**Digital Communications** **Laboratory Report**

Bit Errors and

Parity Checking

by JIE TIAN 2802461t

Rankine 329

February 21, 2024

1. **Importing libraries and Obtaining Digital Data**

Before we start parity checking, we need to import the libraries that we will need to use in this project, they can provide functions such as reading pictures by greyscale, some specific mathematical algorithms, plotting, modulation and demodulation, simulation of signal channels, and so on, here is the corresponding code.

*import numpy as np*

*from PIL import Image*

*from matplotlib import pyplot as plt*

*import scipy*

*import komm*

A number of 8-bit depth grayscale images of various sizes have been provided for use in this laboratory project. We can read the greyscale image and convert it to the corresponding binary file with the following code.



*tx\_im = Image.open("/Users/george/Project/DC Project/Bit Errors and Parity Checking/DC4\_150x100.pgm")*

*Npixels = tx\_im.size[1]\*tx\_im.size[0]*

*plt.figure()*

*plt.imshow(np.array(tx\_im),cmap="gray",vmin=0,vmax=255)*

*plt.show()*

*tx\_bin = np.unpackbits(np.array(tx\_im))*

1. **Noisy Channel Simulation**

The Komm library provides functions for PSK modulation and demodulation and for simulating an additive white Gaussian noise source, we will use the following code to implement BPSK coding, propagate the signal in a noisy channel, and BPSK decoding. The modulated psk waveform is by default unit average power. The scalar snr specifies the signal-to-noise ratio for the channel.

*psk = komm.PSKModulation(2)*

*awgn = komm.AWGNChannel(snr=10 \*\* (6 / 10))*

*tx\_data = psk.modulate(tx\_bin)*

*rx\_data = awgn(tx\_data)*

*rx\_bin = psk.demodulate(rx\_data)*

Use the following code to show the values of Npixels, and the data types and sizes of tx\_bin, tx\_data, rx\_data and rx\_bin.

*print(Npixels, tx\_bin.size, tx\_data.size, rx\_data.size, rx\_bin.size)*

*print(np.array(tx\_im).dtype, tx\_bin.dtype, tx\_data.dtype, rx\_data.dtype, rx\_bin.dtype)*

Below are the results

*15000 120000 120000 120000 120000*

*uint8 uint8 complex128 complex128 int64*

1. **Measuring Bit Error Ratio**

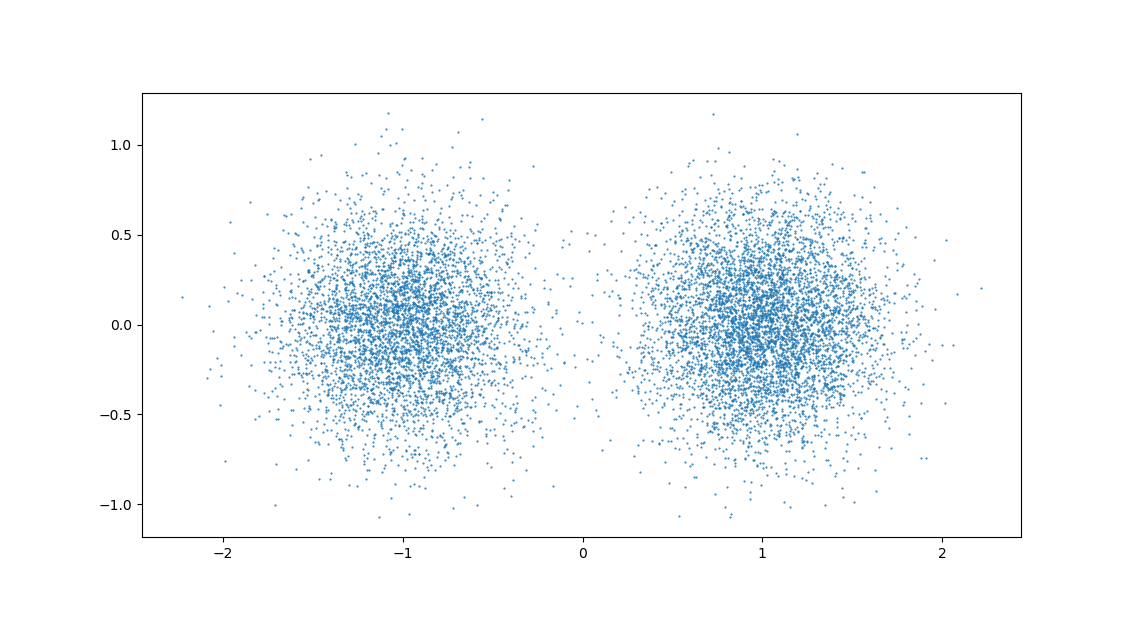
We can display and inspect An I - Q constellation diagram visually using the following where just the first 10000 elements from rx\_data are used.

*plt.figure()*

*plt.axes().set\_aspect("equal")*

*plt.scatter(rx\_data[:10000].real,rx\_data[:10000].imag,s=1,marker=".")*

*plt.show()*



We can also visually inspect the recovered image by rearranging the received data into a decimal-valued matrix corresponding to the original image dimensions and using imshow function to display and we can see some noise points in the recovered image.

*rx\_im = np.packbits(rx\_bin).reshape(tx\_im.size[1],tx\_im.size[0])*

*plt.figure()*

*plt.imshow(np.array(rx\_im), cmap="gray", vmin=0, vmax=255)*

*plt.show()*



To determine the number of bit errors a bitwise comparison of the transmitted and received binary matrices should be made, summing the case of unequal elements. The bit error ratio (ber) is obtained by dividing the number of errors by the total number of bits, 8 \* Npixels. The following code shows the calculation of the ber for a range of dB values for the signal-to-noise ratio and plots them as 𝑏𝑒𝑟 (logarithmic axis) vs 𝑠𝑛𝑟 in dB. Also include a curve showing the theoretical dependence for BPSK.

*x = np.empty(0)*

*y1 = np.empty(0)*

*y2 = np.empty(0)*

*psk = komm.PSKModulation(2)*

*tx\_data = psk.modulate(tx\_bin)*

*for snrdb in range(27, 90, 1):*

*awgn = komm.AWGNChannel(snr=10 \*\* ((snrdb / 10) / 10))*

*# simulate Noisy Channel*

*rx\_data = awgn(tx\_data)*

*rx\_bin = psk.demodulate(rx\_data)*

*ber = np.sum(tx\_bin != rx\_bin) / (Npixels \* 8)*

*print(ber)*

*x = np.append(x, (snrdb / 10))*

*y1 = np.append(y1, ber)*

*ber\_nm = 0.5 \* scipy.special.erfc(np.sqrt(10 \*\* ((snrdb / 10) / 10)))*

*y2 = np.append(y2, ber\_nm)*

*print(ber\_nm)*

*plt.figure()*

*plt.scatter(x, y1) #plot points*

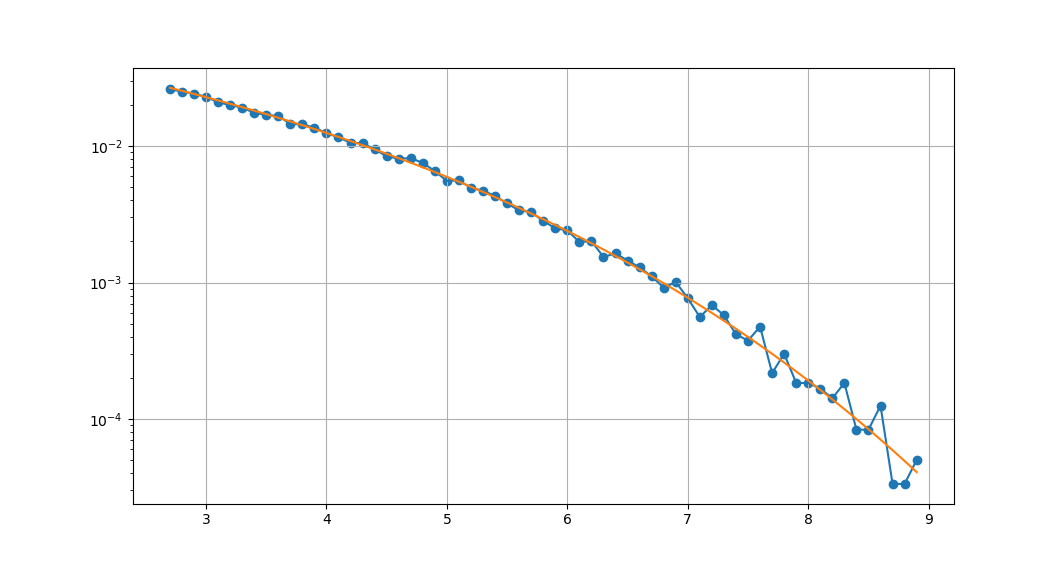
*plt.plot(x, y1) #plot lines*

*plt.plot(x, y2)*

*plt.yscale("log")*

*plt.grid(True)*

*plt.show()*

**

1. **Parity Check**

Since there is always noise in the transmitted image at different signal-to-noise ratios, we introduce Parity Check and Automatic Repeat-reQuest (ARQ). In this part, we consider the data as consisting Npixels × 8 - bit words and replace the least significant bit of each 8-bit word with a parity bit, setting the 8th bit to the modulo 2 sum of the previous 7 bits corresponds to even parity. The parity test of the received data is done by looking at the modulo 2 sum of each 8-bit word. If this is different from the parity setting we should do an Automatic Repeat-reQuest (ARQ) and resend the word, here is the relevant code.

*x = np.empty(0)*

*yber = np.empty(0)*

*yuber = np.empty(0)*

*yarq = np.empty(0)*

*for snrdb in range(27, 90, 1):*

*print(snrdb)*

*ARQs = 0*

*# BPSK*

*psk = komm.PSKModulation(2)*

*# Quadra - ture Phase Shift Keying (QPSK)*

*# psk = komm.PSKModulation(4, phase\_offset=np.pi / 4)*

*# 4 - QAM*

*# qam = komm.QAModulation(4, base\_amplitudes=1 / np.sqrt(2))*

*# 16 - QAM*

*# qam = komm.QAModulation(16, base\_amplitudes=1 / np.sqrt(10))*

*# 256 - QAM*

*# qam = komm.QAModulation(256, base\_amplitudes=1 / np.sqrt(170))*

*print(psk.energy\_per\_symbol, psk.bits\_per\_symbol)*

*awgn = komm.AWGNChannel(snr=10 \*\* ((snrdb / 10) / 10))*

*rx\_bin = np.empty(0)*

*for num in range(0, Npixels \* 8, 8):*

*# add even parity check bit(8th bit each word)*

*tx\_bit\_stream = np.array(tx\_bin[num:num + 8])*

*tx\_bit\_stream[7] = (np.sum(tx\_bit\_stream) - tx\_bit\_stream[7]) % 2*

*tx\_bin[num:num + 8] = tx\_bit\_stream*

*# simulate Noisy Channel*

*tx\_data\_bit\_stream = psk.modulate(tx\_bit\_stream)*

*rx\_data\_bit\_stream = awgn(tx\_data\_bit\_stream)*

*rx\_bin\_bit\_stream = psk.demodulate(rx\_data\_bit\_stream)*

*# parity test*

*if (np.sum(rx\_bin\_bit\_stream) % 2):*

*# Automatic Repeat-reQuest (ARQ)*

*rx\_data\_bit\_stream = awgn(tx\_data\_bit\_stream)*

*rx\_bin\_bit\_stream = psk.demodulate(rx\_data\_bit\_stream)*

*ARQs += 1*

*rx\_bin = np.append(rx\_bin, rx\_bin\_bit\_stream)*

*# Calculate ber, uncorrected ber and ratio of ARQs*

*ber = np.sum(tx\_bin != rx\_bin) / (Npixels \* 8)*

*uber = 0.5 \* scipy.special.erfc(np.sqrt(10 \*\* ((snrdb / 10) / 10) / psk.bits\_per\_symbol))*

*print(uber)*

*x = np.append(x, (snrdb / 10))*

*yber = np.append(yber, ber)*

*yuber = np.append(yuber, uber)*

*print(ber, ARQs / (Npixels \* 8))*

*yarq = np.append(yarq, ARQs / (Npixels \* 8))*

*plt.figure()*

*# plot ber*

*plt.scatter(x, yber)*

*plt.plot(x, yber, label=f"Ber")*

*# plot theoretical curve for uncorrected ber*

*plt.plot(x, yuber, label=f"Theoretical curve")*

*# plot ratio of ARQs*

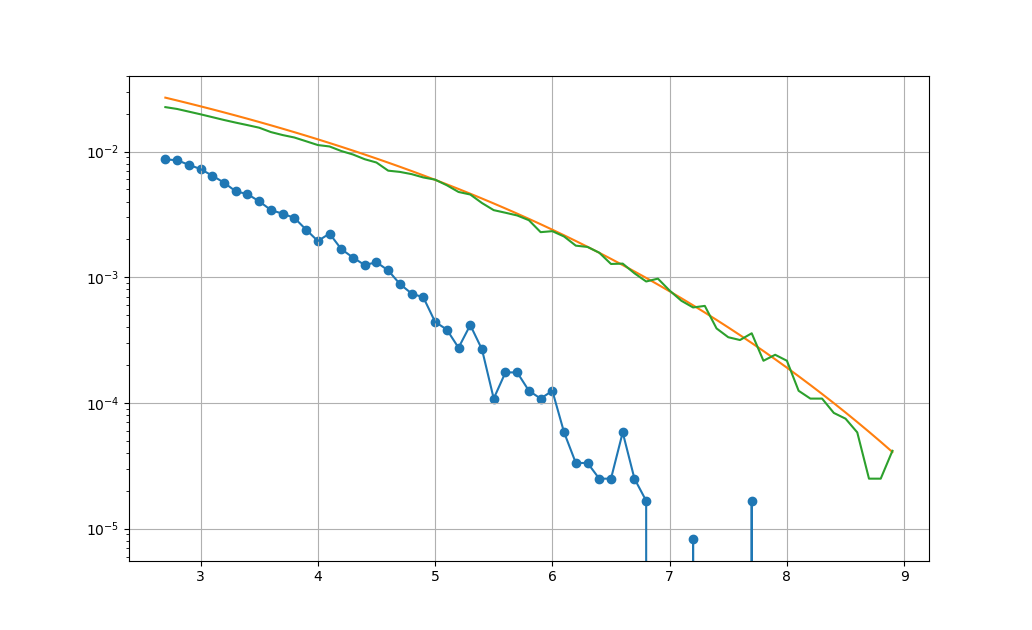
*plt.plot(x, yarq, label=f"ARQs ratio")*

*plt.yscale("log")*

*plt.grid(True)*

*plt.show()*

Visually inspect the received image and determine the bit error ratio. On a single graph, plot the 𝑏𝑒𝑟 as a function of 𝑠𝑛𝑟 in dB, along with the theoretical curve for uncorrected 𝑏𝑒𝑟 and the ratio of the total number of ARQs to the number of bits, Here are the results.



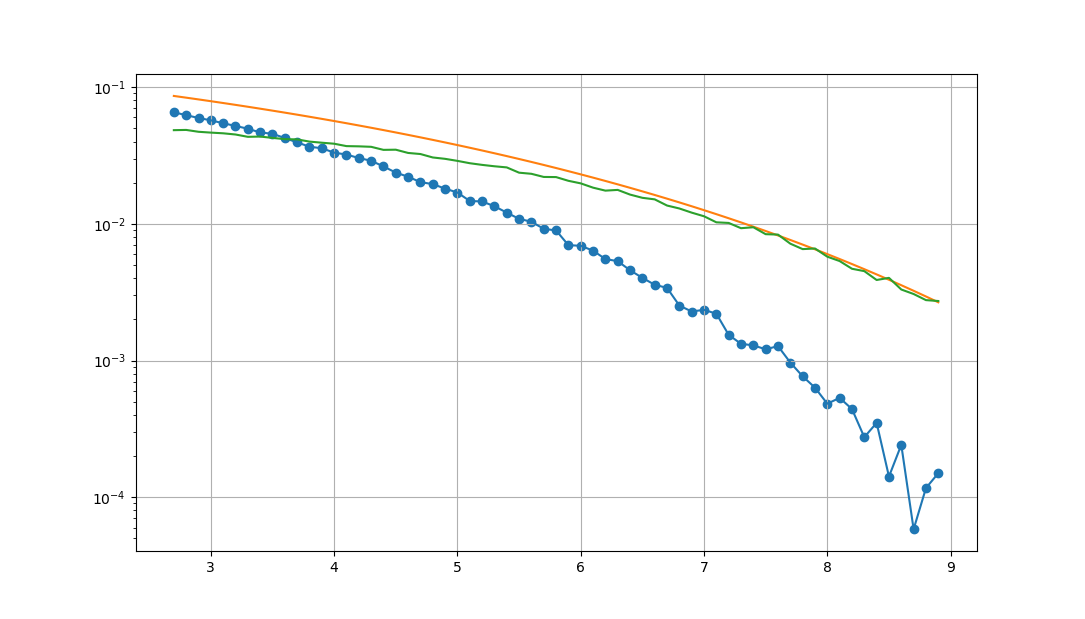
The recovered image was obtained at a signal-to-noise ratio of 9. With the addition of Parity Check and Automatic Repeat-reQuest (ARQ), it can be seen that there is no longer any noticeable noise point and that the ber and the ratio of ARQs are getting lower and lower as the signal-to-noise ratio gets higher and higher.

1. **QAM Modulation**
2. **Quadrature Phase Shift Keying (QPSK)**

By changing the variable psk to the following configuration, we can repeat the exercise for Quadrature Phase Shift Keying (QPSK).

*psk = komm.PSKModulation(4,phase\_offset=np.pi/4)*

The following plots show the results at different signal-to-noise ratios.



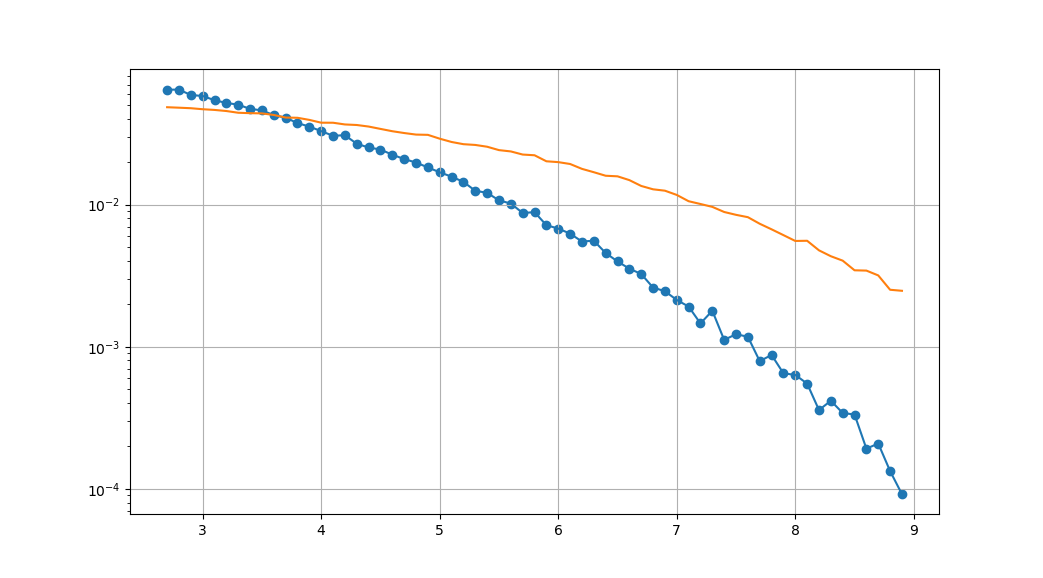
1. **4 – QAM**

By changing the variable qam to the following configuration, we can repeat the exercise for 4 – QAM, we can inspect the value qam.energy\_per\_symbol and then set base\_amplitudes to a value such that qam.energy\_per\_symbol becomes unity.

*print(psk.energy\_per\_symbol, psk.bits\_per\_symbol)*

*qam = komm.QAModulation(4, base\_amplitudes=1 / np.sqrt(2))*

The following plots show the results at different signal-to-noise ratios.

****

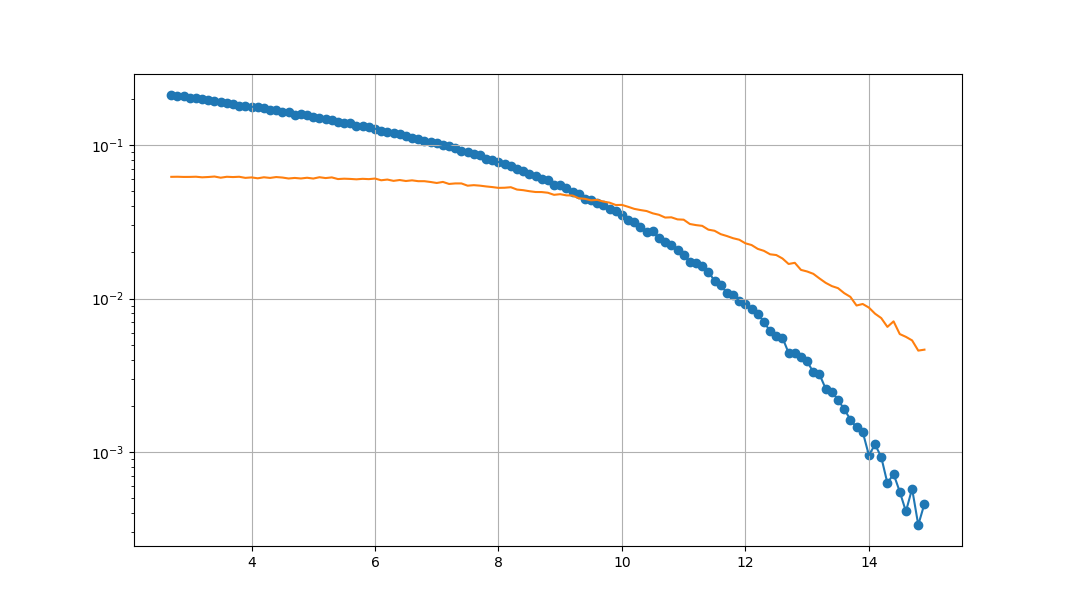
1. **16 – QAM**

By changing the variable qam to the following configuration, we can repeat the exercise for 16 – QAM, we can inspect the value qam.energy\_per\_symbol and then set base\_amplitudes to a value such that qam.energy\_per\_symbol becomes unity.

*print(psk.energy\_per\_symbol, psk.bits\_per\_symbol)*

*qam = komm.QAModulation(16, base\_amplitudes=1 / np.sqrt(10))*

The following *plots* show the results at different signal-to-noise ratios.

****

1. **256 – QAM**

By changing the variable qam to the following configuration, we can repeat the exercise for 256 – QAM, we can inspect the value qam.energy\_per\_symbol and then set base\_amplitudes to a value such that qam.energy\_per\_symbol becomes unity.

*print(psk.energy\_per\_symbol, psk.bits\_per\_symbol)*

*qam = komm.QAModulation(256, base\_amplitudes=1 / np.sqrt(170))*

The following *plots* show the results at different signal-to-noise ratios.

