ENG4052: Digital Communications 4: Forward Error Correction

Ran Shuai (2633609R)

Table Of Contents

Introduction

Forward Error Correction (FEC), is a method of increasing the reliability of data communication. In one-way communication channels, once an error is detected, the receiver is not able to request retransmission. FEC is a method of transmitting redundant information along with the data so that when errors occur during transmission, the receiver can use the redundant information to reconstruct the data.

In this experiment, we will try three types of forward error correction codes, and clarify that forward error correction can indeed correct errors by observing the difference in bit error rate between using forward error correction codes and not using them.

The three kinds of FEC code are respectively BCH, Convolutional, and Concatenated code.

BCH Code

BCH codes are a class of the most important cyclic codes that can correct multiple random errors. Parity check codes can only detect errors but cannot correct them. And BCH code is one of the cyclic codes.

In the previous discussion, all we did was to construct a code and then calculate its minimum distance to estimate its error-correcting capability. However, in BCH codes, we will use another method: first specify the number of errors we want it to correct, and then construct the code accordingly.

To correct t errors, it is required that the minimum code distance d_0 is less than or equal to 2t+1.

What I did

In this experiment, I used four different BCH codes. They are (7,4) BCH code, (15,5) BCH code, (31,6) BCH code, (63,10) BCH code.

The entire process should be as follows: first, use BCH to encode the image data, then add noise, then use QPSK to modulate the signal, then use QPSK to demodulate the signal, and finally decode the signal using BCH. The SNR range I chose is from -3 to 9. Compare the bit error rate (BER) of the BCH-encoded data with the BER of non-BCH-encoded data, and plot the results.

The Results

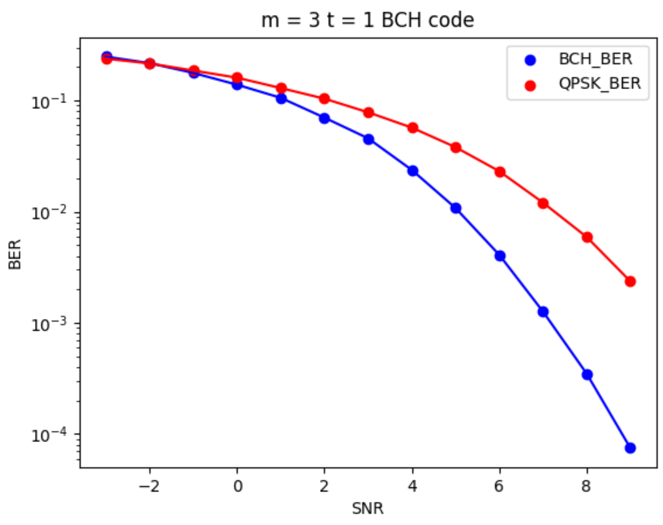

Figure 1.1: (7,4) Code BER of BCH versus BER of non-BCH

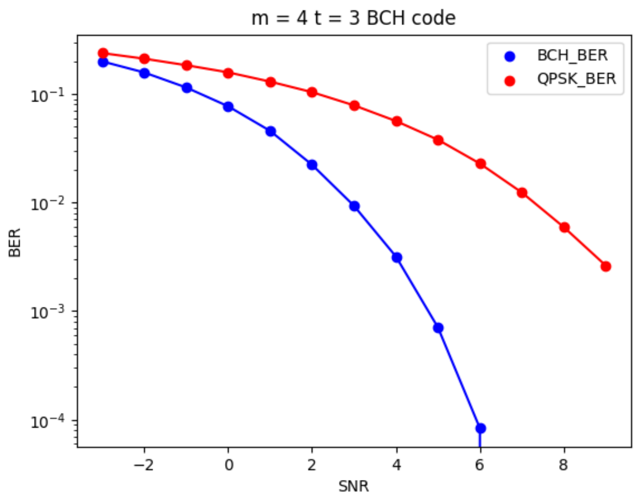

Figure 1.2: (15,5) Code BER of BCH versus BER of non-BCH

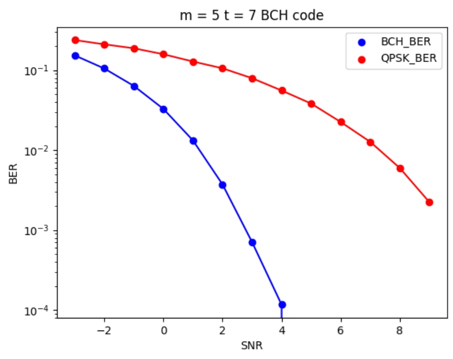

Figure 1.3: (31,6) Code BER of BCH versus BER of non-BCH

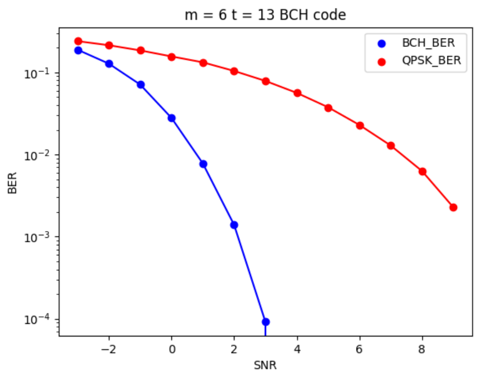

Figure 1.4: (7,4) Code BER of BCH versus BER of non-BCH

Conclusion

1. From Figures 1.1, 1.2, 1.3, and 1.4, we can see that the bit error rate of the modulated signal using BCH encoding is significantly lower than that of the modulated signal without BCH encoding.

2. Moreover, the advantage of BCH codes is particularly evident when the signal-to-noise ratio is high.

3. The more parity check bits a BCH code has, the higher its error correction efficiency and the lower its bit error rate.

Convolutional Code

An error-correcting code called a convolutional code generates an output codeword by convolutions the input data with a fixed convolutional code (a set of shift registers and some fixed-weighted connections between them). Communication systems often use convolutional codes because they can increase system dependability while preserving the same data transmission velocity. Wireless communications, for example, can benefit from the use of convolutional codes since they can identify and repair a certain number of faults.

What I did

The transfer function matrix I chose is [[0o5,0o7]]. This means that each input produces two outputs, so the encoded data will be twice as long as the original data.

The entire process is as follows: first, use convolutional coding to encode the original data, then use QPSK to modulate the signal, then add Gaussian white noise, then use QPSK to demodulate the signal, and finally use convolutional decoding to decode the demodulated signal.

Then, plot the bit error rate (BER) of QPSK signals with and without convolutional coding as a function of SNR, with SNR on the x-axis and BER on the y-axis.

Results

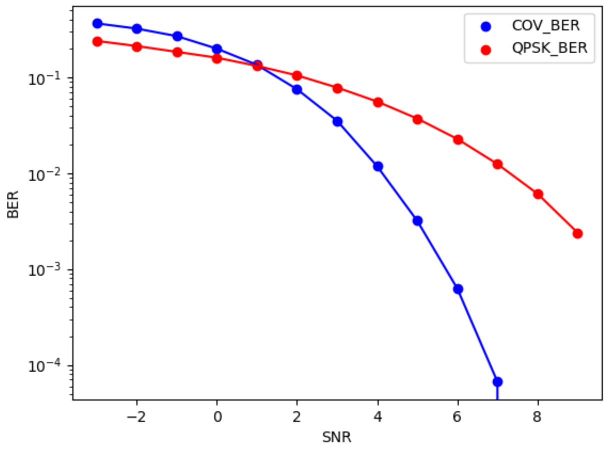

Figure 2.1: [[0o5,0o7]] Convolutional Code

Conclusion

At very low signal-to-noise ratios, the error rate of convolutional codes can be high, even higher than that of not using convolutional codes. This is because of the following reasons:

The coding scheme of convolutional codes results in data that is longer than the original data, making it more susceptible to bit errors in low signal-to-noise ratio situations, as the received noise can interfere with more bits.

The shift registers in convolutional codes have a delay effect, which means that erroneous bits can be retained and accumulated in the decoder, resulting in a higher decoding error rate.

The decoder of convolutional codes uses iterative decoding algorithms, which require multiple iterations to ultimately determine the value of each bit. Therefore, at very low signal-to-noise ratios, the decoder may fail to correctly identify and correct erroneous bits.

Concatenated Code

Concatenated coding refers to using two coding methods in sequence. Firstly, the original data is encoded using BCH coding to obtain encoded data A. Then, A is encoded using convolutional coding to obtain B. B is decoded using convolutional decoding to obtain A, which is finally decoded using BCH decoding to obtain the original data.

What I did

I misunderstood the experiment content when doing this cascade code experiment. My image is a comparison between the image with cascade code and the BER without cascade code.

The results

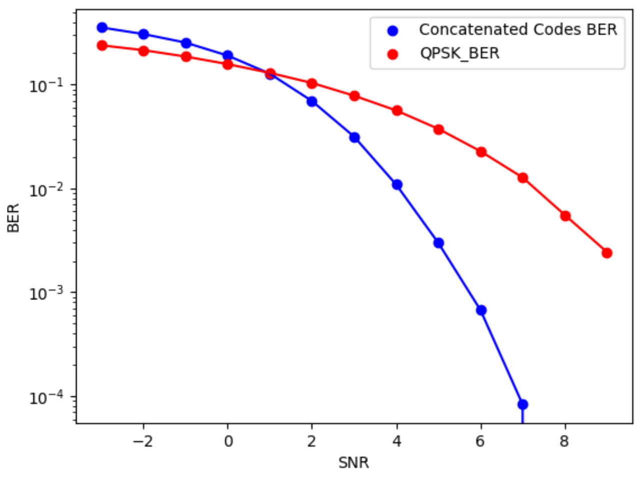

Figure 3.1: BER comparison chart of concatenated codes.

Conclusion

I think the error-correction efficiency of concatenated codes is better than pure convolutional coding, but obviously lower than BCH coding. I think this may be due to the small amount of data.

Conclusion

1. In the convolutional coding, I initially used cov code to encode and decode the data at once, but I found it slow and error-prone. So I divided the data into groups of 8 and encoded them separately. This made the code run much faster, but it would be very cumbersome in the case of cascade coding and involve many array dimension transformations. In the end, I found that the backtracking depth was the cause of my slow convolutional decoding. Therefore, I decided to modulate all data at once using cov code.

2. I misunderstood the experiment content of the cascade coding, which led to my failure to complete the experiment of cascade coding.

3. The more parity bits used in BCH encoding, the higher the error correction efficiency, but this also means a lower code rate.

4. The code rate refers to the ratio of the number of output bits to the number of input bits after encoding, i.e., the number of output bits corresponding to each input bit. Therefore, a low code rate means that the encoded data is longer than the original data, requiring more transmission time and bandwidth. Moreover, as the number of bits to be transmitted increases, the error rate during transmission also increases, requiring stronger error correction capability. A low code rate can also lead to an increase in transmission delay, which may affect real-time requirements.

Appendix

```
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
import komm
from scipy import special
tx_im = Image.open("DC4_150x100.pgm")
Npixels = tx_im.size[1]*tx_im.size[0]
# print(np.shape(tx_im))
# print(Npixels)
plt.figure()
plt.imshow(np.array(tx_im),cmap="gray",vmin=0,vmax=255)
plt.show()
tx_bin = np.unpackbits(np.array(tx_im))
print(tx_bin.shape)
print(tx_bin.dtype)
print(tx_bin)

# find total error bits
def error_Bits(array1,array2):
    cmp_cnt = np.sum([p[0] != p[1] for p in zip(array1, array2)])
#     print("Error bits",cmp_cnt)
    return cmp_cnt

# show Pics
def show_Pic(array):
    rx_im = np.packbits(array).reshape(tx_im.size[1],tx_im.size[0])
    plt.imshow(np.array(rx_im),cmap="gray",vmin=0,vmax=255)
    plt.show()
    return 0
# show_Pic(tx_bin)

# 得到分解后的二维数组,列数要与 k 相同才能正常使用
def slice_Array(arrayToBeSliced,k):
    x_2d = arrayToBeSliced.reshape(-1, k)  # 将一维数组重塑为二维数组, 每行
有 4 个元素
    return x_2d

#paras m t the    original array
def BCH_Code(m,t,arrayToBeSliced):
    code = komm.BCHCode(m,t)
    n,k = code.length, code.dimension
#     print(n,k)
```

```
    encoded_Code = np.zeros((int(120000/k), n), dtype=np.int32)
#     print(encoded_Code.shape)

    org_Code = np.copy(slice_Array(tx_bin,k))
    loop_Times = int(120000/k)
#     print(org_Code[0])
    for i in range(loop_Times):
        code_word =  code.encode(org_Code[i])
        encoded_Code[i] = code_word
    return encoded_Code

# paras m t     the encoded_array
def BCH_Decode(m,t,array):
    decoded_Code = np.array([], dtype=np.int32)  # 创建一个空的整数类型数
组
    code = komm.BCHCode(m,t)
    n,k = code.length, code.dimension
#     print(n,k)

    loop_Times = int(120000/k)
    print(loop_Times)
    for i in range(loop_Times):
        message_word = code.decode(array[i])
        decoded_Code = np.append(decoded_Code,message_word)
        print(i,end = "\r")
    return decoded_Code

# get total bit errors of QPSK
def QPSK_bit_Errors(SNR):
    psk = komm.PSKModulation(4)
    awgn = komm.AWGNChannel(snr=10**(SNR/10.))
    tx_data = psk.modulate(tx_bin)
    rx_data = awgn(tx_data)
    rx_bin = psk.demodulate(rx_data)
    cmp_cnt = np.sum([p[0] != p[1] for p in zip(tx_bin, rx_bin)])
    return (cmp_cnt/120000.0)

# find the total error bit
# parameters m t and SNR can be changed
def BCH_Ber_Cal(m,t,SNR):
    encoded_Code = BCH_Code(m,t,tx_bin)
    n = np.power(2, m) - 1
    psk = komm.PSKModulation(4)
    awgn = komm.AWGNChannel(snr=10**(SNR/10.))
```

```python
    encoded_Code_1d = encoded_Code.ravel()

    tx_data = psk.modulate(encoded_Code_1d)
    rx_data = awgn(tx_data)
    rx_bin = psk.demodulate(rx_data)

    arr_n = rx_bin.reshape((-1, n))
    decoded_Code = np.copy(BCH_Decode(m,t,arr_n))
#     print("total error bits",error_Bits(tx_bin,decoded_Code))
#     show_Pic(decoded_Code)
    return (error_Bits(tx_bin,decoded_Code)/120000.)

# QPSK_BER = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,
10.0, 11.0, 12.0, 13.0])
# BCH_BER = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,
10.0, 11.0, 12.0, 13.0])
# SNR_array = np.array([-3,-2,-1,0,1,2,3,4,5,6,7,8,9])

QPSK_BER  =  np.arange(1, 14, dtype=float)
BCH_BER   =  np.arange(1, 14, dtype=float)
COV_BER_N   =  np.arange(1, 14, dtype=float)
BOTH_BER   =  np.arange(1, 14, dtype=float)
SNR_array = np.arange(-3, 10, dtype=np.int32)

def draw_Pics(m,t):
    print(m,t,"code")
    for i in range(-3,10):
        print(i,end = "\r")
        print(i,"dB")
        BCH_BER[i+3] = BCH_Ber_Cal(m,t,i)
        QPSK_BER[i+3] = QPSK_bit_Errors(i)

        print("BCH_BER",BCH_BER[i+3])
        print("QPSK_BER",QPSK_BER[i+3])

    fig, ax = plt.subplots()

    ax.scatter(SNR_array, BCH_BER, color='blue', label='BCH_BER')
    ax.plot(SNR_array, BCH_BER, color='blue')

    ax.scatter(SNR_array, QPSK_BER, color='red', label='QPSK_BER')
    ax.plot(SNR_array, QPSK_BER, color='red')
    ax.set_xlabel('SNR')
    ax.set_ylabel('BER')
    ax.legend()

    ax.set_yscale('log')
    ax.set_title("m = "+str(m)+" t = "+str(t)+" BCH code")
    plt.show()
    print("-----------------------------------")
draw_Pics(3,1)
draw_Pics(4,3)
draw_Pics(5,7)
draw_Pics(6,13)

# def cov_encode(message):
# print(message.size)

def COV_BER(tx_bin,SNR):
    code = komm.ConvolutionalCode(feedforward_polynomials=[[0o7, 0o5]])
    tblen = 18
    encoder = komm.ConvolutionalStreamEncoder(code, initial_state=0)
    encoded = encoder(tx_bin)

    psk = komm.PSKModulation(4)
    awgn = komm.AWGNChannel(snr=10**(SNR/10.))
    tx_data = psk.modulate(encoded)
    rx_data = awgn(tx_data)
    rx_bin = psk.demodulate(rx_data)


    decoder = komm.ConvolutionalStreamDecoder(code,
traceback_length=tblen, input_type="hard")
    decoded = decoder(np.append(rx_bin, np.zeros(2*tblen,
dtype=np.int32)))
    print("ok")
    decoded_final = decoded[tblen:]
    show_Pic(decoded_final)

    number = error_Bits(tx_bin,decoded_final)

    return (number/120000.)

# print(COV_BER(tx_bin,-3))
# print(COV_BER(tx_bin,-2))
# COV_BER(tx_bin,-1)
```

```python
# COV_BER(tx_bin,0)

def draw_Pics_Cov(tx_bin):
    for i in range(-3,10):
        print(i,end = "\r")
        COV_BER_N[i+3] = COV_BER(tx_bin,i)
        QPSK_BER[i+3] = QPSK_bit_Errors(i)
        print("COV_BER",COV_BER_N[i+3])
        print("QPSK_BER",QPSK_BER[i+3])

    fig, ax = plt.subplots()

    ax.scatter(SNR_array, COV_BER_N, color='blue', label='COV_BER')
    ax.plot(SNR_array, COV_BER_N, color='blue')

    ax.scatter(SNR_array, QPSK_BER, color='red', label='QPSK_BER')
    ax.plot(SNR_array, QPSK_BER, color='red')

    ax.set_xlabel('SNR')
    ax.set_ylabel('BER')
    ax.legend()

    ax.set_yscale('log')
    plt.show()
    print("-----------------------------------")
draw_Pics_Cov(tx_bin)
# print(COV)

# BOTH Code
def both_Code_BER(SNR):
    encoded_Code = BCH_Code(3,1,tx_bin)
    encoded_Code_1d = encoded_Code.ravel()

    code = komm.ConvolutionalCode(feedforward_polynomials=[[0o7, 0o5]])
    tblen = 18
    encoder = komm.ConvolutionalStreamEncoder(code, initial_state=0)
    encoded = encoder(encoded_Code_1d)

    psk = komm.PSKModulation(4)
    awgn = komm.AWGNChannel(snr=10**(SNR/10.))
    tx_data = psk.modulate(encoded)
    rx_data = awgn(tx_data)
    rx_bin = psk.demodulate(rx_data)
    decoder = komm.ConvolutionalStreamDecoder(code,
traceback_length=tblen, input_type="hard")
    decoded = decoder(np.append(rx_bin, np.zeros(2*tblen,
dtype=np.int32)))
    print("ok")
    decoded_final = decoded[tblen:]
    arr_n = decoded_final.reshape((-1, 7))

    final_Code = BCH_Decode(3,1,arr_n)
    numbers = error_Bits(final_Code,tx_bin)

    return (numbers/120000.)
#     show_Pic(final_Code)
# print(both_Code_BER(1))
# number = error_Bits(tx_bin,decoded_final)

# def
def draw_Both(m,t,array):
    for i in range(-3,10):
        print(i,end = "\r")
        BOTH_BER[i+3] = both_Code_BER(i)
        QPSK_BER[i+3] = QPSK_bit_Errors(i)
        print("COV_BER",COV_BER_N[i+3])
        print("QPSK_BER",QPSK_BER[i+3])

    fig, ax = plt.subplots()

    ax.scatter(SNR_array, BOTH_BER, color='blue', label='Concatenated
Codes BER')
    ax.plot(SNR_array, BOTH_BER, color='blue')

    ax.scatter(SNR_array, QPSK_BER, color='red', label='QPSK_BER')
    ax.plot(SNR_array, QPSK_BER, color='red')

    ax.set_xlabel('SNR')
    ax.set_ylabel('BER')
    ax.legend()

    ax.set_yscale('log')
    plt.show()
    print("-----------------------------------")
draw_Both(3,1,tx_bin) #
```