc: 11 ==> content: 00001  | book: 00001
loc: 12 ==> content: 100  | book: 100
loc: 13 ==> content: 0001  | book: 0001
loc: 14 ==> content: 0100  | book: 0100
loc: 15 ==> content: 0010  | book: 0010
Passed dictionary generation: Program dictionary is same as Pg 282 from textbook
```

```
uncoded_ex: 0 | encoded_ex: 00000 | encoded_ex_actual 00000 | decoded_ex: 0
uncoded_ex: 1 | encoded_ex: 00001 | encoded_ex_actual 00001 | decoded_ex: 1
uncoded_ex: 00 | encoded_ex: 00010 | encoded_ex_actual 00010 | decoded_ex: 00
uncoded_ex: 001 | encoded_ex: 00111 | encoded_ex_actual 00111 | decoded_ex: 001
uncoded_ex: 10 | encoded_ex: 00100 | encoded_ex_actual 00100 | decoded_ex: 10
uncoded_ex: 000 | encoded_ex: 00110 | encoded_ex_actual 00110 | decoded_ex: 000
uncoded_ex: 101 | encoded_ex: 01011 | encoded_ex_actual 01011 | decoded_ex: 101
uncoded_ex: 0000 | encoded_ex: 01100 | encoded_ex_actual 01100 | decoded_ex: 0000
uncoded_ex: 01 | encoded_ex: 00011 | encoded_ex_actual 00011 | decoded_ex: 01
uncoded_ex: 010 | encoded_ex: 10010 | encoded_ex_actual 10010 | decoded_ex: 010
uncoded_ex: 00001 | encoded_ex: 10001 | encoded_ex_actual 10001 | decoded_ex: 00001
uncoded_ex: 100 | encoded_ex: 01010 | encoded_ex_actual 01010 | decoded_ex: 100
uncoded_ex: 0001 | encoded_ex: 01101 | encoded_ex_actual 01101 | decoded_ex: 0001
uncoded_ex: 0100 | encoded_ex: 10100 | encoded_ex_actual 10100 | decoded_ex: 0100
uncoded_ex: 0010 | encoded_ex: 01000 | encoded_ex_actual 01000 | decoded_ex: 0010
uncoded_ex: 01001 | encoded_ex: 11101 | encoded_ex_actual 11101 | decoded_ex: 01001
Passed encoding confirmation: Program encoding and decoding matches pg 281 from textbook
```

Figure 1.1: Test cases from [1] confirmed working for LZ encoding

LZ Encoding Program

Within, the lempelzivencoding.m function is utilized that utilizes the uncoded bitstream, number of bits, and bits per phrase to encode the data producing an encoded bit stream and the associate encoding dictionary.

The testing work assessed based on known examples from the book. The book utilized the aforementioned sequence to generate a predetermined dictionary for 4-bits that was evaluated.

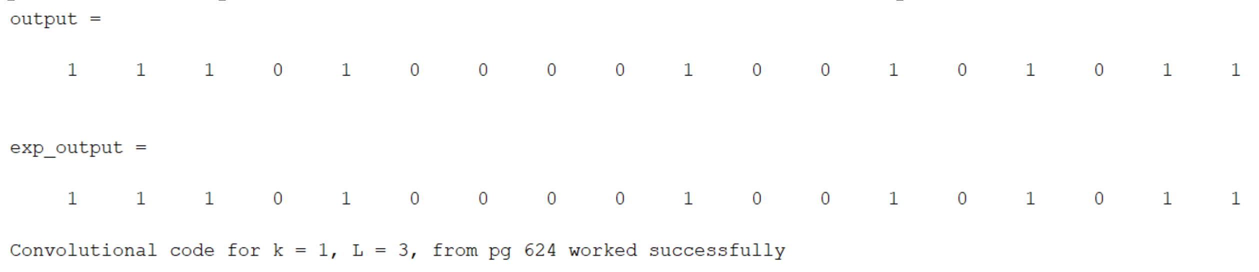
2 Convolutional Codes

2.1 Coding

As opposed to other coding scheme such as Haming Codes or Reed Soloman codes, Convolutional codes only sends parity bits and no message bit. At each stage, k new bits are fed into a shift register comprising of L blocks of k length. The constraint length, block length, the number of linear combinations, and generator polynomial matrix $L, k, n, and\ G$ are utilized to produce the encoded bits. For simplicity, $k = 1$ in our situation to allow for a generator polynomial matrix. The original signal is padded with $k(L-1)$ zeroes to flush the system at the end. The specific parameters is free to be played with, but a common value was utilized in the audio communications systems.

2.1.1 Coding Test

For an input of '1101011' on a k=1, L=3, n=2 system with g1 = [1 0 1], g2 = [1 1 1] from Pg 624 produces the output below that is confirmed similar outlined in Example 9.7.1



```
output =

     1     1     1     0     1     0     0     0     0     1     0     0     1     0     1     0     1     1


exp_output =

     1     1     1     0     1     0     0     0     0     1     0     0     1     0     1     0     1     1

Convolutional code for k = 1, L = 3, from pg 624 worked successfully
```

Figure 2.1.1.1: Convolution Code example from [1, Pg 624]

2.2 Decoding

The decoding method of choice is the Viterbi Decoding Algorithm: “Optimum Decoding of Convolutional code” [1]. A hard decision is made that utilized the Hamming Distance to pick the correct outcome compared to the soft decision that would implement Euclidean distance.
To accomplish this via software, three tables were initialized and populated 1) Next State Table, given Current State and input bit 2) Output Table, given Current State and input bit, and 3) Input Table, given Current State and Next State that has values of -1 when it’s not applicable. The next state shifts the current bits to the right and adds the input bit to the front. The output table is dependent on the generator polynomials.

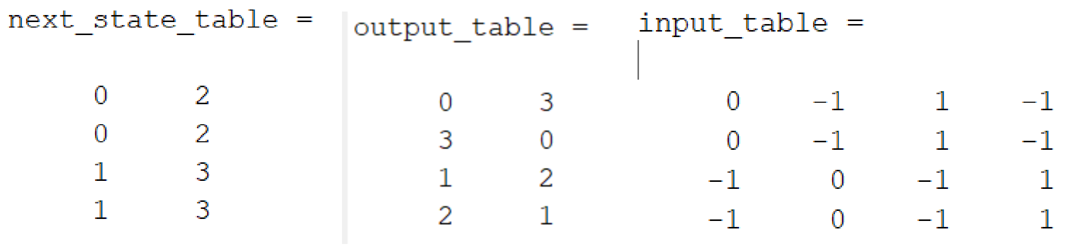
The algorithm for Viterbi Decoding utilizes dynamic programming to gather minimum Hamming Distance at a specific stage number and final state. For each n bits, the hamming distance at each possible state is compared. A table for the path metric was created that represent minimum Hamming distance (with initial state of 0) and the minimum Hamming distance for the full codeword is the minimum of the states at the final stage. The algorithm gets the Hamming

Distance between the output and the received codeword of each transition between states. The two possible previous states are looked up in the Next State Table. The previous states are then used in the output table to get the output. The branch Hamming Distance is calculated between the output and received codeword, then the minimum of the branch Hamming distance + past Hamming distance at the previous stage is utilized. For example, if we are trying to find the minimum of state ‘10’ at stage i, the phase metric is shown below and is done iteratively until the full table is filled. The optimal path is one of smallest Hamming distance in the hard decision and starts at the end column of the table. The minimum Hamming distance is chosen which means the path has the lowest BER. First, one starts at the end column to find the end state. Then the possible states that could lead to the end state is obtained and their minimum Hamming distance is compared. The lower one is utilized all the way until the end. The test below shows it in action.

$$PM['10', i] = \min\{(PM['00', i-1] + Branch['00' \rightarrow '10']), (PM['01', i-1 + Branch['01 \rightarrow ; 10'])\}$$

The generator function utilized is from the example in the book. The path memory truncation method was not implemented, so the Viterbi decoding takes a long time at the moment. Future work seeks to truncate it and use puncture tables for increased code rate and deciphering rate.

2.1.2 Decoding Test



```
next_state_table =

     0     2
     0     2
     1     3
     1     3
```

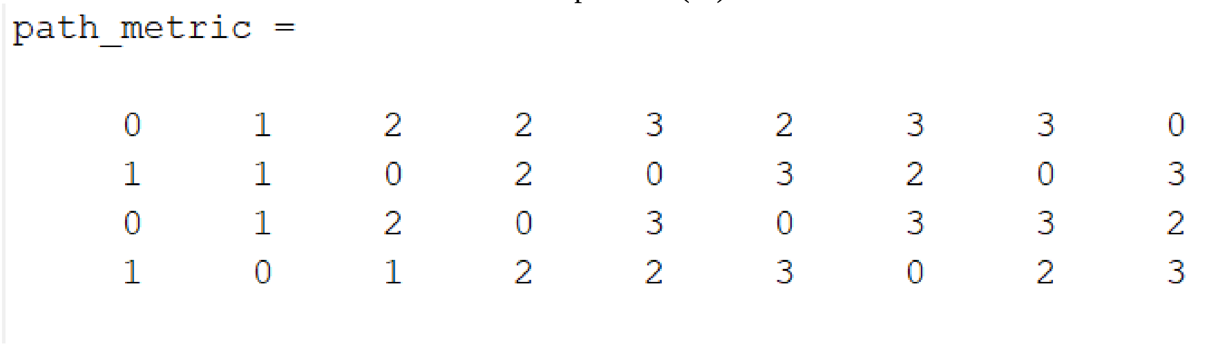
```
output_table =

     0     3
     3     0
     1     2
     2     1
```

```
input_table =

     0    -1     1    -1
     0    -1     1    -1
    -1     0    -1     1
    -1     0    -1     1
```

Figure 2.1.2.1: Three tables shown: 1) Next state(NS) table (given CS and input bit) 2) Output Table (given CS and input bit) and 3) Input Table (given CS and NS) returning the input bit or not possible(-1)



```
path_metric =

     0     1     2     2     3     2     3     3     0
     1     1     0     2     0     3     2     0     3
     0     1     2     0     3     0     3     3     2
     1     0     1     2     2     3     0     2     3
```

```
decoded_input =

     1     1     0     1     0     1     1     0     0


state_sequence =

     0     2     3     1     2     1     2     3     1     0
```

Figure 2.1.2.2: Dynamic Programming of Hamming distance Path Metric at each state within each stage that is used to reverse the state sequence history and decoded input

3 Quadrature-Amplitude Modulation

Quadrature-Amplitude Modulation (QAM) [1, Pg 357-360] is a bandpass modulation that encodes the data in both the amplitude and phase. As compared to PSK and the likes, each symbol does not have equal energy, but it allows a higher bit rate at a specified power.

3.1 Modulation

QAM has a constellation diagram shown in Figure 3.1.1 where each point is always at least a certain distance away from any other point so it is shaped as a square. Treating the in-phase(I) and quadrature(Q) component as a complex number, the magnitude and phase for ith codeword, is A_i and ϕ_I. The bandpass signal is modulated into the signal $x(t)$ given an amplitude A, window function $g_T(t)$, carrier frequency f_c, and carrier phase θ. Also shown below, the amplitude and phase can be written as a sum of two oscillators shifted by 90 degrees as separately amplified cosine and sine waves.

$$x(t) = A\left(A_j\, \mathrm{g_T(t)} \cos\left(2\pi f_c t + \theta - \phi_j\right)\right) = A g_T(t)(A_{iI} \cos(2\pi f_c t + \theta) + A_{iQ} \sin(2\pi f_c t + \theta)$$

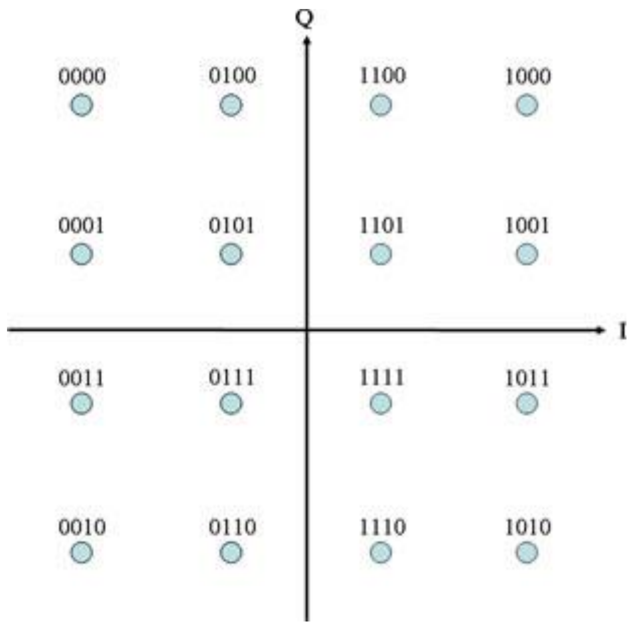


Figure 3.1.1: 16-QAM example.

A 16-QAM modulation scheme was chosen that transmits 4 bits with each symbol. The carrier phase was set to 0 in this project and synchronicity was assumed for simplicity. The window function is a pure window with no regard for phase discontinuity in the transmitted signal. For a symbol period of T_b, 4 bit are transmitted for a bit rate of $\frac{4}{T_b}$. Four codewords were transmitted and plotted for example.

In the program, the function modulation.m that takes the bitstream, sample frequency, modulation scheme, and associate parameters is used that runs the modqam, currently only setup for 16-QAM that takes in the before variables plus additional parameters: bitstream, amplitude scaling, M-ary setup, carrier frequency, symbol period, and sampling frequency.

The test code verified successful QAM modulation, how two symbols appear, showing via simulation the amplitude and phase vs in-phase and quadrature results, and showing the output of correlator to an example, and finally generating BER and SNR curves for QAM alone.

3.2 Demodulation

The demodulation assumes an accurate carrier phase estimation or locking. The analysis with a phase offset produces detrimental effects on the SNR which is understood for observations but will not be implemented in the system. Given the synchronicity assumption, windows of the received signal is correlated with the in-phase and quadrature basis functions, $\psi(t)$ and scaled appropriately to the below equation from continuous to discrete domain and correlationr_i. The energy of the gate function that has value of 1 from$[0, T_b] = E_g$. The output of r_i is an estimate for I and Q constellation point respectively. With the below equation, in the absence of noise, the constellation had accurate correlate value. The respective I and Q point are then compared with distance to all constellations and chosen whichever Euclidean distance is smallest.

$$\psi_1(t) = 2\cos(2\pi f_c t)$$
$$\psi_2(t) = 2\cos(2\pi f_c t)$$

$$r_i = \int_0^t \big(\psi_i(t) y(t)\big) dt \approx \sum_{n=0}^{N} \frac{1}{T_b} \frac{1}{fs} \psi_i[n] \frac{1}{A} y[n]$$

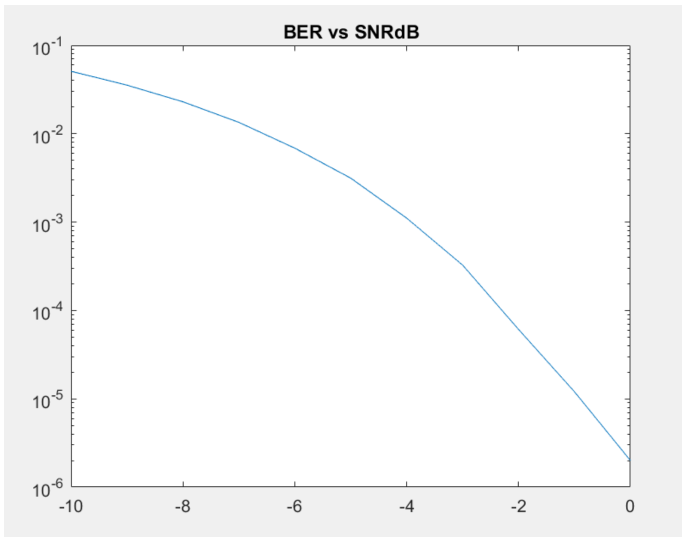


Figure 3.1.2: BER vs SNR for QAM (Up to 10^-6 accuracy, higher SNR did not have a bit flip at this resolution)

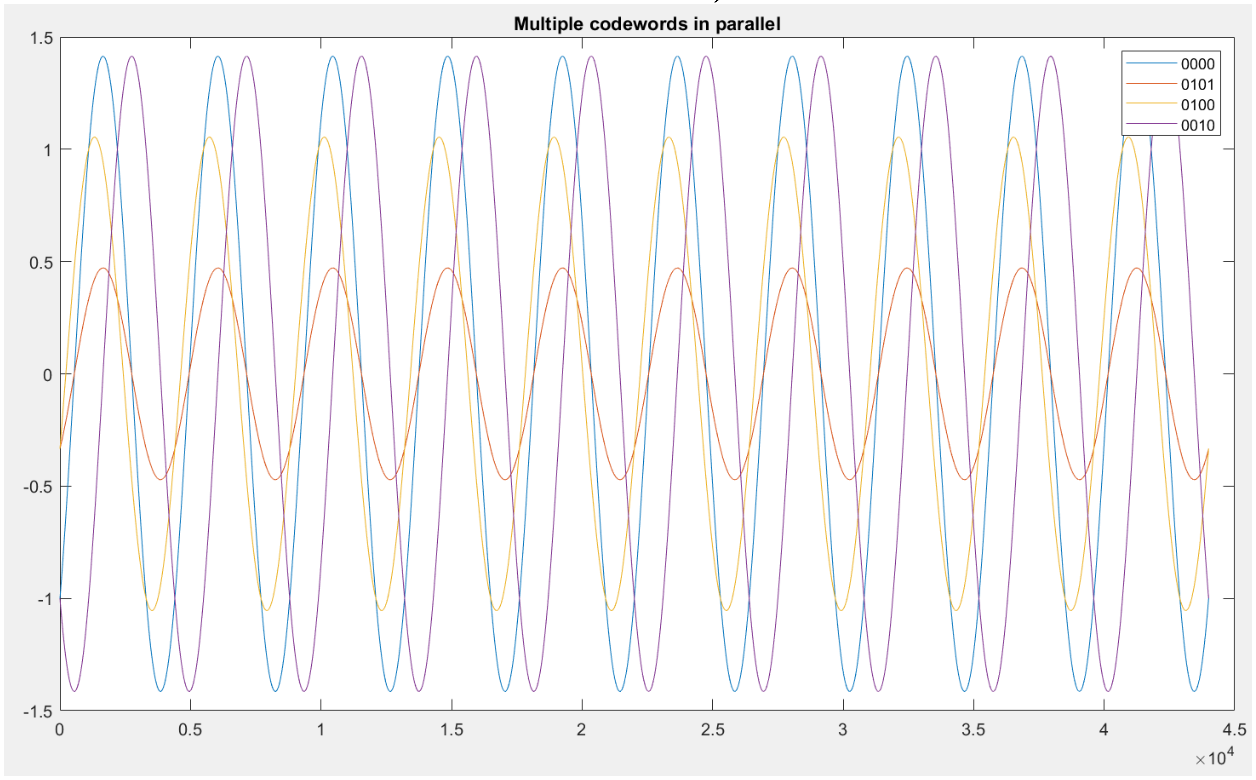


Figure 3.1.3: Time-domain output of QAM overlapping for different codewords for comparison. 0101 and 0000 have same phase but different amplitude for example.

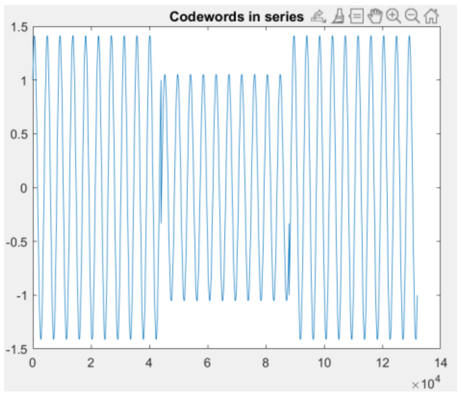


Figure 3.1.4: Codewords in series showing phase discontinuities at window edge due to perfect gate windowing.

```
Expected -- A_I: -1.000000, A_Q: 1.000000, B_I: -0.333333, B_Q: 0.333333
Calculated -- A_I: -1.000045, A_Q: 1.000000, B_I: -0.333348, B_Q: 0.333333
```

Figure 3.1.5: Few I and Q components demodulated for comparison of result.

4. Channel

The channel is a simple additive white Gaussian noise(AWGN) channel for explicit analytical solution. A specified SNR in decibel determined the additive noise while keeping the transmitted signal constant.

5. Results

The table below shows the result of the project. The program took roughly 3 hours to run for each simulation which did not make it feasible to do a system analysis as a whole, but instead analyze each individual portion. In this time, simulation were ran at three SNR values to observe general effect on results. In terms of the effectiveness of QAM and Convolutional Codes, low bit error rate were observed from -10 to 10 decibels. At -10 dB, the convolutional code created a BER of roughly 2.89*10^-5 suggesting that the convolutional code provided roughly a 5 dB coding gain to the system with these parameters. Interesting to note, is the effect of uncompressing given bit error. Despite very low bit error in the communication channel, because the Lempel-ziv uses a dictionary with different mapping of 12-bit encoding to variable-sized uncoded bit streams. The few bit error in the Viterbi decoding bloomed into a 28.86% Lempel-Ziv decoding error, or reconstruction bit error. For example, if the actual dictionary location is at 400_8, but the first bit is flipped, the decoded dictionary location is 000_8. Mentioned in section 1, The first location, 000_8 maps to only one bit of ‘0’ or ‘1’, whichever comes first. On the other hand, the location of 400_8 is much further in the dictionary when longer bit sequences are observed (such as an 11 bits). In this example, 10 bits are thrown away! For bits for dictionary location of similar length contents, many of the bits are not the same which further adds to the error. The -10 decibel performance highlights this behavior and the danger of the Lempel-Ziv source coding under noisy condition and no retransmission in packets. And accordingly, incorrect bit caused a few samples to be very large for the double value and eventually produce an error for MSE.

SNR dB	MSE	QAM BER	Post-Coding BER	Reconstruction bit Error	Compression Ratio	Notes
10	0	0	0	0	0.3544	Normal
-5	0.035	2.38*10^-5	0	0	0.3201	Normal
-10	INF	8.8*10^-3	2.89*10^-5	0.2886	0.3597	Incomprehensible

Table 5.1: Performance Metric computed at different SNR with LZ Bits = 12

6. Conclusion and Future Work

In conclusion, this project gave the opportunity to not only learn the theoretical material in communication system engineering, but also apply, via software, and observe various effects on the result. The Lempel-Ziv encoding scheme was popular in the late 1900's widely spread amongst computers, during the time of the textbook's release. It was used in many sources from that time such GIF, PDF, and more, but has been beaten by other compression techniques like in gzip that produced better ratios. The compressed bit stream is then encoded for stronger strength and separation in convolutional coding. Afterwards, QAM is applied that is widely used in networking devices like WiFi, cable television, and more. 5G emergence implements Orthogonal Frequency Division Multiplexing techniques that has roots from QAM and Frequency Division Multiplexing [5].

Future work can further gather data and compare them to theoretical work for things such as BER vs SNR, SNR gain due to convolutional codes, higher resolution (like 10^-8), that could not be assessed due to time-constraints. Additionally, this project can include implementation of alternative methodologies. This can include another audio encoding that transmits parameters such as Analysis-Synthesis Technique or using Lempel-Ziv with a time-varying dictionary, or an alternative channel coding with Turbo Codes, or the newer Polar codes. For Modulation, implementing OFDM would be interesting to observe. On a similar vein, as QAM modulate a sinusoids parameters of amplitude and phase, OFDM modulating amplitude, phase, and frequency. In the detection and estimation class had works from Stoica found an algorithm to develop an estimator for Cramer-Rao lower bound on these sinusoidal parameters. Especially for the carrier phase issue mentioned, one such approach could be applied as an alternative to synchronization and phase lock loops.

A. References

[1] Communication Systems Engineering, 2nd edition, by Proakis
[2] Fur Elise Recording: https://www.youtube.com/watch?v=_mVW8tgGY_w
[3] QAM image: http://ecelabs.njit.edu/ece489v2/lab5.php
[4] Lempel-Ziv Uses: https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch#Uses
[5] 5G OFDM: https://www.5gtechnologyworld.com/the-basics-of-5gs-modulation-ofdm/
[6] Viterbi Decoding of Convolution Codes: http://web.mit.edu/6.02/www/f2011/handouts/8.pdf