Project Evaluation From

Project Title: OFDM system construction and the performance of OFDM system on different channels, different modulation methods and different coding methods.

Name: Zhang Ziteng **Class: 1601019**

Criterion	Excellent	Good	Fair	Poor
Relevance and Appropriateness of the Topic				
Proposal(Task description and planning)				
Technical Quality				
Workload and Complexity				
Writing and Representation (including the language)				
Adequate Illustrations or Drawings				
Oral Representation and Question Answering				
Overall Rate	(Excellent -- Poor) 100 90 80 70 60 <60			

OFDM system construction and the performance of OFDM system on different channels, different modulation methods and different coding methods.

Zhang Ziteng 16010199044,

Class 1601019

Abstract: Orthogonal frequency division multiplexing (OFDM) is a multi-carrier parallel transmission technology which has the capability of anti-multipath interference. It is the key technology of the fourth-generation mobile communication system. This paper introduces OFDM system construction and the performance comparison of and different channels, different modulation modes and different coding modes in OFDM system.

Keywords: Orthogonal frequency division multiplexing (OFDM); computer simulation; MATLAB;16QAM;QPSK

1 introduction

OFDM is a frequency division multiplexed multi-carrier transmission method, and each multiplexed signal (each carrier) is orthogonal. This technology has been recognized by the industry as the core technology of the new generation of wireless mobile communication systems and has high research value. The OFDM technology converts high-speed data streams into multiple parallel low-speed data streams by serial/parallel conversion, and then distributes them to subchannels on mutually orthogonal subcarriers of different frequencies for transmission. The biggest advantage of OFDM is its ability to combat frequency selective fading or narrowband interference while maintaining high spectrum utilization. This paper will discuss OFDM system construction and the performance of OFDM system on different channels, different modulation methods and different coding methods.

2 Principle of OFDM

The main idea of OFDM is to divide a channel into N subchannels, one carrier on each subchannel, called a subcarrier, and each subcarrier is orthogonal to each other. In implementation, a high-speed serial input data signal stream is converted into N parallel low-speed sub-data streams, and modulated onto each sub-carrier for transmission. After serial/parallel conversion, N parallel data are simultaneously output and modulated on N subcarriers. No. N subcarriers can be expressed as $\boldsymbol{f_n = f_c + n/T_S}$ .Then an OFDM signal at the m th moment can be expressed as

$$\boldsymbol{S_m(t) = \mathrm{Re}\{\sum_{n=0}^{N-1} d(n) e^{j2\pi f_n t}\}, 0 < t < T}$$

Since OFDM is emitted after the serial-parallel conversion of N symbols, the symbol rate is the original OFDM symbol rate1/N, so $\boldsymbol{T_s = \mathrm{N}T_s}$.When the guard interval between OFDM symbols is not considered,

$$\boldsymbol{S_m(t) = \mathrm{Re}\{\sum_{n=0}^{N-1} d(n) e^{j2\pi f_n t}\}}$$

$$\boldsymbol{= Re\left\{\sum_{n=0}^{N-1} d(n) e^{j2\pi \frac{n}{NT_s} t}\, e^{j2\pi f_c t}\right\}}$$

$$\boldsymbol{= Re\{X(t) e^{j2\pi f_c t}\}, 0 < t < T}$$

Where, $\mathbf{X(t)} = \boldsymbol{\sum_{n=0}^{N-1} d(n) e^{j2\pi \frac{n}{NT_s} t}}$.$\mathbf{X(t)}$ is the complex equivalent baseband of the transmitted signal signal. Sampling $\mathbf{X(t)}$ with a sampling rate of $1/T_s$,then when $\mathrm{k} = \mathbf{k}\boldsymbol{T_s}$, the sampled value $\mathbf{X(k)}$ satisfies $\mathbf{X(k)} = \mathbf{X(k}\boldsymbol{T_s}) = \boldsymbol{\sum_{n=0}^{N-1} d(n) e^{j\frac{2\pi}{NT_s} nk}}$. $\mathbf{X(k)}$ is exactly the result of the N-point inverse discrete Fourier transform (IDFT) of d(n). In practical applications, we can use the IFFT operation to complete the subcarrier modulation process of the OFDM complex equivalent baseband signal, and complete the demodulation process with Fast Fourier Transform (FFT).

3 OFDM system structure

The structure of an OFDM system is shown in Figure 1. The OFDM system consists of input data, channel coding, constellation mapping, training sequence insertion, serial-to-parallel conversion, IFFT, serial-to-parallel conversion, cyclic prefix, channel, de-cyclic prefix, and serial-to-parallel conversion**,** FFT, parallel-serial conversion, channel estimation, demodulation, channel decoding, the data output.

3.1 Input data

Input data is the binary data that source generates to be transmitted.In MATLAB, we use the *randi([0,1],1, length)* function to generate random numbers between 0and1 as the input data.

3.2 Channel coding and decoding

Source coding is a kind of transformation of source symbols for the purpose of improving communication effectiveness. Here we use RS coding and concatenated coding with RS (reed Solomon) coding and convolutional coding and compare the performance of these two coding methods.

RS encoding is an encoding method based on a finite field, finite field element by which is transformed configured generator polynomial g (x), and so that the codeword polynomial calculated for each information segment is g (x) of times. The RS code generator polynomial is generally selected as follows: $g(x) = (x-a)(x-a^2)\dots(x-a^{2t}) = \prod_{i=1}^{2t}(x-a^i)$, Where a^i is an element in $gf(2^m)$. If d(x) is used to represent the information segment polynomial, the codeword polynomial c(x) can be constructed as follows. First calculate the business formula h(x) and the remainder: r(x)： $\frac{x^{n-k}d(x)}{g(x)} = h(x)g(x) + r(x)$，Take the remainder r(x) as the check word; Then let $c(x) = x^{n-k}d(x) + r(x)$, that is, place the information bits in the first half of the codeword, and supervise the bitwise manner in the second half of the codeword then we can get$\frac{c(x)}{g(x)} = \frac{x^{n-k}d(x)}{g(x)} + r(x) + r(x) = h(x)g(x)$. Therefore, the codeword polynomial c(x) must be divisible by the generator polynomial g(x). If the receiver detects that the remainder is not 0, it can be judged that there is an error in the received codeword. Since this RS code can correct t m-ary error code words, the RS code is particularly suitable for channels with burst errors. Here we use the RS code of 5 input 7 output. In MATLAB we use the gf() function to convert the input signal into a Galois field function, and then use *rsenc(data,7,5)*to generate a 11-input 15-output source RS code. We use *rsdec(gf(yrsgs41,4), nn, kk)* to decode the demodulated data.

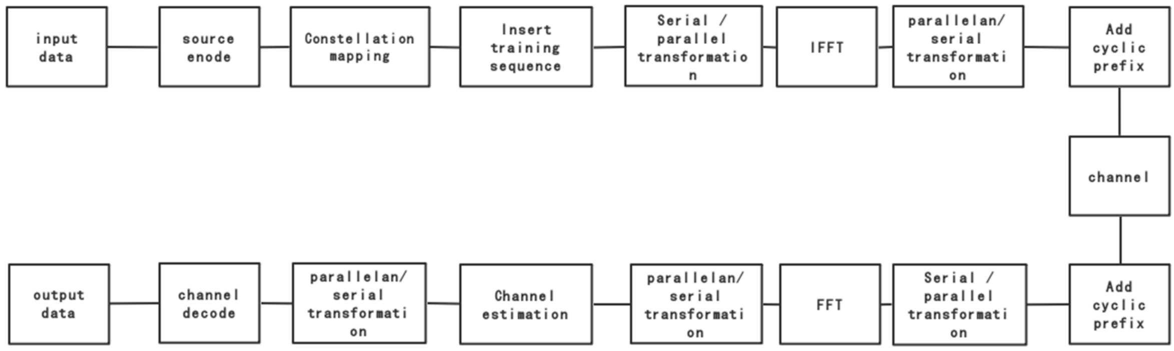


Figure 1 OFDM system structure

The convolutional code is an error control code, and the convolutional code is represented by (n, k, L), where n is the number of output bits, k is the number of bits input, and L is the constraint length. As a memory error correcting code, the encoding rule is to encode k input bits into n output bits, and the encoded symbols are related not only to the input k bits but also to the previous L-1 group bits. Here we use a 1/2 feedback convolutional encoder (2, 1, 7) with a constraint length of 7.

For convolutional codes, we can use Viterbi decoding to decode. This decoding method obtains a decoded codeword closest to the encoded codeword based on the information that has been received. The coding criterion used to obtain such a codeword is maximum likelihood decoding.

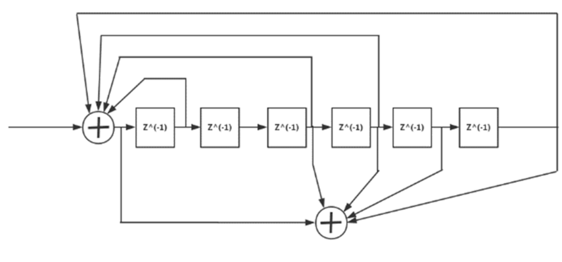


Figure 2 convolutional codes generator

In MATLAB, we use *trellis = poly2trellis(7,[133 171])* which is in Figure 2, This encoder has a constraint length of 7, a generator polynomial matrix of [133 171].The first generator polynomial matches the feedback connection polynomial because the first output corresponds to the systematic bits. The feedback polynomial is represented by the binary vector [1 1 1 1 0 1 1], corresponding to the upper row of binary digits in the diagram. These digits indicate connections from the outputs of the registers to the adder. The initial 1 corresponds to the input bit. The octal representation of the binary number 1111011 is 171.

The second generator polynomial is represented by the binary vector [1 0 1 1 0 1 1], corresponding to the lower row of binary digits in the diagram. The octal number corresponding to the binary number 1011011 is 137.We use this function to generate a convolutional code with a constraint length of 7, and encode the source information by the function *source_coded_data=convenc(inforSource, trellis)*.We use the *vitdec (De_Bit, trellis, 42, 'trunc', 'hard')* function to decode the demodulated data.

For concatenated coding, we can consider the coding, channel, and decoding as a generalized channel. This channel also has errors, so it can be further error-corrected. When two concatenation codes are concatenated to form one concatenated code, the coding in the generalized channel is called an inner code, and the channel coding in which the generalized channel is a channel is called an outer code. Here we think that the convolutional code is the inner code and the RS code is the outer code.

3.3 Constellation mapping and demodulation

The digital baseband signal can be written as a result of weighted accumulation of different basis functions. The coefficient $\{s_{ij}\}$ representing the signal $S_i(t)$ is written as a vector $s_i = \{S_{i1}, S_{i2} \dots S_{iN}\}^T \in \mathbb{R}^N$, which is referred to as a signal constellation point of $S_i(t)$. All signal

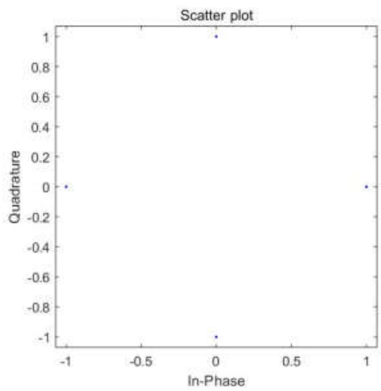
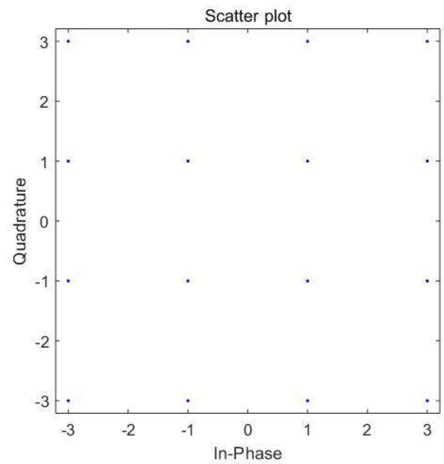


Figure 3 constellation point of QPSK

constellation points $\{S_1, S_2 \dots S_M\}$ constitute a signal constellation. For a given base signal $\{\phi_1(t), \phi_2(t) \dots \phi_N(t)\}$, the signal $S_i(t)$ has a one-to-one correspondence with its constellation point S_i. QPSK carries information through the phase. Send signal

$$S_i(t) = Ag(t)\cos\left[\frac{2\pi(i-1)}{4}\right] cos2\pi f_c t - Ag(t)\sin\left[\frac{2\pi(i-1)}{4}\right] sin2\pi f_c t$$

The constellation points (S_{i1}, S_{i2}) are given by $S_{i1} = A\cos\left[\frac{2\pi(i-1)}{4}\right]$ and $S_{i2} = A\sin\left[\frac{2\pi(i-1)}{4}\right]$ $i = 1 \dots 4$. Constellation diagram is shown in Figure 3.
16QAM carries information in both amplitude and phase.



The transmission signal of 16QAM is $S_i(t) =$

Figure 4 constellation point of 16QAM

$A_i\cos(\theta_i)\, g(t)\cos(2\pi f_c t) - A_i \sin(\theta_i) g(t)\sin(2\pi f_c t)$ $0 \le t \le T$. For a square constellation, S_{i1} and S_{i2} are at $(2\mathrm{i}-1-4)\mathrm{d}$, $\mathrm{i} = 1 \dots 4$. Constellation diagram is shown in Figure 4.

3.4 Serial to parallel transformation

Since the data generated by the source needs to be modulated onto different carriers, and the previous analysis can be used to perform OFDM modulation by the IFFT method, it is necessary to convert the original serially transmitted data into N long parallel data for IFFT

transmit. When the signal is transmitted in the channel, it is transmitted in serial form, so the parallel to serial conversion is performed later.

3.5 Cyclic prefix

Due to the multipath phenomenon in the actual wireless channel, a bundle of signals arrives at the receiving terminal through multiple paths from the transmitting terminal, and the delays of signal propagation are different due to different distances of multiple paths. Signals with different delays are superimposed to cause mutual interference between symbols.

In order to solve the problem of interference between codes caused by multipath, we can increase the symbol time. When the symbol time is much larger than the time delay of the channel, the influence of interference between codes on the symbol decision will be greatly reduced, but the result is a reduction in the transmission speed of the symbols. In order to effectively reduce the influence of inter-code crosstalk while minimizing the impact on the transmission rate of the symbol itself, we can insert a guard interval between each OFDM symbol, and the guard interval length Ts is generally larger than the maximum of the radio channel. The delay spreads so that the multipath component of one symbol does not interfere with the next symbol.

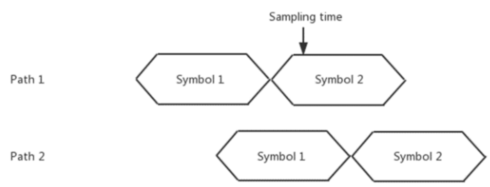


Figure 3 mutual interference between symbols

During this guard interval may be inserted without any signal, that is, an idle transmission period. However, in this case, due to the influence of multipath propagation, adding a guard interval of 0 will cause the waveforms in the integration interval to be discontinuous, destroying the orthogonality between the subcarriers, and inter-channel interference will occur. At this time, the result of the sampling time is not affected by the symbol symbols of other paths, but interference occurs between different subcarriers.

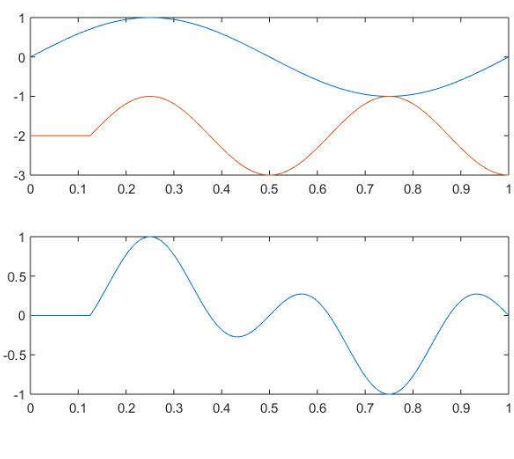


Figure 4 the orthogonality destory between the subcarriers

To resolve inter-subcarrier interference, a cyclic prefix needs to be added within the guard interval. The signal in the cyclic prefix is the same as the portion of the OFDM symbol tail that is the cyclic prefix length, and the length is greater than the maximum delay of the channel. After doing this, for each subcarrier, the waveform at the guard interval becomes continuous, that is, the waveform of each subcarrier is continuous during the whole period of the guard interval plus the symbol duration. In the interval, the two waveforms are multiplied and then the result of integration is 0, and the orthogonality of the subcarriers is restored.In an actual system, before the OFDM symbol is sent to the channel, the cyclic prefix is first added and then sent to the channel for transmission. At the receiving terminal, the cyclic prefix portion at the beginning of the received symbol is first discarded, and then the remaining portion of the width $\mathrm{N}T_s$ is subjected to Fourier transform and then demodulated. Since the length of the cyclic prefix is greater than the maximum delay of the channel, the delay signal with a delay less than the guard interval will not generate ICI during the demodulation process.

3.6 IFFT and FFT

As can be seen from the second section, we can OFDM the baseband symbols by IFFT and demodulate the OFDM signals by FFT.

3.7 Inserting pilot and channel estimation

Due to the randomness of channel noise and the influence of channel multipath, in order to obtain the original data stream, we need to estimate the channel at the receiving terminal to obtain the reference phase and amplitude on each subcarrier of the OFDM symbol, so as to recover raw data bits without error. The accuracy of the channel estimation directly affects the performance of the entire OFDM system. There are two common methods of channel estimation: channel estimation based on pilot information and blind channel estimation based on cyclic prefix. Here we use channel estimation based on pilot information. The insertion pilot method uses the pilot to obtain the channel information of the pilot position, and then obtains the channel information of the entire data transmission by the information of the pilot portion. Several commonly used algorithms for channel estimation of pilot position in OFDM systems are forced zero estimation (ZF), maximum likelihood estimation (ML), least squares estimation (LS), minimum mean square error estimation (MMSE), pilot position's channel information is the ratio of the signal received by the receiving termianl to the signal sent by the transmitting terminal. Here we use least squares estimation (LS).

Assume that the channel model is $\mathrm{Y} = \mathrm{XH} + \mathrm{W}$, where Y is the received signal and X is the known pilot transmit signal. H is channel information and W is channel noise. The least squares estimation is to estimate the channel information H such that $\mathrm{J} = (\mathrm{Y} - Y_e)^H(\mathrm{Y} - Y_e) = (Y - X_e H_e)^H (Y - X_e H_e)$ is the smallest, where Y_e is the estimated received pilot , H_e is the estimated pilot information, and X_e is the estimated transmit pilot. The derivation gives $\mathrm{H} = X^{-1}Y$. It can be seen that the LS estimation only needs to know the transmission signal X, observe the noise W, and other statistical features of the received signal Y for the parameter H to be determined without other information, so the biggest advantage of the LS channel estimation algorithm is that the structure is simple and the calculation amount Small, the channel characteristics of the pilot position subcarriers can be obtained only by performing a division operation on each carrier. In MATLAB, we use the *commsrc.pn ('GenPoly', [1 0 0 0 0 1 1],'NumBitsOut', pilot_num*pilot_len)* to generate the pn code. We use pn code as the piloted carrier for the cross-correlation between any two pn code sequences is 0. The Pn sequence is a pseudo-random sequence, and the autocorrelation value of the pn sequence is 1. When the period of the pn sequence is sufficiently large, the cross-correlation of any two pn code sequences is almost 0. Using this property, we join at the transmitting end. The pn code sequence is used as a training sequence.

3.8 Channel

The AWGN channel is a Gaussian additive white noise channel. The Gaussian additive white noise means that the noise of the channel is evenly distributed in the spectrum, and the amplitude is normally distributed. The noise of the AWGN channel is superimposed on the transmitted signal. The Rayleigh fading channel is a statistical model of the radio signal propagation environment. Due to noise, multipath problems and shadowing in the channel, the signal amplitude is random, i.e. "fading", and its envelope obeys the Rayleigh distribution. Here, we use *awgn(tx_signal_serial, SNR(a), 'measured')* to simulate the AWGN channel, using *chan=comm.RayleighChannel('SampleRate',550000, ... 'PathDelays',[0 2e-6],'AveragePathGains',[0 -3],'MaximumDopplerShift',100);* to simulate the Rayleigh channel. It is assumed here that there are two paths, one direct path and one indirect paths in the Rayleigh channel, and the delays are 2*10^ (-6) seconds, the path loss is -3db.

4 Result analysis

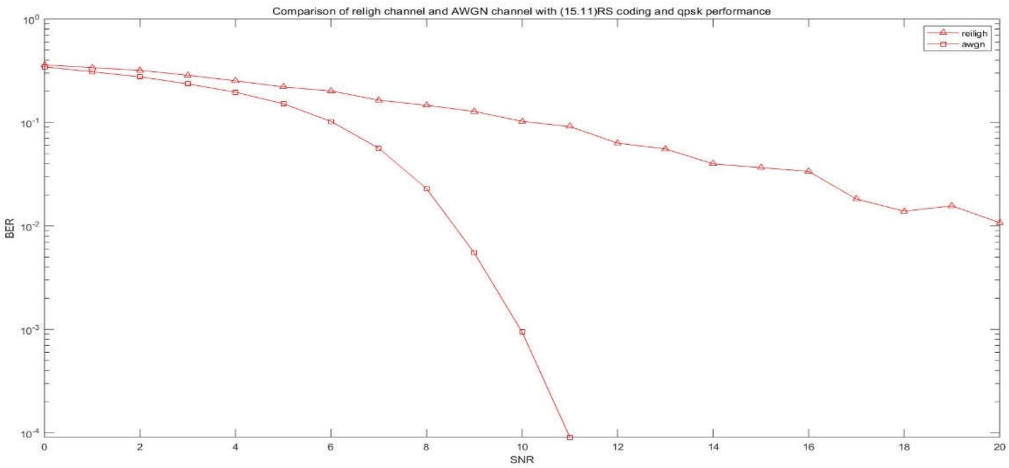


Figure 5 Comparison of AWGN channel and Rayleigh channel performance

4.1 Comparison of AWGN channel and Rayleigh channel performance

The comparison of AWGN channel and Rayleigh channel performance is shown in the Figure5. We adopt the QPSK modulation method, and the channel coding adopts (15,11)RS coding. We pass the data signal through the AWGN channel and use the *biterr (data, old_data)* function to compare the output data with the original data to calculate the bit error rate. Similarly, we pass the data signal through the Rayleigh channel and then pass the AWGN channel to imitate the real Rayleigh channel, and then calculate the bit error rate in the same way as before. From the Figure we can see that When the signal-to-noise ratio is small, the error rate of the AWGN channel and the Rayleigh channel are not much different, and when the signal-to-noise ratio is relatively large, the performance of the AWGN channel is better than that of the Rayleigh channel. This is because in the Rayleigh channel, there is not only the influence of noise, but also the effects of multipath and shadow. When the signal-to-noise ratio is large enough, the AWGN channel error rate tends to be zero, and the Rayleigh channel's bit error rate fluctuates within a certain range. Therefore, when the signal-to-noise ratio is sufficiently large, the influence of AWGN channel noise on the signal can be ignored. Except for the interference of the signal in the Rayleigh channel, it is mainly related to multipath and has little relationship with noise.

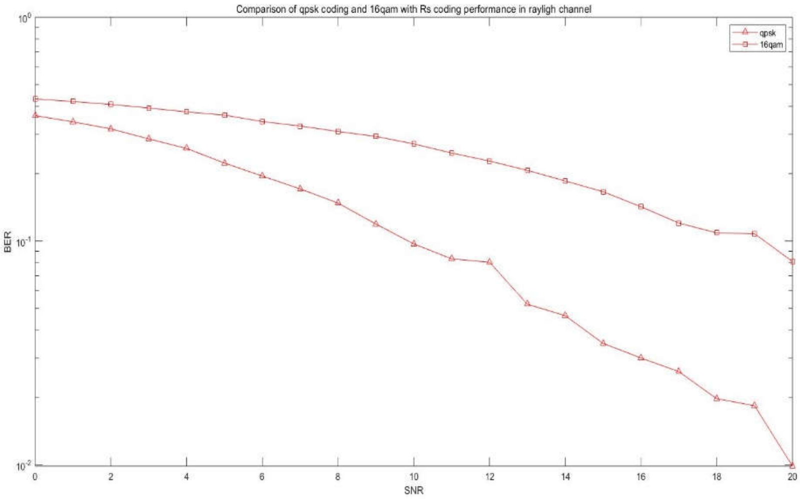


Figure 6 Comparison of QPSK and 16QAM performance

4.2 Comparison of QPSK and 16QAM performance

In the case of Rayleigh channel, the coding mode is rs coding, the performance of QPSK is much larger than that of 16QAM. Because when the signal-to-noise ratio is the same, that is, the power of the signal is the same, the larger the constellation is, the energy allocated to each constellation point is small, and the anti-interference is poor. In practice, the quality of the channel is better than the simulation. The transmission rate of 16QAM is higher than QPSK, so we actually use 16QAM.

4.3 Comparison of RS coding and RS with Convolutional coding performance

The comparison of RS coding and RS with Convolutional coding performance is shown in Figure11.

When the signal-to-noise ratio is small, the RS coding and convolutional code concatenated coding performance is worse than the RS coding only. This is because the concatenated coding has a threshold effect. When the signal-to-noise ratio is small, the concatenation is due to the principle of information non-increasing. The performance improvement of the code is less than the loss of information caused by system cascading, resulting in the performance of the concatenated code is not as good as a single code. When the signal-to-noise ratio is increased, the concatenated coding of the RS coding and the convolutional code is better than the performance of only the convolutional code coding. The concatenated code is more reliable than the single RS coding, and is more adaptable to the channel with stronger interference.

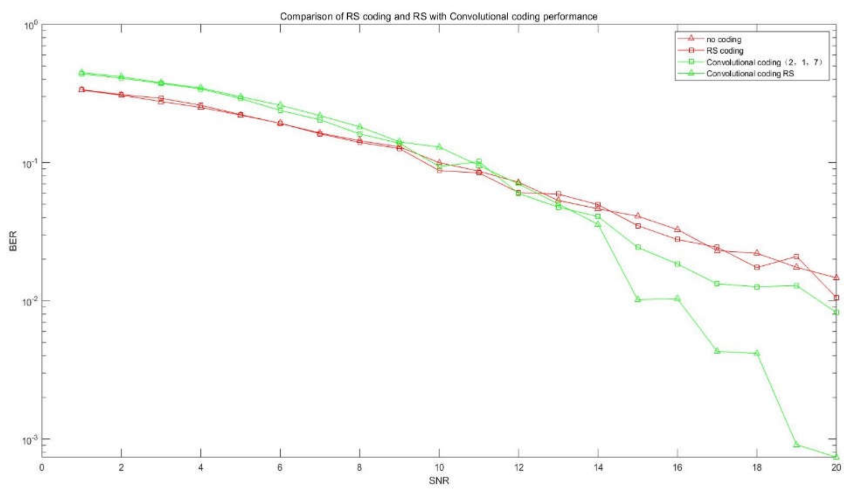


Figure 7 Comparison of different coding performance

The limited number of simulation points and the influence of the multipath of the Rayleigh channel leads to the last part of the waveform fluctuates.

5 Reference

[1] 巩林玉,陈长兴,金贵斌,刘保军,GONG Lin-yu,CHEN Chang-xing,JIN Gui-bin,LIU Bao-jun- 《邮电设计技术》2009 年 4 期

[2]马一森 李 智 赵建华 徐 炜 张 东.OFDM 通信系统的分析及其 Simulink 仿真研究[J].计算机与数字工程,:,2010.11(2):137-139

6 Code

6.1 Code of comparing of AWGN channel and Rayleigh channel performance

```
function
[error_bit_all,error_symbol_all]=rs_awgn_coding()
sonCarrierNum_temp=88;
symbols_Per_Carrier=1000;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=2;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));%
FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例
1/6~1/4
pilot_Inter=1;%插入导频间隔
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM
符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
kk=11;
%-------------------------------信源输入----------------------
----------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbol
s_Per_Carrier*bits_Per_Symbol);
%---------------------------信道编码---------------------------
----------------------
msg4_temp=reshape(inforSource,4,[])';
msg4=bi2de(msg4_temp,'left-msb');%将原来的数据转
换为 4 位 16 进制
msg4_togf=reshape(msg4,kk,[]).'; %带转换的矩阵，十
一输入
msgGF=gf(msg4_togf,4);%转换为伽罗华域
msgrs=rsenc(msgGF,nn,kk); %(15,11）RS 编码 11 个输
入 15 个输出
msgrs1=reshape(msgrs.',1,length(msg4)/kk*nn);% 将 rs
编码输出转成一行
msgrs2=de2bi(double(msgrs1.x),'left-msb');%十进制转
二进制
source_coded_data_rs=reshape(msgrs2',1,length(msg4)/k
k*nn*4);%待调制信号 输出一行信号（数据）
%----------------------------调制--------------------------------
---------------------
data_temp1=
reshape(source_coded_data_rs,modulate_bit,[])'; %以
每组 2 比特进行分组，输出两列数据
modulate_data=pskmod(bi2de(data_temp1),2^(modulate
_bit),pi/4);%输出一列数据
%-------------------------插入导频-----------------------------
-----------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,mo
dulate_length);modulate_data(31:60,:)];%在原来输出数
据的中间插 0
h1=commsrc.pn('GenPoly', [1 0 0 0 0 1
1],'NumBitsOut',61*modulate_length,'InitialConditions',[
0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
    modulate_data_pn(:,(2*i-
1))=modulate_data_temp(:,i);
    modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);mod
ulate_data_pn(1:30,:)];
%-----------------------------------ifft----------------------------
---------
time_signal_ifft=ifft(modulate_data_pn_out);
%-----------------------------------cp-----------------------------
----------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_if
ft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
[time_signal_cp_wide,time_signal_cp_length]=size(time
_signal_cp);
%-------------------------------------并串变换-----------------
-------------------
for ii=1:modulate_length
    time_signal_out(:,ii)=[time_signal_cp(:,2*ii-
1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
for a=1:21
% ----------------------------------信道------------------------
---------------
% chan=comm.RayleighChannel('SampleRate',550000, ...
%         'PathDelays',[0 2e-6],'AveragePathGains',[0 -
3],'MaximumDopplerShift',100,'RandomStream','mt1993
7ar with seed','Seed',8007);
% chan=comm.RayleighChannel('SampleRate',550000, ...
%         'PathDelays',[0 2e-6],'AveragePathGains',[0 -
3],'MaximumDopplerShift',100);
```

```
% Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn(   time_signal_out_1,a-1,'measured');%添加高斯白噪声
%     awgn_signal=time_signal_out_1;
%-----------------------------------串并转换---------------------------------
    receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%-----------------------------------去循环前缀---------------------------------
    receive_data=receive_signal_perallel(27:90,:);
%----------------------------------fft--------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%--------------------------------信道估计---------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length

channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
      estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%--------------------------------------解调---------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=pskdemod(demodulate_data_temp,2^(modulate_bit),pi/4);
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
%------------------------------------------译码-----------------------------------
[real_data_temp1_wide,real_data_length]=size(real_data_temp1);
yrsgs4=reshape(real_data_temp1  ,4,real_data_temp1_wide*real_data_length/4).';
yrsgs41=bi2de(yrsgs4,'left-msb');
yrsgs41=reshape(yrsgs41,nn,length(yrsgs41)/nn).';
ygsrsdecode=rsdec(gf(yrsgs41,4),nn,kk);
d1=reshape(ygsrsdecode.x',1,[]);
d2=de2bi(d1,'left-msb').';
rx_decode=reshape(d2,1,[]);
%---------------------------------------------误码率----------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%---------------------------------------------误符号率-------------------------------
error_symbol_data1=
reshape(rx_decode,modulate_bit,[])';    %以每组2比特进行分组，输出两列数据
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2=
reshape(inforSource,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
end
%---------------------------------------------------------------------
function
[error_bit_all,error_symbol_all]=rs_reiligh_coding()
sonCarrierNum_temp=88;
symbols_Per_Carrier=1000;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=2;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));%FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例 1/6~1/4
pilot_Inter=1;%插入导频间隔
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM 符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
kk=11;
%------------------------------信源输入-------------------------------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbols_Per_Carrier*bits_Per_Symbol);
%-------------------------信道编码---------------------------------------------
msg4_temp=reshape(inforSource,4,[])';
msg4=bi2de(msg4_temp,'left-msb');%将原来的数据转换为 4 位 16 进制
msg4_togf=reshape(msg4,kk,[]).'; %带转换的矩阵，十一输入
msgGF=gf(msg4_togf,4);%转换为伽罗华域
```

```
msgrs=rsenc(msgGF,nn,kk); %(15,11）RS 编码 11 个输入 15 个输出
msgrs1=reshape(msgrs.',1,length(msg4)/kk*nn);% 将 rs 编码输出转成一行
msgrs2=de2bi(double(msgrs1.x),'left-msb');% 十进制转二进制
source_coded_data_rs=reshape(msgrs2',1,length(msg4)/kk*nn*4);%待调制信号 输出一行信号（数据）
%----------------------------调制----------------------------------------------------
data_temp1=
reshape(source_coded_data_rs,modulate_bit,[])';      %以每组 2 比特进行分组，输出两列数据
modulate_data=pskmod(bi2de(data_temp1),2^(modulate_bit),pi/4);%输出一列数据
%-------------------------插入导频---------------------------------------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,modulate_length);modulate_data(31:60,:)];%在原来输出数据的中间插 0
h1=commsrc.pn('GenPoly',   [1   0   0   0   0   1 1],'NumBitsOut',61*modulate_length,'InitialConditions',[0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
     modulate_data_pn(:,(2*i-1))=modulate_data_temp(:,i);
     modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);modulate_data_pn(1:30,:)];
%----------------------------------ifft-------------------------------------
time_signal_ifft=ifft(modulate_data_pn_out);
%----------------------------------cp--------------------------------------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_ifft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
[time_signal_cp_wide,time_signal_cp_length]=size(time_signal_cp);
%-----------------------------------并串变换-----------------------------------
for ii=1:modulate_length
     time_signal_out(:,ii)=[time_signal_cp(:,2*ii-1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
for a=1:21
% ---------------------------------信道---------------------------------------
% chan=comm.RayleighChannel('SampleRate',550000, ...
%          'PathDelays',[0  2e-6],'AveragePathGains',[0  -3],'MaximumDopplerShift',100,'RandomStream','mt19937ar with seed','Seed',8007);
chan=comm.RayleighChannel('SampleRate',550000, ...
     'PathDelays',[0      2e-6],'AveragePathGains',[0      -3],'MaximumDopplerShift',100);
Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn(    Rayleigh_signal,a-1,'measured');%添加高斯白噪声
%     awgn_signal=time_signal_out_1;
%-----------------------------------串并转换----------------------------------
   receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%-----------------------------------去循环前缀-----------------------------
   receive_data=receive_signal_perallel(27:90,:);
%---------------------------------fft-----------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%-------------------------------信道估计------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length

channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
     estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%-------------------------------------解调---------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=pskdemod(demodulate_data_temp,2^(modulate_bit),pi/4);
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
```

```
%------------------------------------------译码------------------------------------
[real_data_temp1_wide,real_data_length]=size(real_data_temp1);
yrsgs4=reshape(real_data_temp1 ,4,real_data_temp1_wide*real_data_length/4).';
yrsgs41=bi2de(yrsgs4,'left-msb');
yrsgs41=reshape(yrsgs41,nn,length(yrsgs41)/nn).';
ygsrsdecode=rsdec(gf(yrsgs41,4),nn,kk);
d1=reshape(ygsrsdecode.x',1,[]);
d2=de2bi(d1,'left-msb').';
rx_decode=reshape(d2,1,[]);
%--------------------------------------------误码率----------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%---------------------------------------------误符号率-------------------------------
error_symbol_data1=
reshape(rx_decode,modulate_bit,[])';    %以每组2比特进行分组，输出两列数据
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2=
reshape(inforSource,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
end
```

6.2 Code of comparing of QPSK and 16QAM performance

```
function
[error_bit_all,error_symbol_all]=rs_qpsk_coding()
sonCarrierNum_temp=88;
symbols_Per_Carrier=1000;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=2;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));%FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例 1/6~1/4
pilot_Inter=1;%插入导频间隔
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM 符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
kk=11;
%------------------------------信源输入---------------------------------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbols_Per_Carrier*bits_Per_Symbol);
%--------------------------信道编码-----------------------------------------------
msg4_temp=reshape(inforSource,4,[])';
msg4=bi2de(msg4_temp,'left-msb');%将原来的数据转换为 4 位 16 进制
msg4_togf=reshape(msg4,kk,[]).'; %带转换的矩阵，十一输入
msgGF=gf(msg4_togf,4);%转换为伽罗华域
msgrs=rsenc(msgGF,nn,kk); %(15,11）RS 编码 11 个输入 15 个输出
msgrs1=reshape(msgrs.',1,length(msg4)/kk*nn);% 将 rs 编码输出转成一行
msgrs2=de2bi(double(msgrs1.x),'left-msb');%十进制转二进制
source_coded_data_rs=reshape(msgrs2',1,length(msg4)/kk*nn*4);%待调制信号 输出一行信号（数据）
%----------------------------调制-------------------------------------------------
data_temp1=
reshape(source_coded_data_rs,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
modulate_data=pskmod(bi2de(data_temp1),2^(modulate_bit),pi/4);%输出一列数据
%------------------------插入导频---------------------------------------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,modulate_length);modulate_data(31:60,:)];%在原来输出数据的中间插 0
h1=commsrc.pn('GenPoly', [1 0 0 0 0 1 1],'NumBitsOut',61*modulate_length,'InitialConditions',[0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
      modulate_data_pn(:,(2*i-1))=modulate_data_temp(:,i);
      modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);modulate_data_pn(1:30,:)];
%-----------------------------------ifft-------------------------------------
time_signal_ifft=ifft(modulate_data_pn_out);
%----------------------------------cp--------------------------------------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_ifft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
```

```
[time_signal_cp_wide,time_signal_cp_length]=size(time_signal_cp);
%------------------------------------并串变换-------------------------------------
for ii=1:modulate_length
       time_signal_out(:,ii)=[time_signal_cp(:,2*ii-1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
for a=1:21
% ----------------------------------信道---------------------------------------
% chan=comm.RayleighChannel('SampleRate',550000, ...
%          'PathDelays',[0 2e-6],'AveragePathGains',[0 -3],'MaximumDopplerShift',100,'RandomStream','mt19937ar with seed','Seed',8007);
chan=comm.RayleighChannel('SampleRate',550000, ...
       'PathDelays',[0    2e-6],'AveragePathGains',[0    -3],'MaximumDopplerShift',100);
Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn(    Rayleigh_signal,a-1,'measured');%添加高斯白噪声
%       awgn_signal=time_signal_out_1;
%-----------------------------------串并转换---------------------------------
     receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%-----------------------------------去循环前缀--------------------------------
     receive_data=receive_signal_perallel(27:90,:);
%----------------------------------fft--------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%---------------------------------信道估计---------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length

channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
       estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%--------------------------------------解调--------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=pskdemod(demodulate_data_temp,2^(modulate_bit),pi/4);
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
%-------------------------------------------译码---------------------------------
[real_data_temp1_wide,real_data_length]=size(real_data_temp1);
yrsgs4=reshape(real_data_temp1   ,4,real_data_temp1_wide*real_data_length/4).';
yrsgs41=bi2de(yrsgs4,'left-msb');
yrsgs41=reshape(yrsgs41,nn,length(yrsgs41)/nn).';
ygsrsdecode=rsdec(gf(yrsgs41,4),nn,kk);
d1=reshape(ygsrsdecode.x',1,[]);
d2=de2bi(d1,'left-msb').';
rx_decode=reshape(d2,1,[]);
%----------------------------------------------误码率-----------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%-----------------------------------------------误符号率--------------------------------
error_symbol_data1=
reshape(rx_decode,modulate_bit,[])';     %以每组 2 比特进行分组，输出两列数据
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2=
reshape(inforSource,modulate_bit,[])';     %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
end
%-------------------------------------------------------------------
function
[error_bit_all,error_symbol_all]=rs_16qam_coding()
sonCarrierNum_temp=176;
symbols_Per_Carrier=1000;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=4;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));%FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例 1/6~1/4
pilot_Inter=1;%插入导频间隔
```

```
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM 符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
kk=11;
%------------------------------信源输入---------------------------------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbols_Per_Carrier*bits_Per_Symbol);
%--------------------------信道编码-------------------------------------------------
msg4_temp=reshape(inforSource,4,[])';
msg4=bi2de(msg4_temp,'left-msb');%将原来的数据转换为 4 位 16 进制
msg4_togf=reshape(msg4,kk,[]).'; %带转换的矩阵，十一输入
msgGF=gf(msg4_togf,4);%转换为伽罗华域
msgrs=rsenc(msgGF,nn,kk); %(15,11）RS 编码 11 个输入 15 个输出
msgrs1=reshape(msgrs.',1,length(msg4)/kk*nn);% 将 rs 编码输出转成一行
msgrs2=de2bi(double(msgrs1.x),'left-msb');%十进制转二进制
source_coded_data_rs=reshape(msgrs2',1,length(msg4)/kk*nn*4);%待调制信号 输出一行信号（数据）
%---------------------------调制----------------------------------------------------
data_temp1=
reshape(source_coded_data_rs,modulate_bit,[])';      %以每组 2 比特进行分组，输出两列数据
modulate_data=qammod(bi2de(data_temp1),2^(modulate_bit));%输出一列数据
%------------------------插入导频---------------------------------------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,modulate_length);modulate_data(31:60,:)];%在原来输出数据的中间插 0
h1=commsrc.pn('GenPoly',    [1    0    0    0    0    1 1],'NumBitsOut',61*modulate_length,'InitialConditions',[0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
       modulate_data_pn(:,(2*i-1))=modulate_data_temp(:,i);
       modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);modulate_data_pn(1:30,:)];
%----------------------------------ifft------------------------------------
time_signal_ifft=ifft(modulate_data_pn_out);
%---------------------------------cp--------------------------------------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_ifft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
[time_signal_cp_wide,time_signal_cp_length]=size(time_signal_cp);
%------------------------------------并串变换-----------------------------------
for ii=1:modulate_length
       time_signal_out(:,ii)=[time_signal_cp(:,2*ii-1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
for a=1:21
% ----------------------------------信道---------------------------------------
% chan=comm.RayleighChannel('SampleRate',550000, ...
%            'PathDelays',[0  2e-6],'AveragePathGains',[0  -3],'MaximumDopplerShift',100,'RandomStream','mt19937ar with seed','Seed',8007);
chan=comm.RayleighChannel('SampleRate',550000, ...
       'PathDelays',[0       2e-6],'AveragePathGains',[0       -3],'MaximumDopplerShift',100);
Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn(    Rayleigh_signal,a-1,'measured');%添加高斯白噪声
%     awgn_signal=time_signal_out_1;
%-----------------------------------串并转换--------------------------------
     receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%-----------------------------------去循环前缀---------------------------------
     receive_data=receive_signal_perallel(27:90,:);
%----------------------------------fft---------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%--------------------------------信道估计-------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length
```

```
channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
      estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%--------------------------------------解调---------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=qamdemod(demodulate_data_temp,2^(modulate_bit));
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
%-------------------------------------------译码-------------------------------------
[real_data_temp1_wide,real_data_length]=size(real_data_temp1);
yrsgs4=reshape(real_data_temp1  ,4,real_data_temp1_wide*real_data_length/4).';
yrsgs41=bi2de(yrsgs4,'left-msb');
yrsgs41=reshape(yrsgs41,nn,length(yrsgs41)/nn).';
ygsrsdecode=rsdec(gf(yrsgs41,4),nn,kk);
d1=reshape(ygsrsdecode.x',1,[]);
d2=de2bi(d1,'left-msb').';
rx_decode=reshape(d2,1,[]);
%---------------------------------------------误码率----------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%---------------------------------------------误符号率--------------------------------
error_symbol_data1=
reshape(rx_decode,modulate_bit,[])';    %以每组2比特进行分组，输出两列数据
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2=
reshape(inforSource,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
end
```

6.3 Code of comparing of RS coding and RS with Convolutional coding performance

```
function [error_bit_all,error_symbol_all]=rs_coding()
sonCarrierNum_temp=88;
symbols_Per_Carrier=1000;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=2;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));% FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例 1/6~1/4
pilot_Inter=1;%插入导频间隔
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM 符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
kk=11;
%------------------------------信源输入----------------------------------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbols_Per_Carrier*bits_Per_Symbol);
%---------------------------信道编码-----------------------------------------------
msg4_temp=reshape(inforSource,4,[])';
msg4=bi2de(msg4_temp,'left-msb');%将原来的数据转换为 4 位 16 进制
msg4_togf=reshape(msg4,kk,[]).'; %带转换的矩阵，十一输入
msgGF=gf(msg4_togf,4);%转换为伽罗华域
msgrs=rsenc(msgGF,nn,kk); %(15,11）RS 编码 11 个输入 15 个输出
msgrs1=reshape(msgrs.',1,length(msg4)/kk*nn);% 将 rs 编码输出转成一行
msgrs2=de2bi(double(msgrs1.x),'left-msb');%十进制转二进制
source_coded_data_rs=reshape(msgrs2',1,length(msg4)/kk*nn*4);%待调制信号 输出一行信号（数据）
st2 = 4831;
inter_rs_code = randintrlv(source_coded_data_rs,st2); % Interleave.
%----------------------------调制-----------------------------------------------
data_temp1= reshape(inter_rs_code,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
modulate_data=pskmod(bi2de(data_temp1),2^(modulate_bit),pi/4);%输出一列数据
%------------------------插入导频---------------------------------------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,modulate_length);modulate_data(31:60,:)];%在原来输出数据的中间插 0
```

```
h1=commsrc.pn('GenPoly',     [1     0     0     0     0     1 1],'NumBitsOut',61*modulate_length,'InitialConditions',[ 0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
      modulate_data_pn(:,(2*i-1))=modulate_data_temp(:,i);
      modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);modulate_data_pn(1:30,:)];
%----------------------------------ifft------------------------------------
time_signal_ifft=ifft(modulate_data_pn_out);
%----------------------------------cp-------------------------------------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_ifft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
[time_signal_cp_wide,time_signal_cp_length]=size(time_signal_cp);
%------------------------------------并串变换-----------------------------------
for ii=1:modulate_length
      time_signal_out(:,ii)=[time_signal_cp(:,2*ii-1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
for a=1:20
% ----------------------------------信道-------------------------------------
% chan=comm.RayleighChannel('SampleRate',550000, ...
%           'PathDelays',[0  2e-6],'AveragePathGains',[0  -3],'MaximumDopplerShift',100,'RandomStream','mt19937ar with seed','Seed',8007);
chan=comm.RayleighChannel('SampleRate',550000, ...
      'PathDelays',[0       2e-6],'AveragePathGains',[0       -3],'MaximumDopplerShift',100);
Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn(  Rayleigh_signal,a,'measured');% 添加高斯白噪声
%     awgn_signal=time_signal_out_1;
%----------------------------------串并转换----------------------------------
     receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%----------------------------------去循环前缀-----------------------------------
     receive_data=receive_signal_perallel(27:90,:);
%----------------------------------fft--------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%--------------------------------信道估计-----------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length

channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
      estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%--------------------------------------解调---------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=pskdemod(demodulate_data_temp,2^(modulate_bit),pi/4);
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
%-------------------------------------------译码-----------------------------------
deinter_con    =    randdeintrlv(real_data_temp1,st2);    % Deinterleave.
[real_data_temp1_wide,real_data_length]=size(deinter_con);
yrsgs4=reshape(deinter_con ,4,real_data_temp1_wide*real_data_length/4).';
yrsgs41=bi2de(yrsgs4,'left-msb');
yrsgs41=reshape(yrsgs41,nn,length(yrsgs41)/nn).';
ygsrsdecode=rsdec(gf(yrsgs41,4),nn,kk);
d1=reshape(ygsrsdecode.x',1,[]);
d2=de2bi(d1,'left-msb').';
rx_decode=reshape(d2,1,[]);
%---------------------------------------------误码率------------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%----------------------------------------------误符号率-------------------------------
error_symbol_data1=
reshape(rx_decode,modulate_bit,[])';      %以每组 2 比特进行分组，输出两列数据
```

```
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2=
reshape(inforSource,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
end
%-------------------------------------------------------------------
function
[error_bit_all,error_symbol_all]=con_rs_coding()
sonCarrierNum_temp=44;
symbols_Per_Carrier=1000;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=2;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));%FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例 1/6~1/4
pilot_Inter=1;%插入导频间隔
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM 符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
kk=11;
%------------------------------信源输入---------------------------------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbols_Per_Carrier*bits_Per_Symbol);
%--------------------------信道编码-------------------------------------------------
msg4_temp=reshape(inforSource,4,[])';
msg4=bi2de(msg4_temp,'left-msb');%将原来的数据转换为 4 位 16 进制
msg4_togf=reshape(msg4,kk,[]).'; %带转换的矩阵，十一输入
msgGF=gf(msg4_togf,4);%转换为伽罗华域
msgrs=rsenc(msgGF,nn,kk); %(15,11）RS 编码 11 个输入 15 个输出
msgrs1=reshape(msgrs.',1,length(msg4)/kk*nn);% 将 rs 编码输出转成一行
msgrs2=de2bi(double(msgrs1.x),'left-msb');%十进制转二进制
source_coded_data_rs=reshape(msgrs2',1,length(msg4)/kk*nn*4);%待调制信号 输出一行信号（数据）
st2 = 4831;
inter_rs_code = randintrlv(source_coded_data_rs,st2); % Interleave.
trellis = poly2trellis(7,[133 171]);     %(2,1,7)卷积编码输入一位，输出两位，约束长度为 7,1011011-->133//1111001-->171
source_coded_data_con=convenc(inter_rs_code,trellis);
%---------------------------调制-----------------------------------------------------
data_temp1=
reshape(source_coded_data_con,modulate_bit,[])';     %以每组 2 比特进行分组，输出两列数据
modulate_data=pskmod(bi2de(data_temp1),2^(modulate_bit),pi/4);%输出一列数据
%------------------------插入导频---------------------------------------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,modulate_length);modulate_data(31:60,:)];%在原来输出数据的中间插 0
h1=commsrc.pn('GenPoly',    [1   0   0   0   0   1   1],'NumBitsOut',61*modulate_length,'InitialConditions',[0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
      modulate_data_pn(:,(2*i-1))=modulate_data_temp(:,i);
      modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);modulate_data_pn(1:30,:)];
%----------------------------------ifft------------------------------------
time_signal_ifft=ifft(modulate_data_pn_out);
%----------------------------------cp--------------------------------------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_ifft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
[time_signal_cp_wide,time_signal_cp_length]=size(time_signal_cp);
%-----------------------------------并串变换------------------------------------
for ii=1:modulate_length
      time_signal_out(:,ii)=[time_signal_cp(:,2*ii-1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
for a=1:20
% ---------------------------------信道--------------------------------------
% chan=comm.RayleighChannel('SampleRate',550000, ...
```

```
%        'PathDelays',[0 2e-6],'AveragePathGains',[0 -3],'MaximumDopplerShift',100,'RandomStream','mt19937ar with seed','Seed',8007);
chan=comm.RayleighChannel('SampleRate',550000, ...
    'PathDelays',[0    2e-6],'AveragePathGains',[0    -3],'MaximumDopplerShift',100);
Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn( Rayleigh_signal,a,'measured');% 添加高斯白噪声
%    awgn_signal=time_signal_out_1;
%----------------------------------串并转换----------------------------------
    receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%----------------------------------去循环前缀----------------------------------
    receive_data=receive_signal_perallel(27:90,:);
%----------------------------------fft--------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%--------------------------------信道估计--------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length

channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
     estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%-------------------------------------解调---------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=pskdemod(demodulate_data_temp,2^(modulate_bit),pi/4);
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
%----------------------------------------译码-----------------------------------
trellis = poly2trellis(7,[133 171]);    %(2,1,7)卷积编码输入一位，输出两位，约束长度为 7,1011011-->133//1111001-->171
decode_con=vitdec(real_data_temp1,trellis,34,'trunc','hard');    %硬判决
deinter_con  =  randdeintrlv(decode_con,st2);  % Deinterleave.
[real_data_temp1_wide,real_data_length]=size(deinter_con);
yrsgs4=reshape(deinter_con ,4,real_data_temp1_wide*real_data_length/4).';
yrsgs41=bi2de(yrsgs4,'left-msb');
yrsgs41=reshape(yrsgs41,nn,length(yrsgs41)/nn).';
ygsrsdecode=rsdec(gf(yrsgs41,4),nn,kk);
d1=reshape(ygsrsdecode.x',1,[]);
d2=de2bi(d1,'left-msb').';
rx_decode=reshape(d2,1,[]);
%-------------------------------------------误码率----------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%--------------------------------------------误符号率--------------------------------
error_symbol_data1=
reshape(rx_decode,modulate_bit,[])';   %以每组 2 比特进行分组，输出两列数据
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2=
reshape(inforSource,modulate_bit,[])';   %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
%------------------------------------------------------------
function
[error_bit_all,error_symbol_all]=con_rs_coding()
%31 个子频
sonCarrierNum_temp=60;
symbols_Per_Carrier=100;%每子载波含符号数/帧数
bits_Per_Symbol=1;%每符号含比特数
modulate_bit=2;%调制阶数(每个符号比特数)
IFFT_bin_length=2^ceil(log2(sonCarrierNum_temp));%FFT 点数
PrefixRatio=1/4;%保护间隔与 OFDM 数据的比例 1/6~1/4
pilot_Inter=1;%插入导频间隔
% CP=PrefixRatio*IFFT_bin_length ;%每一个 OFDM 符号添加的循环前缀长度为 1/4*IFFT_bin_length
CP=25;
SNR=0:1:20; %信噪比 dB
nn=15;
```

```
kk=11;
%-------------------------------信源输入-----------------------------------------------
inforSource=randi([0,1],1,sonCarrierNum_temp*symbols_Per_Carrier*bits_Per_Symbol);
%--------------------------信道编码-----------------------------------------------
trellis = poly2trellis(7,[133 171]);    %(2,1,7)卷积编码输入一位，输出两位，约束长度为 7,1011011-->133//1111001-->171
source_coded_data_con=convenc(inforSource,trellis);
%---------------------------调制----------------------------------------------------
data_temp1= reshape(source_coded_data_con,modulate_bit,[])';    %以每组 2 比特进行分组，输出两列数据
modulate_data=pskmod(bi2de(data_temp1),2^(modulate_bit),pi/4);%输出一列数据
%------------------------插入导频---------------------------------------------
modulate_data=reshape(modulate_data,60,[]);
[modulate_wide,modulate_length]=size(modulate_data);
modulate_data_temp=[modulate_data(1:30,:);zeros(1,modulate_length);modulate_data(31:60,:)];%在原来输出数据的中间插 0
h1=commsrc.pn('GenPoly',  [1  0  0  0  0  1 1],'NumBitsOut',61*modulate_length,'InitialConditions',[0 0 0 0 0 1]);
pn_code_temp=generate(h1);
pn_code=2*pn_code_temp-1;
pn_code=reshape(pn_code,61,[]);
modulate_data_pn=zeros(61,2*modulate_length);
for i=1:modulate_length
    modulate_data_pn(:,(2*i-1))=modulate_data_temp(:,i);
    modulate_data_pn(:,2*i)=pn_code(:,i);
end
modulate_data_pn(62:64,:)=0;
modulate_data_pn_out=[modulate_data_pn(31:64,:);modulate_data_pn(1:30,:)];
%----------------------------------ifft------------------------------------
time_signal_ifft=ifft(modulate_data_pn_out);
%----------------------------------cp--------------------------------------
time_signal_cp=[time_signal_ifft(39:64,:);time_signal_ifft(1:64,:)];%把 ifft 的末尾 CP 个数补充到最前面
[time_signal_cp_wide,time_signal_cp_length]=size(time_signal_cp);
%------------------------------------并串变换-----------------------------------
for ii=1:modulate_length
    time_signal_out(:,ii)=[time_signal_cp(:,2*ii-1);time_signal_cp(:,2*ii)];
end
time_signal_out_1=reshape(time_signal_out,[],1);
% ----------------------------------信道----------------------------------------
for a=1:20
% chan=comm.RayleighChannel('SampleRate',550000, ...
%          'PathDelays',[0  2e-6],'AveragePathGains',[0  -3],'MaximumDopplerShift',100,'RandomStream','mt19937ar with seed','Seed',8007);
chan=comm.RayleighChannel('SampleRate',550000, ...
    'PathDelays',[0    2e-6],'AveragePathGains',[0    -3],'MaximumDopplerShift',100);
Rayleigh_signal=chan(time_signal_out_1);
awgn_signal=awgn(  Rayleigh_signal,a,'measured');% 添加高斯白噪声
%     awgn_signal=time_signal_out_1;
%------------------------------------串并转换---------------------------------
    receive_signal_serial=awgn_signal;

receive_signal_perallel=reshape(receive_signal_serial,time_signal_cp_wide,[]);
%------------------------------------去循环前缀-------------------------------
    receive_data=receive_signal_perallel(27:90,:);
%----------------------------------fft--------------------------------------
frequency_data_no_cp=fft(receive_data);
frequency_data=[frequency_data_no_cp(35:64,:);frequency_data_no_cp(1:31,:)];
[frequency_data_wide,frequency_data_length]=size(frequency_data);
%--------------------------------信道估计---------------------------------------
channel_condition=zeros(frequency_data_wide,modulate_length);
estimate_data=zeros(frequency_data_wide,modulate_length);
for iii=1:modulate_length

channel_condition(:,iii)=frequency_data(:,2*iii)./pn_code(:,iii);
    estimate_data(:,iii)=frequency_data(:,(2*iii-1))./channel_condition(:,iii);
end
real_data_temp=[estimate_data(1:30,:);estimate_data(32:61,:)];
%--------------------------------------解调-------------------------------------
demodulate_data_temp=reshape(real_data_temp,1,[]);
demodulate_data=pskdemod(demodulate_data_temp,2^(modulate_bit),pi/4);
demodulate_data_bits_temp=reshape(demodulate_data,[],1);
demodulate_data_bits=de2bi(demodulate_data_bits_temp);
```

```
real_data_temp1 = reshape(demodulate_data_bits',1,[]);
%-------------------------------------------译码-------------------------------------
trellis = poly2trellis(7,[133 171]);    %(2,1,7)卷积编码输入一位，输出两位，约束长度为 7,1011011-->133//1111001-->171
rx_decode=vitdec(real_data_temp1,trellis,35,'trunc','hard');   %硬判决
%---------------------------------------------误码率----------------------------------
[error_num,error_ratio]=biterr(inforSource,rx_decode);
error_bit_all(a,1)=error_ratio;
%-----------------------------------------------误符号率--------------------------------
error_symbol_data1= reshape(rx_decode,modulate_bit,[])';   %以每组 2 比特进行分组，输出两列数据
error_symbol_data_receive=bi2de(error_symbol_data1);
error_symbol_data2= reshape(inforSource,modulate_bit,[])';   %以每组 2 比特进行分组，输出两列数据
error_symbol_data_transmite=bi2de(error_symbol_data2);%输出一列数据
[error_symbol_num,error_symbol_ratio]=symerr(error_symbol_data_receive,error_symbol_data_transmite);
error_symbol_all(a,1)=error_symbol_ratio;
end
end
```