digicomlab3

Torin Storkey

February 2023

1 Introduction

In this lab we investigated the ability of different modulation times to be used in digital data transmission in different signal to noise regimes. The bit error rate for each of the modulation modes was tracked. After this a technique to mitigate the bit error rate which was the addition of a parity bit. Then for each of the modes the difference between the bit error rates with the parity bit and the bit error rates without the parity bit were compared.

2 Modulation and Noise

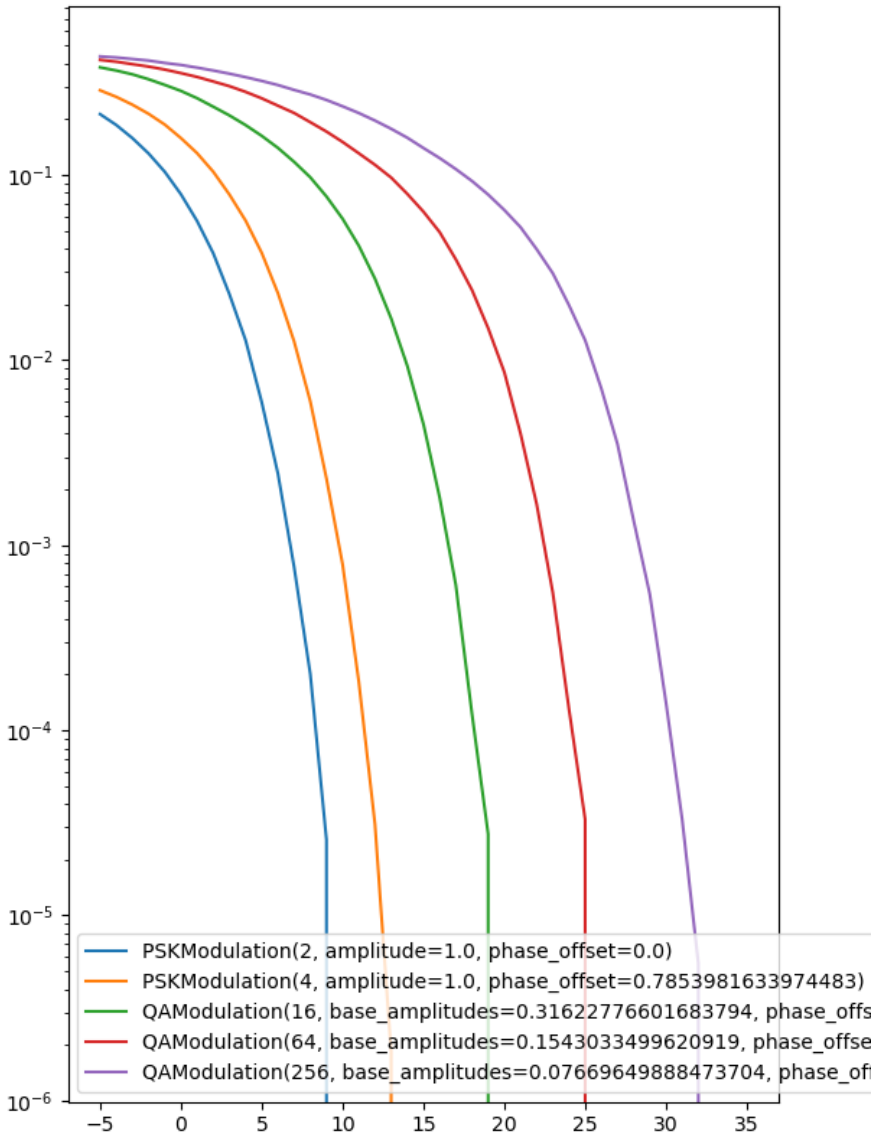
In this lab a number of different modulation types were used. These were binary phase shift keying, Quadrature Phase shift keying, 16 Quadrature Amplitude Modulation (QAM), 64 QAM and 256 QAM. It is important for a fair comparison to be made for the energy per symbol to be equal across the all modulation types. To do this the square root of the average power was used as a scale factor for the modulation constellation ensuring an average energy per symbol of 1. With each of the modes data from a test image was modulated according to the correct type. Then the signal was passed through a noisy channel to add noise. After this it was demodulated. This was used to determine a bit error rate.

3 Bit Error Rate

With digital data it is important that it arrives correctly to the receiver. For most types of data having bits being lost or flipped would corrupt the data making it unusable. To verify that a transmission channel is usable the bit error rate is tracked for the channel by looking at the rate at which the transmitted bit isn't properly resolved by the receiver. This allows the bit error rate to be determined. For a data stream with no redundancy or parity methods the bit error rate must be zero. This often requires a good signal to noise level especially for modulation modes with more bits per symbol. To reduce the signal to noise level requirements parity is used to verify the data transmission.

4 Results

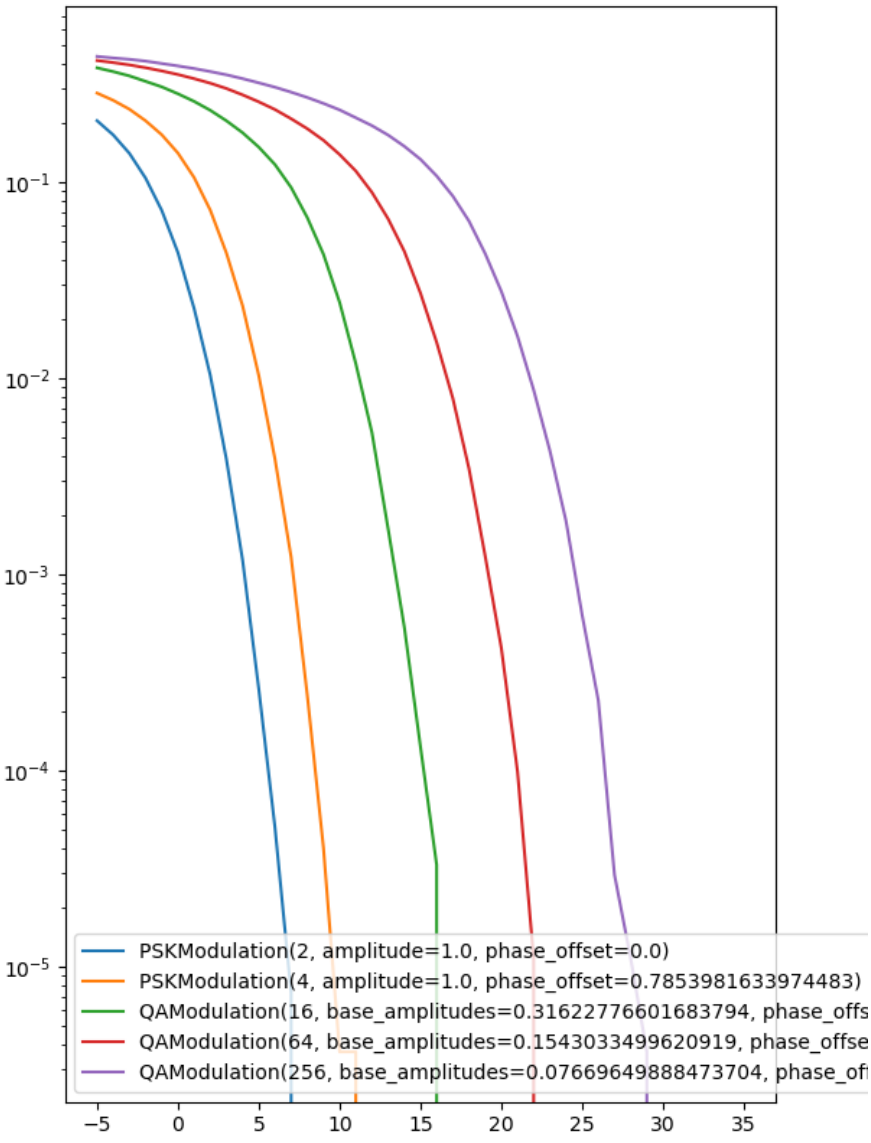
4.1 Bit Error Rate Without Parity



As Can be seen With bit error rate on the y axis and the signal to noise ratio

on the x axis different modulation modes produce different bit error rates for each of the signal to noise ratios.

4.2 Bit Error Rate With Parity

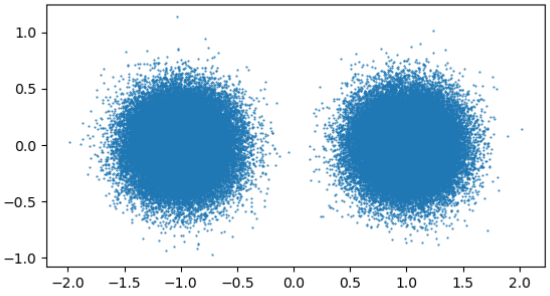


With Parity bits included this improves by around 2 dB for all the modes. This

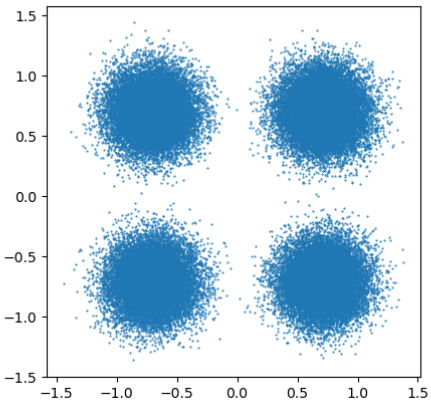
allows larger bits per symbol modulation types to be used at poorer signal to noise ratios.

4.3 example constellations

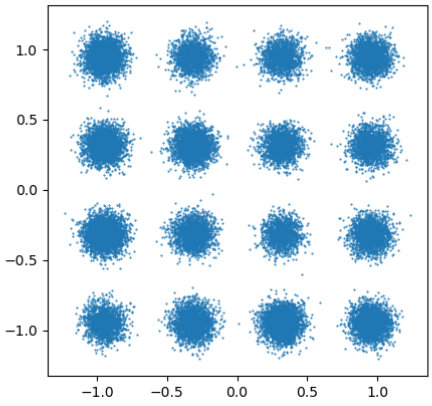
4.3.1 BPSK



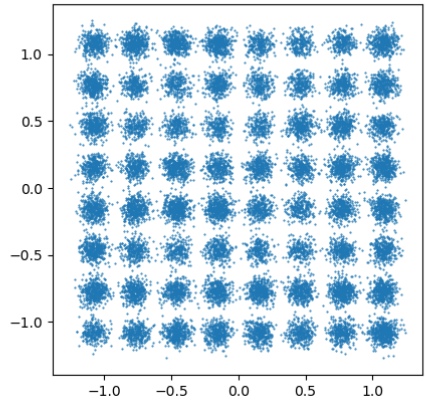
4.3.2 QPSK



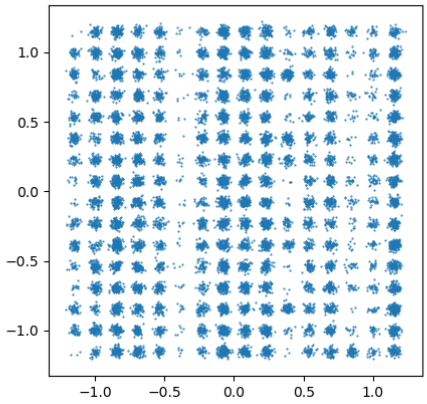
4.3.3 16QAM



4.3.4 64QAM



4.3.5 256QAM



5 Conclusion

The more bits per symbol a modulation method uses the more bit errors are caused for a given signal to noise ratio. Therefore selecting your modulation type based on your signal to noise ratio is important. The signal to noise requirements can be reduced by using a parity bit. This reduces the required signal to noise ratio by 2 dB.

6 Appendix

6.1 Code

```
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
import komm


def close_event():
    plt.close()
    plt.close()

def qam_bases(n):
    n_val = np.sqrt(n)/2
    if n_val != int(n_val):
        raise AssertionError("Not 2^2m")
```

```
    n_val = int(n_val)100m world record
    vals = np.linspace(1,2*n_val-1,n_val)
    total_e = 0
    for q in vals:
        for i in vals:
            print(i,q)
            total_e += q**2+i**2
    e_sym = total_e/len(vals)**2
    vals = vals/np.sqrt(e_sym)
    return vals[0]

def calc_pad(length,sym_size):
    return (sym_size-(length%sym_size))%sym_size

def parity_gen(bitray_i):
    bitray = np.array(bitray_i)
    data = np.concatenate([bitray[i::8] for i in range(7)])
    blocks = data.reshape(7,len(bitray)//8)
    parity = blocks.sum(axis=0)%2
    bitray[7::8]= parity
    return bitray

def parity_check(bitray):
    collections = bitray.reshape((len(bitray)//8,8))
    errors = collections.sum(axis=1)%2
    return errors

class map_track:
    def __init__(self,recive) -> None:
        self.recive = recive
        self.map = np.arange(len(self.recive))

    def map_apply(self,idx):
        self.map = self.map[idx]

    def data_write(self,data):
        self.recive[self.map] = data



interim_plots = False
m = 64
n = qam_bases(m)
print(n)
modulations = [komm.PSKModulation(2),
    komm.PSKModulation(4,phase_offset=np.pi/4)]\
```

```
    + [(komm.QAModulation(16*2**(2*n),qam_bases(16*2**(2*n))))\
    for n in range(3)]
print(modulations)
tx_im = Image.open("imsend-10.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_raw = np.unpackbits(np.array(tx_im))
n_bits = len(tx_bin_raw)



ber_db = range(-5,36)
ber_mods = []
for mod in modulations:
    ber_ray = []
    pad = calc_pad(n_bits,mod.bits_per_symbol)
    tx_bin = np.append(tx_bin_raw,np.zeros(pad))
    for i in ber_db:
        print(mod.energy_per_symbol)
        awgn = komm.AWGNChannel(snr=10**(i/10))
        tx_data = mod.modulate(tx_bin)
        rx_data = awgn(tx_data)
        rx_bin = mod.demodulate(rx_data)
        if pad == 0:
            rx_bin_raw = rx_bin
        else:
            rx_bin_raw = rx_bin[:-pad]
        print(i)
        cmp = tx_bin_raw != rx_bin_raw
        errors = sum(cmp)
        print(errors)
        bit_error_rate = errors/Npixels/8
        ber_ray.append(bit_error_rate)
        rx_im = np.packbits(rx_bin_raw).reshape(tx_im.size[1],tx_im.size[0])
        if interim_plots:
            fig = plt.figure()
            timer = fig.canvas.new_timer(interval = 500)
            timer.add_callback(close_event)
            timer.start()
            plt.imshow(np.array(rx_im),cmap="gray",vmin=0,vmax=255)
            plt.figure()
            plt.axes().set_aspect("equal")
            plt.scatter(rx_data[:100000].real,rx_data[:100000].imag,s=1,marker="
            plt.show()
```

```
        ber_mods.append(ber_ray)
        if interim_plots:
            timer.stop()
            plt.plot(ber_db,ber_ray)
            plt.yscale("log")
            plt.show()
            timer.start()
    if interim_plots:
        timer.stop()
    for item,modulation in zip(ber_mods,modulations):
        title = repr(modulation)
        plt.plot(ber_db,item,label=title)
    plt.legend()
    plt.yscale("log")
    plt.show()



    print(len(tx_bin_raw))
    print(len(tx_bin_raw))
    tx_bin_raw = parity_gen(tx_bin_raw)
    print(tx_im.size)
    tx_img = np.packbits(tx_bin_raw).reshape(tx_im.size[1],tx_im.size[0])
    plt.imshow(np.array(tx_img),cmap="gray",vmin=0,vmax=255)
    plt.show()
    print(f"lentxbin{len(tx_bin_raw)}")
    ber_db = range(-5,36)
    ber_mods_p = []
    for mod in modulations:
        ber_ray = []
        pad = calc_pad(n_bits,np.lcm(8,mod.bits_per_symbol))
        tx_bin = np.append(tx_bin_raw,np.zeros(pad))
        tx_bin = parity_gen(tx_bin)
        for i in ber_db:
            #print(len(tx_bin),len(tx_bin_raw))
            print(f"decibels {i}")
            parity_satisfied = False
            awgn = komm.AWGNChannel(snr=10**(i/10))
            tx_data = mod.modulate(tx_bin)
            rx_data = awgn(tx_data)
            rx_bin = mod.demodulate(rx_data)
            tx_bin_interim = tx_bin
            rx_bin_interim = rx_bin
            map_apply_track = map_track(rx_bin)
            while not parity_satisfied:
```

```
        print(np.sum(tx_bin),np.sum(rx_bin))
        parity = parity_check(rx_bin_interim).astype(bool)
        if sum(parity) == 0:
            break
        mask = np.repeat(parity,8)
        indexes = np.nonzero(mask)[0]
        #print("lens before and after")
        #print(len(rx_bin_interim),len(tx_bin_interim))
        rx_bin_interim = rx_bin_interim[indexes]
        tx_bin_interim = tx_bin_interim[indexes]
        map_apply_track.map_apply(indexes)
        #print(len(rx_bin_interim),len(tx_bin_interim))
        retran_pad = calc_pad(len(tx_bin_interim),np.lcm(8,mod.bits_per_symb
        tx_bin_trans = np.append(tx_bin_interim,np.zeros(retran_pad))
        awgn = komm.AWGNChannel(snr=10**(i/10))
        tx_data = mod.modulate(tx_bin_trans)
        rx_data = awgn(tx_data)
        rx_bin_uncut = mod.demodulate(rx_data)
        print("before")
        print(rx_bin_interim,rx_bin)
        if retran_pad == 0:
            rx_bin_interim[:] = rx_bin_uncut
        else:
            rx_bin_interim[:] = rx_bin_uncut[:-retran_pad]
        map_apply_track.data_write(rx_bin_interim)
        print("after")
        print(rx_bin_interim,rx_bin)
    if pad == 0:
        rx_bin_raw = rx_bin
    else:
        rx_bin_raw = rx_bin[:-pad]
    cmp = tx_bin_raw != rx_bin_raw
    errors = sum(cmp)
    print(errors)
    bit_error_rate = errors/Npixels/8
    ber_ray.append(bit_error_rate)
    rx_im = np.packbits(rx_bin_raw).reshape(tx_im.size[1],tx_im.size[0])
    if interim_plots:
        fig = plt.figure()
        timer = fig.canvas.new_timer(interval = 500)
        timer.add_callback(close_event)
        timer.start()
        plt.imshow(np.array(rx_im),cmap="gray",vmin=0,vmax=255)
        plt.figure()
        plt.axes().set_aspect("equal")
        plt.scatter(rx_data[:100000].real,rx_data[:100000].imag,s=1,marker="
```

```
            plt.show()

    ber_mods_p.append(ber_ray)
    if interim_plots:
        timer.stop()
        plt.plot(ber_db,ber_ray)
        plt.yscale("log")
        plt.show()
        timer.start()
if interim_plots:
    timer.stop()

plt.figure()
for item,modulation in zip(ber_mods_p,modulations):
    title = repr(modulation)
    plt.plot(ber_db,item,label=title)
plt.legend()
plt.yscale("log")
plt.figure()
for item,modulation in zip(ber_mods,modulations):
    title = repr(modulation)
    plt.plot(ber_db,item,label=title)
plt.legend()
plt.yscale("log")
plt.show()
```