****

ENG4052: Digital Communication 4 (2022-23)

Lab3: Bit Errors and Parity Checking

**Haoshi Huang (2635088H)**

**Submission Date**

**22/2/2023**

**1 Introduction**

Firstly, this lab will simulate transmission process of BPSK and QPSK in the communication channel with **Additive White Gaussian Noise (AWGN)** at different **Signal Noise Ratio (snr)**. Secondly, we will use a simple Parity Checking to mitigate the effects of AWGN in PSK and QAM with **Bit-error-ratio (ber)** to evaluate the effects. Finally, we plot the ber as a function of snr in dB respectively, in order to draw conclusion.

We will use Library komm 0.7.1 to modulate and demodulate signal and create AWGN. Library NumPy1.23 is imported to implement advanced math operation. Library Scipy 1.9.2 has *special.erfc(snr)* function to compute theoretical ber. Library Pillow 9.2.0 is used to read image file. Library matplotlib 3.6 is used to display graphics and plot the bit error ratio as a function of signal noise ratio in dB.

**2 Noisy Channel Simulation**

**2.1 Digital image data**

Taking a grayscale image as an example as shown in the Fig. 2.1, it’s read as 2-dimensional arrays with the help of Lib pillow. Using the function *numpy.unpackbits()* changes the data to single dimension, which is helpful for processing data and transmission.

树林旁的照片

中度可信度描述已自动生成

*Figure 2.1 a grayscale image*

For reusing code, **Class imgInfo** is defined as shown in the Fig. 2.2, which has member property including size of image, number of bits per pixel, etc.

图形用户界面, 文本, 应用程序

描述已自动生成

*Figure 2.2 Class imgInfo*

Method *openImagetoBin* is used to open and show image file and convert the image data to binary signal.

Method *createParityCod*e (referring to **3 Parity Check**) can generate binary code with parity information according to the parameter *parity\_mode* (0 means even checking, 1 means odd checking).

Method *displayDemodImage* is used to show demodulated image, comparing with the original image.

**2.2 Additive White Gaussian Noise**

Additive White Gaussian Noise (AWGN) is a type of basic random noise, which is always used to simulate the effects of practical noise in electronic system, such as communication system. ‘Additive’ noise means that the relationship between the noise and signal is addition. ‘White’ noise means the power spectrum density of stochastic noise is a constant. ‘Gaussian’ noise means that the stochastic noise obeys the Gaussian Distribution.

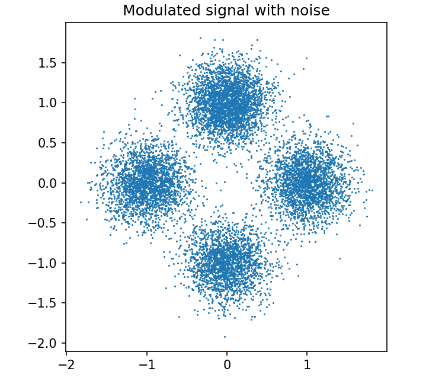
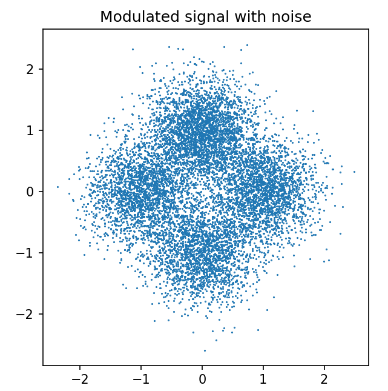
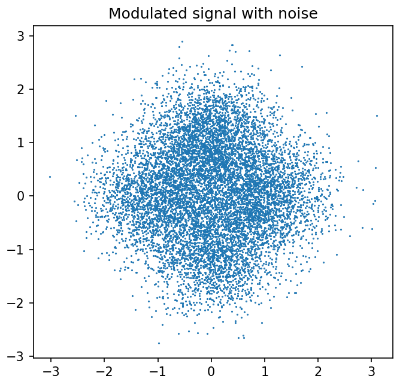
In the code, AWGN is hypothetically added at the moment of receiving single byte modulated signal.

文本

描述已自动生成

*Figure 2.3 Simulate receiver*

Taking QPSK as an example, the Fig. 2.4 plot the modulated signal with stochastic noise with parameters *phase\_offset = np.pi /2* (the default is *np.pi/4*) in the condition of *SNR* from *2 dB* to *9 dB*. The higher SNR is, the more concentrated the signal is.



*Figure 2.4 Distribution of modulated signal with different SNR*

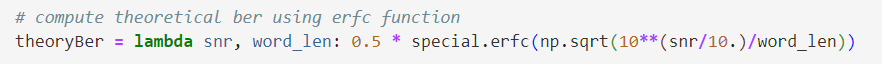
**2.3 Bit Error Ratio**

BER is used to evaluate the effects of noise in this lab including theoretical ber and practical ber.

We use Lib SciPy to compute theoretical ber as the following formula shown. And code is shown in Fig. 2.4.

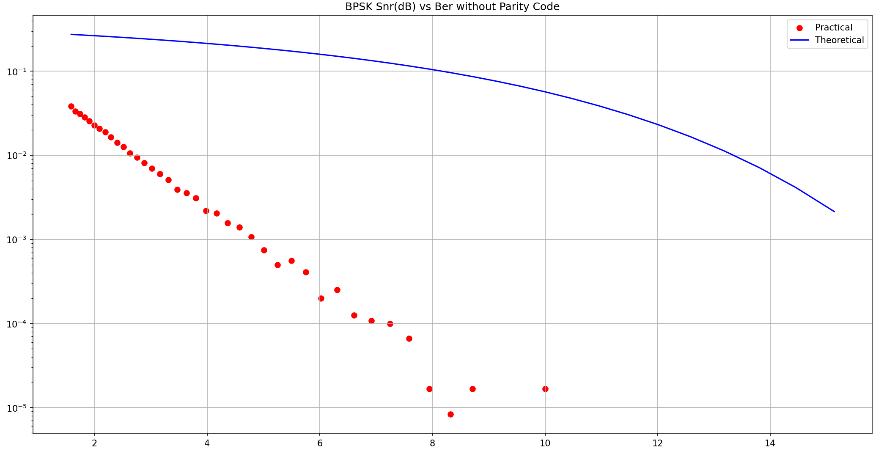
图片包含 示意图

描述已自动生成

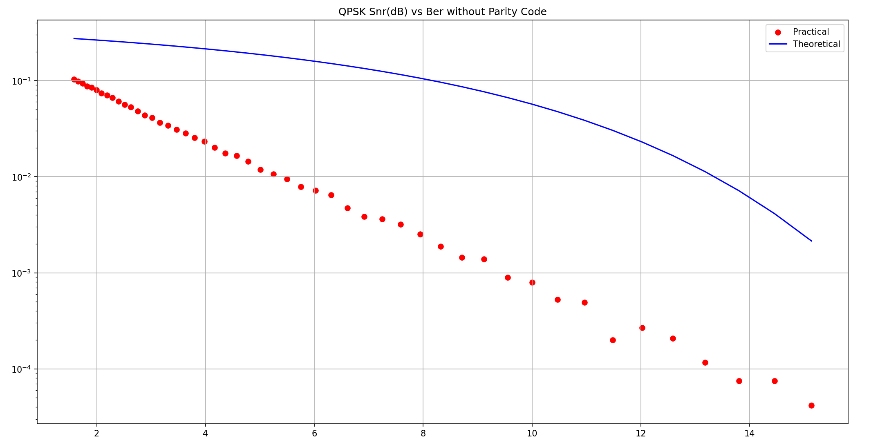


*Figure 2.4 Theoretical Bit Error Ratios using ERFC function*

In assignment 1.4, ber is defined as dividing the number of errors by the total number of bits, 8\*Npixels. The relation plot of log (BER) vs SNR (2 dB – 12 dB) is shown in the Fig. 2.5 with BPSK and in the Fig. 2.6 with QPSK, but without Parity Check.



*Figure 2.5 BPSK without Parity Check*



*Figure 2.6 QPSK without Parity Check*

In assignment 1.5&1.6, ber is defined as the ratio of the total number of Automatic Repeat-reQuest (ARQs) to the number of pixels referring to **3 Parity Check**.

**3 Parity Check in PSK & QAM**

**3.1 Config Modulation**

For reuse code, Class modConfig is defined, whose member has method (‘psk’, ‘qam’), orders (2 (2-psk), 4, 16, 256), snr, base\_amplitude, phase\_offset, modulation object and awgn object, as shown in the Fig. 3.1.

图形用户界面, 应用程序

描述已自动生成

*Figure 3.1 Class imgInfo*

**3.2 Generate Parity Code & Do Parity Check**

At once time transmitting eight bits, i.e., one pixel, we modifying the **lowest** bit to be the parity code because our eyes cannot distinguish the change of gray value due to the change of the lowest bit. The method createParityCode in Class imgInfo can generate the parity code. The core of code is shown in the Fig. 3.2. It’s noted that we need to cast INT type into BOOL type because of we use NOT operator in the odd field. If not, inversing a ‘0’ of INT type data will not get a ‘1’.

图形用户界面, 文本

描述已自动生成

*Figure 3.2 Method createParityCode*

Through the analysis of the logical expression, we can use **single statement** to implement the check operation as shown in the Fig. 3.3.

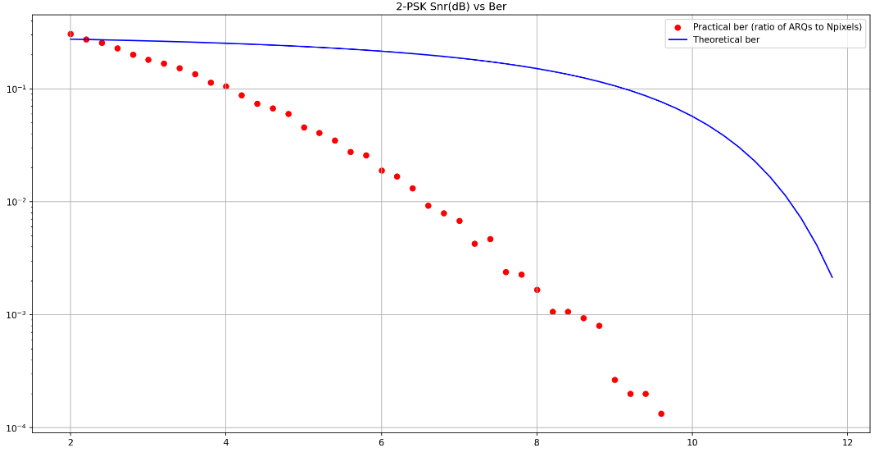
文本

描述已自动生成

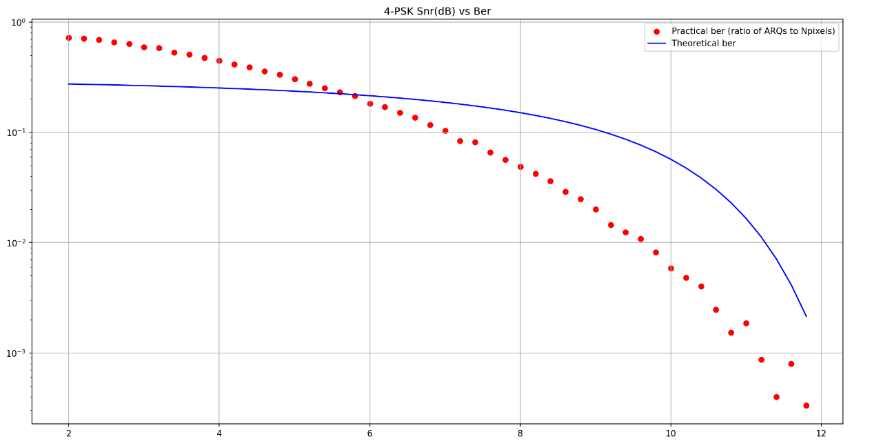
*Figure 3.3 Method doParityCheck*

**3.2 PSK & QAM modulation with Parity Check**

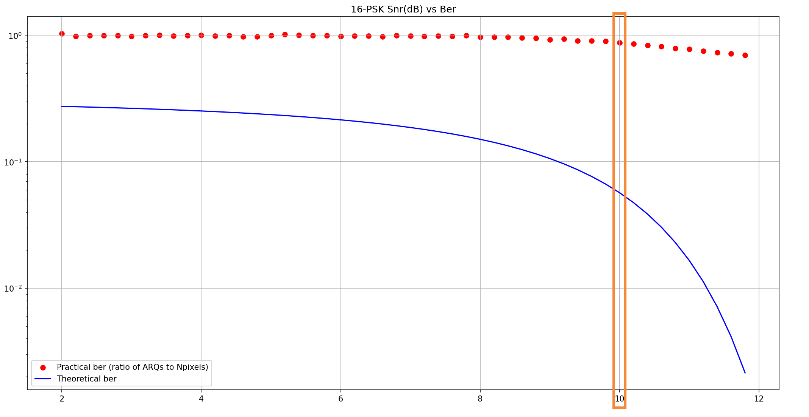
The final step is that we call the encapsulated function above to stimulate PSK and QAM modulation with differential SNR. We can plot the ratio of the total number of ARQs to the number of pixels from the Fig. 3.4 to 3.11. The Bule line represents theoretical Ber; the red scattered points represent practical Ber. The x-axis represents the changes of SNR from 2 dB to 12 dB.



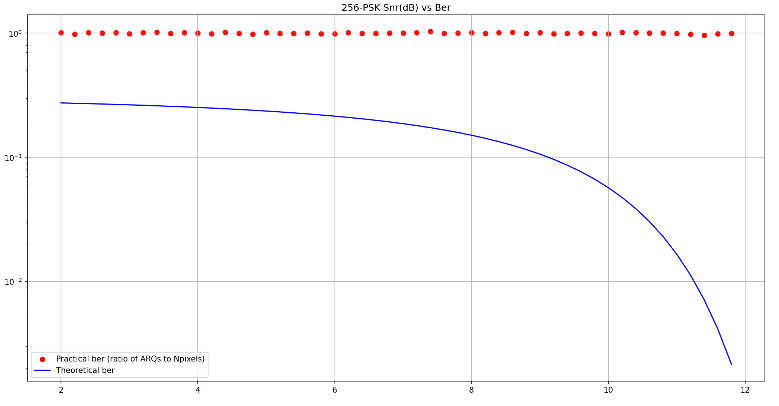
*Figure 3.4* ***BPSK*** *with Parity Check*



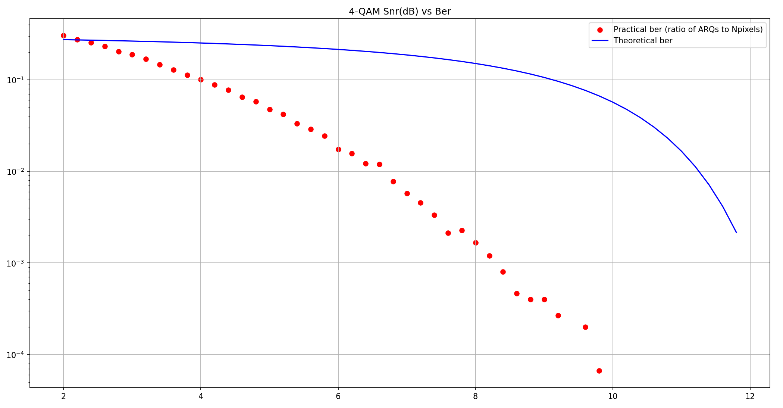
*Figure 3.5* ***QPSK*** *with Parity Check*



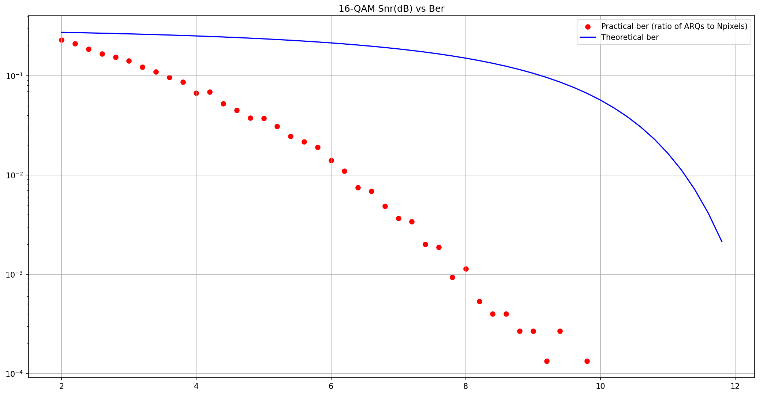
*Figure 3.6* ***16-PSK*** *with Parity Check*



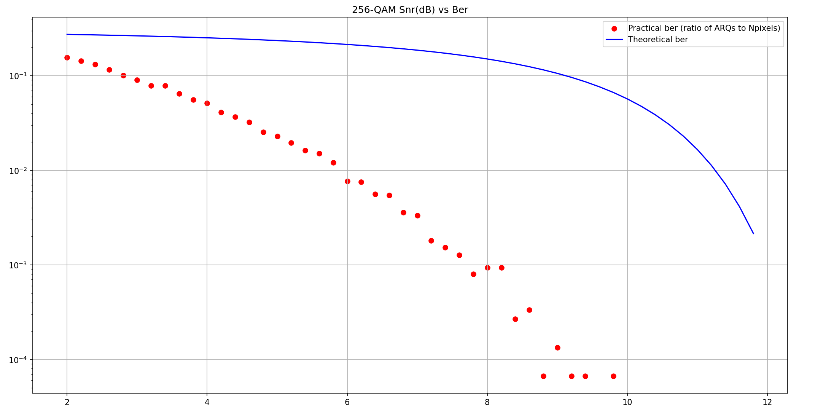
*Figure 3.7* ***256-PSK*** *with Parity Check*



*Figure 3.8* ***4-QAM*** *with Parity Check*



*Figure 3.9* ***16-QAM*** *with Parity Check*



*Figure 3.10* ***256-QAM*** *with Parity Check*

**4 Conclusion**

**4.1 PSK vs QAM**

PSK modulation only use phase to transmit information. QAM modulation is a compound modulation technique, which combines Amplitude and Phase modulation. The amplitude and phase of modulated signal by QAM contains original information which means carrying more information. Comparing the Fig. 3.5 (**QPSK**) to the Fig. 3.8 (**4-QAM**), with the same noise and transmission bandwidth, QAM has stronger anti-interference capability and carries more information. But QPSK is susceptible to noise. Analyzing the constellation diagram of PSK and QAM can obtain the same conclusion.

**4.2 Effects of AWGN**

The increment of SNR means the stronger AWGN. With the same modulation method and the number of orders, taking 4-QAM as an example in the Fig. 4.1, bit-error-ratio is negatively correlated with SNR.

图表, 散点图

描述已自动生成

*Figure 4.1* ***4-QAM*** *with Parity Check*

Additionally, for PSK modulation, the higher the order, the more likely it is to make errors and receive noise interference. In other words, if we want to get a greater transmission accuracy, the higher the orders, the higher threshold of the signal-to-noise ratio needs to be stronger constrained as shown in the Fig. 3.6, where the orange box field is the threshold of SNR.

**4.3 Limitation of Parity Code**

Comparing to the Fig. 3.4, 3.5, 3.8, 3.9, 3.10, the relation plot shows the Ber is much lower than theoretical Ber in the most SNR areas. We can draw the conclusion that when using parity check, even passing the checking at the receiving end, the demodulated data may not be the same as the original information. So parity check is simply implemented, but its checking capability is very limited.

**Appendix:**

**AWGNChannel.py**

'''

Author : Eureke

Date : 2023-02-13 10:28:40

LastEditors : Marcus Wong

LastEditTime : 2023-02-22 23:04:42

Description : AWGN Channel

'''

import numpy as np

from PIL import Image

from matplotlib import pyplot as plt

import komm

from scipy import special

from ParityCheck import imgInfo, theoryBer

# compute ber in practice

practiceBer = lambda tx\_bin, rx\_bin : np.sum([pix[0] != pix[1] for pix in zip(tx\_bin, rx\_bin)]) / tx\_bin.size

if \_\_name\_\_ == '\_\_main\_\_':

# 注意当前打开的工作路径的相对路径，这里打开的文件夹是LAB，所以要加子文件夹路径 ./Lab3/

is\_plot = True # True

# open image and binary information

fp = "./Lab3/DC4\_150x100.pgm"

word\_len = 8 # 256 bits per pixel

img = imgInfo(fp, word\_len)

# save ber ,theoretical ber and snr

ber\_out = np.empty(0)

theory\_ber\_out = np.empty(0)

snr\_out = np.empty(0)

for i in np.arange(2., 12., 0.2):

# BPSK modulation

psk = komm.PSKModulation(4)

# Additive white gaussian noise (AWGN) channel

# snr -> signal-to-noise ratio

# snr = np.array([10\*\*(6./10.), 10\*\*(5./10.), 10\*\*(4./10.), 10\*\*(3./10.), 10\*\*(2./10.), 10\*\*(1./10.)])

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

tx\_data = psk.modulate(img.imBin)

# demodulation

rx\_data = awgn(tx\_data)

if (not is\_plot):

# is\_plot = not is\_plot

plt.figure()

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

plt.title("Modulated signal with noise")

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

plt.show()

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

if (False):

img.displayDemodImage(rx\_bin.astype(np.bool\_))

snr = 10\*\*(i/10.)

ber = practiceBer(img.imBin, rx\_bin)

theory\_ber = theoryBer(snr, img.word\_len)

snr\_out = np.append(snr\_out, snr)

ber\_out = np.append(ber\_out, ber)

theory\_ber\_out = np.append(theory\_ber\_out, theory\_ber)

print("snr :", snr)

print('ber practice: {:.30}%'.format(ber \* 100.0 ))

print('ber real: {:.3}%'.format(theory\_ber \* 100.0 ))

if (True):

plt.figure()

plt.title("QPSK Snr(dB) vs Ber without Parity Code")

plt.scatter(snr\_out, ber\_out, color='r', label='Practical')

plt.plot(snr\_out, theory\_ber\_out, color='b', label='Theoretical')

plt.yscale("log")

plt.grid(True)

plt.legend()

plt.show()

**ParityCheck.py**

'''

Author : Eureke

Date : 2023-02-17 11:49:10

LastEditors : Marcus Wong

LastEditTime : 2023-02-22 22:04:22

Description : ParityCheck

'''

import numpy as np

from PIL import Image

from matplotlib import pyplot as plt

import komm

from scipy import special

class imgInfo:

def \_\_init\_\_(self, fp, word\_len):

self.imBin, self.imSize = self.openImagetoBin(fp)

self.Npixels = self.imSize[1] \* self.imSize[0]

self.word\_len = word\_len

# open image file

def openImagetoBin(self, filePath):

im = Image.open(filePath)

if (True):

plt.figure()

plt.imshow(np.array(im),cmap="gray",vmin=0,vmax=255)

plt.show()

imBin = np.unpackbits(np.array(im))

print('original shape: ', imBin.shape)

return imBin, im.size

# use parity code before transmission

def createParityCode(self, parity\_mode):

'''

parity: choose even(0) or odd(1)

'''

indices = np.arange(self.word\_len - 1, self.imBin.size, self.word\_len, dtype=int)

# print(indices[10:])

self.parityCode = np.copy(self.imBin).astype(np.bool\_)

if (not parity\_mode):

# even

for i in indices:

self.parityCode[i] = np.sum(self.imBin[i-7:i]) % 2

else:

# odd

for i in indices:

self.parityCode[i] = ~np.sum(self.imBin[i-7:i]) % 2

self.parityCode = self.parityCode.astype(np.int32)

return self.parityCode

# show demodulated image

def displayDemodImage(self, rx\_bin):

# demod signal with noise

rx\_im = np.packbits(rx\_bin).reshape(self.imSize[1], self.imSize[0])

plt.figure()

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

plt.show()

class modConfig:

def \_\_init\_\_(self, method, orders, snr, base\_amplitudes, phase\_offset):

self.method = method

self.orders = orders

self.snr = snr

self.base\_amplitudes = base\_amplitudes

self.phase\_offset = phase\_offset

self.modulation, self.awgn = self.set\_modulation()

# create komm's modulation object

def set\_modulation(self):

if self.method == 'psk':

modulation = komm.PSKModulation(self.orders, amplitude=self.base\_amplitudes, phase\_offset=self.phase\_offset)

elif self.method == 'qam':

modulation = komm.QAModulation(self.orders, base\_amplitudes=self.base\_amplitudes, phase\_offset=self.phase\_offset)

# Additive white gaussian noise(AWGN)

awgn = komm.AWGNChannel(self.snr)

return modulation, awgn

# self-add snr

def set\_snr(self, new\_snr):

self.snr = new\_snr

self.modulation, self.awgn = self.set\_modulation()

# simulate transmitter send single original word

tx\_ori = lambda signal, word\_len, start\_index : signal[start\_index: start\_index + word\_len]

# region parityCheck

'''

even\_true

(not parity\_mode) and (not (sum % 2)) => 1

odd\_true

parity\_mode and (sum % 2) => 1

equal to not (parity\_mode ^ (sum % 2))

even false

(not parity\_mode) and (sum % 2) => 0

odd\_false

parity\_mode and (not (sum % 2)) => 0

equal to parity\_mode ^ (sum % 2)

'''

# endregion

doParityCheck = lambda rx\_bin, parity\_mode : not (parity\_mode ^ (np.sum(rx\_bin) % 2))

# simulate receiver demodluate signal

def rx\_sim(s\_mod, mod\_config):

rx\_data = mod\_config.awgn(s\_mod)

rx\_bin = mod\_config.modulation.demodulate(rx\_data)

# print(rx\_bin)

return rx\_bin

# stimulate transmit single img

def transmission(img, mod\_config, parity\_mode):

# save checked demodulation signal

rx\_bin = np.empty(0)

# arq counter

arq\_cnt = 0

# simulate single step of transmission

hasTransPixel = 0 # pixel number has been trasmitted

while hasTransPixel < img.Npixels:

# original signal per step

tx\_single = tx\_ori(img.parityCode, img.word\_len, hasTransPixel \* img.word\_len)

# modulation, transmit single word

s\_mod = mod\_config.modulation.modulate(tx\_single)

# receive and demodulate single word

rx\_single = rx\_sim(s\_mod, mod\_config)

# print('tx\_single: ', tx\_single)

# print('tx\_single size: ', tx\_single.size)

# print('s\_mod: ', s\_mod)

# print('s\_mod size: ', s\_mod.size)

# print('rx\_single: ', rx\_single)

# print('rx\_single size: ', rx\_single.size)

# judge transmission error

if doParityCheck(rx\_single, parity\_mode): # no error

hasTransPixel += 1

rx\_bin = np.append(rx\_bin, rx\_single)

# print("pass")

else: # error, repeat transmit

arq\_cnt += 1

# print('error')

# bit error ratio

ber = arq\_cnt / img.Npixels

# print('arq counter: ', arq\_cnt)

# print('Npixels: ', Npixels)

print('snr(dB): ', mod\_config.snr)

print('bit error ratio: {:.3}%'.format(ber \* 100.0 ))

if (False):

img.displayDemodImage(rx\_bin.astype(np.bool\_))

return ber, mod\_config.snr

# compute theoretical ber using erfc function

theoryBer = lambda snr, word\_len: 0.5 \* special.erfc(np.sqrt(10\*\*(snr/10.)/word\_len))

def repeatTransmit(img, parity\_mode, method, orders, snr\_ctrl, base\_amplitudes=1., phase\_offset=0.):

print("Start " + str(orders) + '-' + method + "modulation:")

# create parity code

img.createParityCode(parity\_mode)

# print("original bin: ", img.imBin[88:104])

# print("parity code: ", img.parityCode[88:104])

# initial modulation config

# snr = 10\*\*(6./10.) # dB(信噪比强度) = 10\*lg(signal/noise) = 6 => signal/noise 约为 3-4倍

mod\_config = modConfig(method, orders, snr\_ctrl[0], base\_amplitudes, phase\_offset)

# save ber and snr of each trasmission single image

ber\_out = np.empty(0)

theory\_ber\_out = np.empty(0)

snr\_out = np.empty(0)

for i in np.arange(snr\_ctrl[0], snr\_ctrl[1], snr\_ctrl[2]):

snr = 10\*\*(i/10.)

mod\_config.set\_snr(snr)

ber, snr = transmission(img, mod\_config, parity\_mode)

theory\_ber = theoryBer(snr, img.word\_len)

ber\_out = np.append(ber\_out, ber)

theory\_ber\_out = np.append(theory\_ber\_out, theory\_ber)

snr\_out = np.append(snr\_out, i)

# print(ber\_out)

# print(theory\_ber\_out)

# print(snr\_out)

if (True):

plt.figure()

plt.title(str(orders) + '-' + method.upper() + " Snr(dB) vs Ber")

plt.scatter(snr\_out, ber\_out, color='r', label='Practical ber (ratio of ARQs to Npixels)')

plt.plot(snr\_out, theory\_ber\_out, color='b', label='Theoretical ber')

plt.yscale("log")

plt.grid(True)

plt.legend()

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

# open image and binary information

fp = './Lab3/DC4\_150x100.pgm'

# fp = './Lab3/DC4\_640x480.pgm'

word\_len = 8 # 256 bits per pixel

img = imgInfo(fp, word\_len)

# use parity mode even(0), odd(1)

parity\_mode = 0

snr\_ctrl = [2., 12., 0.2]

# psk modulation

repeatTransmit(img=img, parity\_mode=parity\_mode, method='psk', orders=2, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, parity\_mode=parity\_mode, method='psk', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, parity\_mode=parity\_mode, method='psk', orders=16, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, parity\_mode=parity\_mode, method='psk', orders=256, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

# qam mudulation

repeatTransmit(img=img, parity\_mode=parity\_mode, method='qam', orders=4, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, parity\_mode=parity\_mode, method='qam', orders=16, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)

repeatTransmit(img=img, parity\_mode=parity\_mode, method='qam', orders=256, snr\_ctrl=snr\_ctrl, base\_amplitudes=1., phase\_offset=0.)