

```
%main_OFDM.m
```

```
%一个相对完整的 OFDM 通信系统的仿真设计，包括编码，调制，IFFT，  
%上下变频，高斯信道建模，FFT，PAPR 抑制，各种同步，解调和解码等模  
%块，并统括系统性能的仿真验证了系统设计的可靠性。
```

```
clear all
```

```
close all
```

```
clc
```

```
%++++++全局变量++++++
```

```
% seq_num          表示当前帧是第几帧  
% count_dds_up      上变频处的控制字的累加  
% count_dds_down     下变频处的控制字的累加（整整）  
% count_dds_down_tmp 下变频处的控制字的累加（小数）  
% dingshi           定时同步的定位  
% m_syn             记录定时同步中的自相关平台
```

```
global seq_num
```

```
global count_dds_up
```

```
global count_dds_down
```

```
global count_dds_down_tmp
```

```
global dingshi
```

```
global m_syn
```

```
%++++++
```

```
% SNR_Pre          设定用于仿真的信噪比的初值  
% interval_SNR      设定用于仿真的信噪比间隔  
% frame_num         每一个信噪比下仿真的数据帧数  
% err_int_final     用于计算每帧出现的误比特数  
% fwc_down          设定的接收机初始载波频率控制字  
% fre_offset        设定接收机初始载波频率偏移调整量（单位为 Hz）  
% k0                每次进入卷积编码器的信息比特数  
% G                 卷积编码的生成矩阵
```

```
SNR_Pre=-5;
```

```
interval_SNR=1;
```

```
for SNR_System=SNR_Pre:interval_SNR:5
```

```
frame_num=152;
```

```
dingshi=250;
```

```
err_int_final=0;
```

```
fwc_down=16.050;
```

```
fre_offset=0;
```

```
k0=1;
```

```
G=[1 0 1 1 0 1 1;1 1 1 1 0 0 1];
```

```

disp('-----start-----');

for seq_num=1:frame_num, %frame_num 帧数

%++++++以下为输入数据部分++++++
datain=randint(1,90);
%++++++

%++++++以下为信道卷积编码部分++++++
encodeDATA=cnv_encd(Gk0,datain);
%++++++

%++++++信道交织编码++++++
interlacedata=interlacecode(encodeDATA,8,24);
%++++++

%++++++以下为 QPSK 调制部分++++++
QPSKdata=qpsk(interlacedata);
%++++++

%++++++生成训练序列++++++
if seq_num<3
trainsp_temp=seq_train();
end
%++++++

%++++++插入导频++++++
PILOT=(1+j);
m_QPSKdata=QPSKdata;
data2fft_temp=[m_QPSKdata(1:8),PILOT,m_QPSKdata(9:16),PILOT,m_QPSKdata(17:24),PILOT,m_QPSKdata(25:32),PILOT,m_QPSKdata(33:40),PILOT,m_QPSKdata(41:48),m_QPSKdata(49:56),PILOT,m_QPSKdata(57:64),PILOT,m_QPSKdata(65:72),PILOT,m_QPSKdata(73:80),PILOT,m_QPSKdata(81:88),PILOT,m_QPSKdata(89:end)];
%++++++

trainsp_temp2=[trainsp_temp,zeros(1,128)];
trainsp=[trainsp_temp2(65:256),trainsp_temp2(1:64)];

%++++++降 PAPR 矩阵变换++++++
matix_data=nyquistimp_PS();
matrix_mult=data2fft_temp*matix_data;
%++++++

```

```

data2fft2=[matrix_mult(65:128),zeros(1,128),matrix_mult(1:64)];

%+++++++ifft 运算+++++++
if seq_num==1
    ifftin=trainsp;
elseif seq_num==2
    ifftin=trainsp;
else
    ifftin=data2fft2;
end
IFFTdata=fft_my(conj(ifftin)/256);
IFFTdata=conj(IFFTdata);
% figure
% plot(real(IFFTdata))
% xlabel('realIFFTdata')
% figure
% plot(imag(IFFTdata))
% xlabel('imagIFFTdata')
%+++++++

%+++++++以下为插入循环前后缀，2 倍升采样+++++++
data2fir=add_CYC_upsample(IFFTdata,2);
% ++++++

% ++++++fir 低通滤波+++++++
guiyi_a=[0.0017216    0.010162 0.025512 0.028801 -0.0059219  -0.060115  -0.0496
          0.091431 0.29636  0.3956   0.29636  0.091431 -0.0496  -0.060115  -0.0059219
          0.028801 0.025512 0.010162 0.0017216 ];
%抽样截止频率为 128kHz,通带截止频率为 20kHz，阻带截止频率为 40kHz，带内纹波动小
%于 1dB，带外衰减 100dB
txFIRdatai=filter(guiyi_a,1,real(data2fir));
txFIRdataq=filter(guiyi_a,1,imag(data2fir));
% ++++++

%+++++++发射机 cic 滤波+++++++
CICidat'ai=cic_inter(txFIRdatai,20);
CICidat'aq=cic_inter(txFIRdataq,20);
%+++++++

%+++++++上变频+++++++
fwc_up=16; %控制字可以选择
DUCdata=up_convert_ofdm(fwc_up,CICidat'ai,CICidat'aq);
%+++++++

```

```

%++++++高斯白噪声信道++++++
[DUCdata,datamax]=guiyi_DUCdata(DUCdata);
awgn_data=awgn(DUCdata,SNR_System);
%+++++

%*****接受机*****
%+++++下变频+++++
DUCdata_tmp=awgn_data;
fwc_down=fwc_down+(fre_offset*128/2560000);
r_fre_offset=2560000*((fwc_down-fwc_up)/128);
[DDCdatai,DDCdataq]=down_convert_ofdm(fwc_down,DUCdata_tmp);
%+++++

%+++++接收机 cic 滤波+++++
CICddatai=cic_deci(DDCdatai,40,40);
CICddataq=cic_deci(DDCdataq,40,40);
%+++++

%+++++fir 低通滤波+++++
guiyi_b=[ 0.019527-0.03934 0.049055 -0.018102    -0.1003  0.5944  0.5944  -0.1003
          -0.018102    0.049055 -0.03934 0.019527];
%抽样截止频率为 64kHz,通带截止频率为 20kHz, 阻带截止频率为 30kHz, 带内纹波动小
于 1dB, 带外衰减 60dB
rxFIRdatai=filter(guiyi_b,1,CICddatai);
rxFIRdataq=filter(guiyi_b,1,CICddataq);
%+++++

%+++++量化+++++
q_rxFIRdatai=sign(rxFIRdatai);
q_rxFIRdataq=sign(rxFIRdataq);
%+++++

%+++++定时同步检测+++++
if seq_num<3
time_syn(q_rxFIRdatai,q_rxFIRdataq);
end
%+++++

%+++++频率同步+++++
fre_offset=fre_syn(rxFIRdatai,rxFIRdataq);
%+++++

%+++++fft 运算+++++
if seq_num>2

```

```

seq_num-2
fftw=32+dingshi;
rxFIRdata_syn=rxFIRdatai(fftw:fftw+255)+j*rxFIRdataq(fftw:fftw+255);
FFTdata=fft_my(rxFIRdata_syn);
%+++++

%+++++降 PAPR 逆矩阵变换+++++
fftdata_reg=[FFTdata(193:256),FFTdata(1:64)];
dematrix_data=fftdata_reg*pinv(matix_data);
%+++++

%+++++相位补偿+++++
rx_qpsk_din_th=phase_comp(dematrix_data);
%+++++

%+++++QPSK 解调部分+++++
% figure
% plot(rx_qpsk_din_th, '.')
% xlabel('星座图')
datatemp4=deqpsk(rx_qpsk_din_th);
datatemp4=sign(datatemp4);
for m=1:192
    if datatemp4(m)==-1
        datatemp4(m)=1;
    elseif datatemp4(m)==1
        datatemp4(m)=0;
    end
end
%+++++

%+++++信道解交织+++++
interdout=interlacedecode(datatemp4,8,24);
%+++++

%+++++以下为 viterbi 译码部分+++++
decodeDATA=viterbi(G,k0,interdout);
%+++++

%+++++误比特统计+++++
err_final=sum(abs(decodeDATA-datain))
err_int_final=err_int_final+err_final
end
end
disp('-----');

```

```

SNR_System
err_rate_final((SNR_System-SNR_Pre)./interval_SNR+1)=err_int_final/(90*(frame_num-2))
disp('-----');

end

disp('-----end-----');

SNR_System=SNR_Pre:interval_SNR:5;
figure
semilogy(SNR_System,err_rate_final,'b-*');
xlabel('信噪比/dB')
ylabel('误码率')
axis([-5,5,0,1])
grid on
%+++++

%*****beginning of file*****

%cnv_encd.m
%卷积码编码程序
function output=cnv_encd(G,k0,input)
% cnv_encd(G,k0,input),k0 是每一时钟周期输入编码器的 bit 数,
% G 是决定输入序列的生成矩阵, 它有 n0 行 L*k0 列 n0 是输出 bit 数,
% 参数 n0 和 L 由生成矩阵 G 导出, L 是约束长度。L 之所以叫约束长度
% 是因为编码器在每一时刻里输出序列不但与当前输入序列有关,
% 而且还与编码器的状态有关, 这个状态是由编码器的前(L-1)*k0。
% 个输入决定的,通常卷积码表示为(n0,k0,m), m=(L-1)*k0 是编码
% 器中的编码存储个数, 也就是分为 L-1 段, 每段 k0 个
% 有些人将 m=L*k0 定义为约束长度, 有的人定义为 m=(L-1)*k0
% 查看是否需要补 0, 输入 input 必须是 k0 的整数部

%+++++variables+++++
% G          决定输入序列的生成矩阵
% k0          每一时钟周期输入编码器的 bit 数
% input       输入数据
% output      输入数据
%+++++

if rem(length(input),k0)>0
input=[input,zeros(size(1:k0-rem(length(input),k0)))];
end
n=length(input)/k0;
% 检查生成矩阵 G 的维数是否和 k0 一致

```

```

if rem(size(G,2),k0)>0
error('Error,G is not of the right size.')
end
% 得到约束长度 L 和输出比特数 n0
L=size(G,2)/k0;
n0=size(G,1);
% 在信息前后加 0, 使存储器归 0, 加 0 个数为(L-1)*k0 个
u=[zeros(size(1:(L-1)*k0)),input,zeros(size(1:(L-1)*k0))];
% 得到 uu 矩阵,它的各列是编码器各个存储器在各时钟周期的内容
u1=u(L*k0:-1:1);
%将加 0 后的输入序列按每组 L*k0 个分组, 分组是按 k0 比特增加
%从 1 到 L*k0 比特为第一组, 从 1+k0 到 L*k0+k0 为第二组,.....,
%并将分组按倒序排列。
for i=1:n+L-2
u1=[u1,u((i+L)*k0:-1:i*k0+1)];
end
uu=reshape(u1,L*k0,n+L-1);
% 得到输出, 输出由生成矩阵 G*uu 得到
output=reshape(rem(G*uu,2),1,n0*(L+n-1));
% *****end of file*****

```

```

%*****beginning of file*****
%interlacecode.m

```

```

function dout=interlacecode(din,m,n)
%实现信道的交织编码
%din 为输入交织编码器的数据, m, n 分别为交织器的行列值

```

```

%+++++variables+++++
% din      输入数据
% m        交织器的行值
% n        交织器的列值
% dout     输出数据
%+++++

```

```

for j=1:m
    temp(j,:)=din(j*n-(n-1):j*n);
end

```

```

dout_temp=reshape(temp,1,length(din));
dout=dout_temp(1:end);
%*****end of file*****

```

```

%*****beginning of file*****
%qpsk.m
%QPSK 调制映射
function dout=qpsk(din)

%+++++variables+++++
% din      输入数据
% dout     输出数据
%+++++

din2=1-2*din;
din_temp=reshape(din2,2,length(din)/2);
for i=1:length(din)/2,
    dout(i)=din_temp(1,i)+j*din_temp(2,i);
end
% *****end of file*****

```

```

%*****beginning of file*****
%seq_train.m
%生成用于同步的训练符号
function dout=seq_train()
%第一帧产生短训练序列，第二帧产生长训练序列
%每个短训练符号由 16 个子载波组成，短训练序列
%是由伪随机序列经过数字调制后插 0 后，再经过
%IFFT 之后得到的。具体过程如下：首先采用抽头
%系数为[1 0 0 1]的 4 级移位寄存器产生长度为
%15 的伪随机序列之后末尾补 0，经过 QPSK 调制之
%后的伪随机序列只在 16 的整数倍位置上出现，其
%余的位置补 0，产生长度为 128 的序列，此序列再
%补 128 个 0 经过数据搬移后做 256 点的 IFFT 变换就
%得到 16 个以 16 为循环的训练序列，经过加循环前
%后缀就会产生 20 个相同的短训练序列。长训练序
%列的产生同短训练序列。

global seq_num

if seq_num==1
    fbconnection=[1 0 0 1];
    QPSKdata_pn=[m_sequence(fbconnection),0];
    QPSKdata_pn=qpsk(QPSKdata_pn);
elseif seq_num==2
    fbconnection=[1 0 0 0 0 0 1];

```



```

    QPSKdata_pn=[m_sequence(fbconnection),0];
    QPSKdata_pn=qpsk(QPSKdata_pn);
end

countmod=0;
for k=1:128
    if seq_num==1
        if mod(k-1,16)==0                %生成 16 位循环的短训练符号
            countmod=countmod+1;
            trainsp_temp(k)=QPSKdata_pn(countmod);
        else
            trainsp_temp(k)=0;
        end
    elseif seq_num==2
        if mod(k-1,2)==0
            countmod=countmod+1;
            trainsp_temp(k)=QPSKdata_pn(countmod);
        else
            trainsp_temp(k)=0;
        end
    end
end
end

dout=trainsp_temp;
% *****end of file*****

%*****beginning of file*****
%m_sequence.m
%用线性移位寄存器产生 m 序列

function [mseq]= m_sequence(fbconnection);

%+++++variables+++++
% fbconnection    线性移位寄存器的系数
% mseq            生成的 m 序列
%+++++

n = length(fbconnection);
N = 2^n-1;
register = [zeros(1,n - 1) 1];%定义移位寄存器的初始状态
mseq(1)= register(n);
for i = 2:N
    newregister(1)= mod(sum(fbconnection.*register),2);

```

```

        for j = 2:n,
            newregister(j)= register(j-1);
        end;
        register = newregister;
        mseq(i) = register(n);
    end
% *****end of file*****

%*****beginning of file*****
%nyquistimp_PS.m
%使用改进的 Nyquist 脉冲实现 OFDM 信号的 PAPR 抑制
function dout=nyquistimp_PS()
%改进的 Nyquist 脉冲整形方法能够显著改善 OFDM 信
%号的 PAPR 分布; 该方法实现简单, 和 PTS 和 SLM 相比
%不需迭代计算多个 IFFT 操作, 不需传送边带信息,
%不会引起信号的畸变; 通用性强, 可以调整滚降
%系数以适应任何子载波数的通信系统。当然,
%Nyquist 脉冲成型的方法由于扩展了频谱, 一定程
%度上降低了频谱利用率。

%creat a matrix to shape the subcarries.
%the spectrum of the pulse is as follows:
%    if abs(f)<=Bw*(1-b)
%        spec=1;
%    else if (abs(f)>Bw*(1-b))&(abs(f)<=Bw)
%        spec=exp(aa.*(Bw.*(1-b)-abs(f)));
%    else if (abs(f)>Bw)&(abs(f)<Bw*(1+b))
%        spec=1-exp(aa.*(abs(f)-Bw.*(1+b)));
%    else if abs(f)>=Bw*(1+b)
%        spec=0;
%    end
%    end
%    end
%    end

N=106;
L=11;
b=0.22;

% N=84;
% L=22;
% b=0.5;

```

```

% N=98;
% L=15;
% b=0.3;

% N=116;
% L=6;
% b=0.1;

T=0.004;
Ts=T/N;
Bw=1/Ts;

begin=-Bw*(1+b)+Bw*(1+b)/128;
finish=Bw*(1+b)-Bw*(1+b)/128;
distance=Bw*(1+b)/64;
kk=0;
aa=log(2)/(b.*Bw);

for f=begin:distance:finish
    kk=kk+1;
    if abs(f)<=Bw*(1-b)
        spec=1;
    else if (abs(f)>Bw*(1-b))&(abs(f)<=Bw)
        spec=exp(aa.*(Bw.*(1-b)-abs(f)));
    else if (abs(f)>Bw)&(abs(f)<Bw*(1+b))
        spec=1-exp(aa.*(abs(f)-Bw.*(1+b)));
    else if abs(f)>=Bw*(1+b)
        spec=0;
    end
    end
    end
    end
    C(kk)=spec;
end

for m=0:N-1
    for k=0:(N+2*L-1)
        p(m+1,k+1)=C(k+1)*exp(-i*2*pi.*m.*(k-L)./N);
    end
end

dout=p;

% *****end of file*****

```

```

%*****beginning of file*****
%fft_my.m
%实现 N 点 FFT 运算
function dout=fft_my(din)
%本程序对输入序列 din 实现 DIT——FFT 基 2 算法，点数取大于等于 din 长度的 2 的幂次

%+++++variables+++++
% din          输入数据
% dout         输出数据
%+++++

m=nextpow2(din);
N=2^m;
if length(din)<N
    din=[din,zeros(1,N-length(din))];
end
nxd=bin2dec(fliplr(dec2bin([1:N]-1,m)))+1;
y=din(nxd);
for mm=1:m
    Nmr=2^mm;
    u=1;
    WN=exp(-i*2*pi/Nmr);
    for j=1:Nmr/2
        for k=j:Nmr:N
            kp=k+Nmr/2;
            t=y(kp)*u;
            y(kp)=y(k)-t;
            y(k)=y(k)+t;
        end
        u=u*WN;
    end
end
dout=y;
%*****end of file*****

```

```

%*****beginning of file*****
%add_GI_upsample.m
%加循环前后缀和升采样程序

function dout=add_CYC_upsample(din,upsample)
%插入循环前后缀是将每个 OFDM 符号的前 32 个数据放

```

%到符号尾部，将每个 OFDM 符号的后 32 个数据放到符号头部，  
%升采样是通过中间插零的方式实现

```
%+++++variables+++++
% din      输入数据
% upsample 升采样倍数
% dout     输出数据
%+++++

data_buf=[din(225:256),din,din(1:32)];

temp(1,:)=data_buf;
temp(2:upsample,:)=zeros(upsample-1,length(data_buf));
dout_temp=reshape(temp,1,length(data_buf)*upsample);
dout=dout_temp(1:end);
% *****end of file*****
```

```
%*****beginning of file*****
%cic_inter.m
%发射机的 CIC 滤波器设计
function dout=cic_inter(din,r)
```

%插值 CIC 滤波器通过复制样值实现升采样

```
%+++++variables+++++
% din      输入数据
% r        升采样的插值因子
% dout     输出数据
%+++++

din_1d=[0,din];
diff1=[din,0]-din_1d;
diff1_1d=[0,diff1(1:end-1)];
diff2=[diff1(1:end-1),0]-diff1_1d;
for i=1:r
    temp1(i,:)=diff2(1:end-1);
end
temp2=reshape(temp1,1,length(diff2(1:end-1))*r);
data_int=temp2;
for i=1:length(data_int)
    int1(i)=sum(data_int(1:i));
    int2(i)=sum(int1(1:i));
end
```

```

dout=int2;
%*****end of file*****

%*****beginning of file*****
%up_convert_ofdm.m
%用 DDS 的方式实现上变频
function dout=up_convert_ofdm(fwc_up,dini,dinq)
%fwc_up 是上变频处的频率控制字，每个控制字对应一个载波频率
%count_dds_up 是用于查找存储表的整数值，

%+++++variables+++++
% fwc_down      上变频处的频率控制字
% dini          输入数据的实部
% dinq          输入数据的虚部
% dout          输出数据
%+++++

global seq_num
global count_dds_up

for mk=1:length(dini)
    if (seq_num==1) & (mk==1)
        count_dds_up=0;
    else
        count_dds_up=count_dds_up+fwc_up;
        if count_dds_up>=128
            count_dds_up=count_dds_up-128;
        end
    end
    [up_sin,up_cos]=ram_sin(count_dds_up);
    up_sin_t(mk)=up_sin;
    up_cos_t(mk)=up_cos;
end
for xl=1:length(dini)
    DUCdata(xl)=dini(xl)*up_cos_t(xl)-dinq(xl)*up_sin_t(xl);
end
dout=DUCdata;
% *****end of file*****

%*****beginning of file*****
%ram_sin.m
%构造用于 DDS 的查找表

```

```

function [ysin,ycos]=ram_sin(adr)
%dds 方式需要的 sin 表
%ram_sin 为寄存器名称
%adr 为输入地址,y 为读出数据

%+++++variables+++++
% adr      输入地址
% ysin     输出的正弦值
% ycos     输出的余弦值
%+++++

anl_inc=2*pi/128;
for n=1:128
    data_sin(n)=sin((n-1)*anl_inc);
    data_cos(n)=cos((n-1)*anl_inc);
end
ysin=data_sin(adr+1);
ycos=data_cos(adr+1);
% *****end of file*****

% *****beginning of file*****
%guiyi_DUCdata.m

function [dataout,damax]=guiyi_DUCdata(datain)

%实现数据的归一化

%+++++variables+++++
% datain   输入数据
% dataout  输出数据
% damax    输入数据中的最大值
%+++++

damax=max(abs(datain));
dataout=datain./damax;
% *****end of file*****

% *****beginning of file*****
%down_convert_ofdm.m

function [douti,doutq]=down_convert_ofdm(fwc_down,din)
%用 DDS 的方式实现下变频

```

```

%+++++variables+++++
% fwc_down      下变频处的频率控制字
% din          输入数据
% douti        输出数据的实部
% doutq        输出数据的虚部
%+++++

global seq_num
global count_dds_down
global count_dds_down_tmp

for mkd=1:length(din)
    if (seq_num==1) & (mkd==1)
        count_dds_down=0;
        count_dds_down_tmp=0;
    else
        count_dds_down=round(count_dds_down_tmp+fwc_down);
        count_dds_down_tmp=count_dds_down_tmp+fwc_down;
        if count_dds_down>=128
            count_dds_down=count_dds_down-128;
            count_dds_down_tmp=count_dds_down_tmp-128;
        end
    end
    [up_sin_d,up_cos_d]=ram_sin(count_dds_down);
    up_sin_td(mkd)=up_sin_d;
    up_cos_td(mkd)=up_cos_d;

    DDCdatai(mkd)=din(mkd)*up_cos_td(mkd);
    DDCdataq(mkd)=-din(mkd)*up_sin_td(mkd);
end
douti=DDCdatai;
doutq=DDCdataq;
% *****end of file*****

%*****beginning of file*****
%cic_deci.m
%接收机的 CIC 滤波器设计
function dout=cic_deci(din,r,init)

%抽取 CIC 滤波器通过降采样实现

%+++++variables+++++

```



```

% din      输入数据
% r        降采样的抽取因子
% init     设定的初始值
% dout     输出数据
%+++++

for i=1:length(din),
    int1(i)=sum(din(1:i));
    int2(i)=sum(int1(1:i));
end

data_diff=zeros(1,length(int2)/r);
data_diff=int2(init:r:end);

data_1d=[data_diff,0];
diff1=[0,data_diff]-data_1d;
diff1_1d=[diff1,0];
diff2=[0,diff1]-diff1_1d;
dout=diff2(1:end-2);
% *****end of file*****

%*****beginning of file*****
%time_syn.m
%系统的定时同步
function time_syn(datai,dataq)
%通过前导结构的两个训练帧的延时自相关算法和本地
%互相关检测可以实现精确度非常高的定时同步。

global seq_num
global dingshi
global m_syn

if seq_num==1
    for nc=1:length(datai)-64 %计算相关值
        for m=1:32

m1_syn(m)=(datai(nc+m-1)+j*dataq(nc+m-1))*conj(datai(nc+m-1+16)+j*dataq(nc+m-1+16));
            end
            m2_syn(nc)=sum(m1_syn);
            m_syn(nc)=abs(m2_syn(nc)); %自相关自相关判决函数
        end

% figure

```

```

% plot(m_syn)
% xlabel('采样点索引号')
% ylabel('自相关判决函数')
elseif seq_num==2
    local_seq=[ -1.0000 + 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i
    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    -1.0000 - 1.0000i
    -1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 + 1.0000i    -1.0000 + 1.0000i
    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i
    -1.0000 + 1.0000i    -1.0000 + 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 -
    1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i
    1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i
    1.0000 - 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 +
    1.0000i    -1.0000 - 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    1.0000 + 1.0000i
    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 -
    1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    1.0000 + 1.0000i    -1.0000 + 1.0000i
    -1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 +
    1.0000i    -1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 - 1.0000i
    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i
    -1.0000 - 1.0000i    1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 - 1.0000i    1.0000 - 1.0000i
    1.0000 - 1.0000i    1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i
    1.0000 - 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i
    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 +
    1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i
    1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i
    -1.0000 - 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 - 1.0000i
    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 + 1.0000i    1.0000 + 1.0000i    1.0000 +
    1.0000i    -1.0000 + 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000 - 1.0000i    -1.0000
    + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i
    -1.0000 - 1.0000i    -1.0000 - 1.0000i    1.0000 - 1.0000i    1.0000 - 1.0000i    -1.0000 + 1.0000i
    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 +
    1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i    -1.0000 + 1.0000i
    -1.0000 + 1.0000i    ];

    for nn=1:length(datai)-128 %计算相关值 (输入信号与本地信号互相关)
        for m=1:100
            t1_syn(m)=(datai(nn+m-1)+j*dataq(nn+m-1))*conj(local_seq(m));
        end
        lolol(nn)=sum(t1_syn);
        t_syn(nn)=abs(sum(t1_syn)); %输入信号与本地信号互相关判决函数
    end

    for ni=1:length(t_syn)
        if t_syn(ni)>60
            dingshi=find(t_syn(ni:ni+6))==max(t_syn(ni:ni+6)))+ni-1;

```

```

        break
    end
end
% figure
% plot(t_syn)
% xlabel('采样点索引号')
% ylabel('互相关判决函数')
end
% *****end of file*****

% *****beginning of file*****
%fre_syn.m

function dout=fre_syn(datai,dataq)
%实现系统的频率同步

%频偏的估计范围由帧长度和循环间隔长度来决定。在
%本系统中，短训练序列的频偏估计范围为（2000Hz）；长训
%练序列频偏估计范围为 250Hz，前一个有较大的纠偏范围，估
%计值得到的方差较大；后一个方法的纠偏范围较小，估计值得
%到的方差较小。频率跟踪可以用循环前后缀的周期重复性来完
%成，其频偏估计范围为 125Hz。

%+++++variables+++++
% datai      输入数据的实部
% dataq      输入数据的虚部
% dout       输出数据
%+++++

global seq_num
global dingshi
global m_syn

if seq_num==1
    if m_syn(50)>10
        for m=1:128

fre_back_tmp(m)=(datai(40+m-1)+j*dataq(40+m-1))*conj(datai(40+m-1+16)+j*dataq(40+m-1+1
6));

            end
            fre_back_sum=sum(fre_back_tmp);
            fre_offset_tmp=-320*angle(fre_back_sum)/(2*pi*16);

```

```

        fre_offset=fre_offset_tmp/0.005;
    end
elseif seq_num==2
    for m=1:128

        fre_back_tmp(m)=(datai(dingshi+m-1)+j*dataq(dingshi+m-1))*conj(datai(dingshi+m-1+128)+j*d
        ataq(dingshi+m-1+128));
        end
        fre_back_sum=sum(fre_back_tmp);
        fre_offset_tmp=-320*angle(fre_back_sum)/(2*pi*128);
        fre_offset=fre_offset_tmp/0.005;

    elseif seq_num>2
        for m=1:48

            fre_back_tmp(m)=(datai(dingshi+m)+j*dataq(dingshi+m))*conj(datai(dingshi+m+256)+j*dataq(d
            ingshi+m+256));
            end
            fre_back_sum=sum(fre_back_tmp);
            fre_offset_tmp=-320*angle(fre_back_sum)/(2*pi*256);
            fre_offset= fre_offset_tmp/0.005;
        end
        dout=fre_offset;
        % *****end of file*****

        % *****beginning of file*****

        %phase_comp.m
        %实现 OFDM 符号的相位补偿

        function    dout=phase_comp(din)

        %使用导频的相位信息来调整有效数据的偏移相位,
        %以实现正确的 QPSK 解调

        %+++++variables+++++
        % din        输入数据
        % dout        输出数据
        %+++++

        qpsk_din=[din(1:8),din(10:17),din(19:26),din(28:35),din(37:44),din(46:53),din(54:61),din(63:70),
        din(72:79),din(81:88),din(90:97),din(99:106)];
        pilot=[din(9),din(18),din(27),din(36),din(45),din(62),din(71),din(80),din(89),din(98)];
        ang_offset=angle(sum((1-j)*pilot));

```

```

qpsk_din=qpsk_din*exp(-j*ang_offset);
dout=qpsk_din;
% *****end of file*****

```

```

% *****beginning of file*****
%deqpsk.m

```

```

function dout=deqpsk(din)
%实现 QPSK 解调

```

```

%+++++variables+++++
% din      输入数据
% dout     输出数据
%+++++

```

```

dout_temp(1,:)=real(din);
dout_temp(2,:)=imag(din);

```

```

dout=reshape(dout_temp,1,length(din)*2);
% *****end of file*****

```

```

% *****beginning of file*****
%interlacedecode.m

```

```

function dout=interlacedecode(din,m,n)
%实现信道的交织解码

```

```

%+++++variables+++++
% din      输入数据
% m        交织器的行值
% n        交织器的列值
% dout     输出数据
%+++++

```

```

temp=reshape(din,m,n);
for j=1:m
    dout_temp(j*n-(n-1):j*n)=temp(j,:);
end

```

```

dout=dout_temp(1:end);
% *****end of file*****

```

```

%*****beginning of file*****
%viterbi.m
%viterbi 解码程序
% function [decoder_output,survivor_state,cumulated_metric]=viterbi(G,k,channel_output)
function decoder_output=viterbi(G,k,channel_output)
% [decoder_output,survivor_state,cumulated_metric]=viterbi(G,k,channel_output)
% 其中 G 是一个 n 行 L*k 列矩阵，它的每一行决定了从移位寄存器到输入码字的连接
方式。
% survivor_state 是一个矩阵，它显示了通过网格的最优路径，这个矩阵通过一个单独
% 的函数 metric(x,y)给出。

n=size(G,1);
%检验 G 的维数
if rem(size(G,2),k)~=0
error('Size of G and k do not agree')
end
if rem(size(channel_output,2),n)~=0
error('channle output not of the right size')
end
L=size(G,2)/k;
number_of_states=2^((L-1)*k);
%产生状态转移矩阵，输出矩阵和输入矩阵
for j=0:number_of_states-1
for t=0:2^k-1
[next_state,memory_contents]=nxt_stat(j,t,L,k);
input(j+1,next_state+1)=t;
branch_output=rem(memory_contents*G',2);
nextstate(j+1,t+1)=next_state;
output(j+1,t+1)=bin2deci(branch_output);
end
end
input;
state_metric=zeros(number_of_states,2);
depth_of_trellis=length(channel_output)/n;
channel_output_matrix=reshape(channel_output,n,depth_of_trellis);
survivor_state=zeros(number_of_states,depth_of_trellis+1);
[row_survivor col_survivor]=size(survivor_state);
%开始非尾信道输出的解码
%i 为段，j 为每一阶段的状态，t 为输入
for i=1:depth_of_trellis-L+1
flag=zeros(1,number_of_states);
if i<=L
step=2^((L-i)*k);

```

```

else
step=1;
end
for j=0:step:number_of_states-1
for t=0:2^k-1
branch_metric=0;
binary_output=deci2bin(output(j+1,t+1),n);
for tt=1:n
branch_metric=branch_metric+metric(channel_output_matrix(tt,i),binary_output(tt));
end
if
((state_metric(nextstate(j+1,t+1)+1,2)>state_metric(j+1,1)+branch_metric)&flag(nextstate(j+1,t+1)
+1)==0)
state_metric(nextstate(j+1,t+1)+1,2)=state_metric(j+1,1)+branch_metric;
survivor_state(nextstate(j+1,t+1)+1,i+1)=j;
flag(nextstate(j+1,t+1)+1)=1;
end
end
end
state_metric=state_metric(:,2:-1:1);
end
%开始尾信道输出的解码
for i=depth_of_trellis-L+2:depth_of_trellis
flag=zeros(1,number_of_states);
last_stop=number_of_states/(2^((i-depth_of_trellis+L-2)*k));
for j=0:last_stop-1
branch_metric=0;
binary_output=deci2bin(output(j+1,1),n);
for tt=1:n
branch_metric=branch_metric+metric(channel_output_matrix(tt,i),binary_output(tt));
end
if
((state_metric(nextstate(j+1,1)+1,2)>state_metric(j+1,1)+branch_metric)&flag(nextstate(j+1,1)+1)
==0)
state_metric(nextstate(j+1,1)+1,2)=state_metric(j+1,1)+branch_metric;
survivor_state(nextstate(j+1,1)+1,i+1)=j;
flag(nextstate(j+1,1)+1)=1;
end
end
end
state_metric=state_metric(:,2:-1:1);
end
%从最优路径产生解码输出
%由段得到状态序列，再由状态序列从 input 矩阵中得到该段的输出
state_sequence=zeros(1,depth_of_trellis+1);

```

```

size(state_sequence);
state_sequence(1,depth_of_trellis)=survivor_state(1,depth_of_trellis+1);
for i=1:depth_of_trellis
state_sequence(1,depth_of_trellis-i+1)=survivor_state((state_sequence(1,depth_of_trellis+2-i)+1),
depth_of_trellis-i+2);
end
state_sequence;
decoder_output_matrix=zeros(k,depth_of_trellis-L+1);
for i=1:depth_of_trellis-L+1
dec_output_deci=input(state_sequence(1,i)+1,state_sequence(1,i+1)+1);
dec_output_bin=deci2bin(dec_output_deci,k);
decoder_output_matrix(:,i)=dec_output_bin(k:-1:1)';
end
decoder_output=reshape(decoder_output_matrix,1,k*(depth_of_trellis-L+1));
cumulated_metric=state_metric(1,1);
% *****end of file*****

```

```

%*****beginning of file*****
%nxt_stat.m
%viterbi 解码子程序

```

```

function [next_state,memory_contents]=nxt_stat(current_state,input,L,k)

binary_state=deci2bin(current_state,k*(L-1));
binary_input=deci2bin(input,k);
next_state_binary=[binary_input,binary_state(1:(L-2)*k)];
next_state=bin2deci(next_state_binary);
memory_contents=[binary_input,binary_state];
% *****end of file*****

```

```

%*****beginning of file*****
%deci2bin.m
function y=deci2bin(x,t)
% 十进制数 x 转化为二进制数，二进制数至少表示为 t 位

%*****variables*****
% x      输入数据
% t      二进制数位数
% y      输出数据
%*****

y=zeros(1,t);

```



```

i=1;
while x>=0&i<=t
y(i)=rem(x,2);
x=(x-y(i))/2;
i=i+1;
end
y=y(t:-1:1);
% *****end of file*****

```

```

%*****beginning of file*****

```

```

%bin2deci.m

```

```

function y=bin2deci(x)

```

```

% 将二进制数转化为十进制数

```

```

%+++++variables+++++

```

```

% x      输入数据

```

```

% y      输出数据

```

```

%+++++

```

```

t=length(x);

```

```

y=(t-1:-1:0);

```

```

y=2.^y;

```

```

y=x*y';

```

```

% *****end of file*****

```

```

%*****beginning of file*****

```

```

%metric.m

```

```

%viterbi 解码子程序

```

```

function distance=metric(x,y)

```

```

if x==y

```

```

distance=0;

```

```

else

```

```

distance=1;

```

```

end

```

```

% switch (y)

```

```

% case 0

```

```

% if x==0

```

```

% distance=0.0458;

```

```

% end

```

```

% if x==1

```

```

% distance=2;

```

```

% end
% if x==2;
% distance==1.0458;
% end
% case 1
% if x==0
% distance==2;
% end
% if x==1
% distance=0.0458;
% end
% if x==2
% distance=1.0458;
% end
% otherwise,
% break;
% end
% *****end of file*****

```