Reference codes for 16QAM simulations

March 28 – April 19 2008

Yuping Zhao (Doctor of Science in technology)

Professor, Peking University
Beijing, China
Yuping.zhao@pku.edu.cn

1. Defining Parameters

```
% Definition of parameters
    % calculate the least number simulation needed.
    M=16;                                   % MQAM
    snr=[0:12]+10*log10(4);                 % snr for awgn channe
                                            % error propability in theory
    error_theory=(1-(1-2*(1-1/(sqrt(M)))*qfunc(sqrt(10.^(snr/10)*3/(M-1)))).^2)/4;
                                            % number of binary random data input
    len=floor(1./error_theory)*100;
    len(find(len<5000))=5000;
    % parameters of sqrt cosine filter
    R=0.5;          % roll-factor
    delay=3;        % groop delay of the filter
    sample_rate=8;  % up-sampling rate
```

2-1 main

```
% simulation the ber for different ber
    ber_out=zeros(1,length(snr));
    for ii=1:length(snr)
    % Data generation
        data_in=randint(1,len(ii));
    % 16QAM mudulation
        modulated=QAM16(data_in);
    % up-sampling
        upsampI=up_sample(real(modulated),sample_rate);
        upsampQ=up_sample(imag(modulated),sample_rate);
    % Pulse Shaping Filter
        TxI=my_sqrt_filter(upsampI,R,delay,sample_rate);
        TxQ=my_sqrt_filter(upsampQ,R,delay,sample_rate);
        Tx=TxI+j*TxQ;
```

2-2 main

```
% Add white Gauss Noises
        Rx=add_noise(Tx,snr(ii));
    % Filtered by match filter
        filteredI=my_sqrt_filter(real(Rx),R,delay,sample_rate);
        filteredQ=my_sqrt_filter(imag(Rx),R,delay,sample_rate);
    % Down-sampling
        downsamp=down_sample(filteredI,sample_rate,delay)
        +j*down_sample(filteredQ,sample_rate,delay);
    % Demodulation and adjudge
        demodu=De16QAM(downsamp);
    % Calculate the BER and SER
        ber=Cal_ber(data_in,demodu);
        ber_out(ii)=ber;
     end;
```

3-1 modulation

```
function  y=QAM16(x)
    len=floor(length(x)/4);       % length of input binary data
    I=zeros(1,len);               % phase I and phase Q
    Q=zeros(1,len);
    for i=1:len
        tri_bit=x((i-1)*4+1:i*4);  % every 4 bits mapped into 1 symbol
        % the first and third bits decide phase I
        if [tri_bit(1,1),tri_bit(1,3)]==[0 0]
            I(i)=-3;
        elseif [tri_bit(1,1),tri_bit(1,3)]==[0 1]
            I(i)=-1;
        elseif [tri_bit(1,1),tri_bit(1,3)]==[1 1]
            I(i)=1;
        else
            I(i)=3;
        end;
```

3-2 modulation

```
% the second and fourth bits decide phase Q
        if [tri_bit(1,2),tri_bit(1,4)]==[0 0]
            Q(i)=-3;
        elseif [tri_bit(1,2),tri_bit(1,4)]==[0 1]
            Q(i)=-1;
        elseif [tri_bit(1,2),tri_bit(1,4)]==[1 1]
            Q(i)=1;
        else
            Q(i)=3;
        end;
    end;
    y=I+j*Q;
```

4. up-sampling

```
function data_sampled=up_sample(data_in,sample_rate);
    data_sampled=zeros(1,length(data_in)*sample_rate);
    data_sampled(1:sample_rate:end)=data_in;
```

5. Make filter function

```
function data_filtered=my_sqrt_filter(data_in,R,delay,sample_rate)
% give the time responce of sqrt rcosine filter
    len_filter=delay*2*sample_rate+1;
    n=0:len_filter-1;
    n=-1*delay*sample_rate:delay*sample_rate;
    part1=cos((1+R)*pi*n/sample_rate);
    part2=sin((1-R)*pi*n/sample_rate)./(4*R*n/sample_rate);
    part3=1-(4*R*n/sample_rate).^2;
    hn=4*R*(part1+part2)./(pi*part3);
    % deal with the Inf points
    hn(delay*sample_rate+1)=4*R/pi+1-R;
    index=find(n==sample_rate/4/R | n==-1*sample_rate/4/R);
    hn(index)=(R*sqrt(2)/2/pi)*((2+pi)*sin(pi/4/R)+(pi-2)*cos(pi/4/R));
    % change the maximum value to make the raised cosine filter's maximum value=1
    max_value=max(conv(hn,hn));
    hn=hn/sqrt(max_value);
% the input data filtered
    data_filtered=conv(data_in,hn);
```

6. Carrier modulation

```
function Txfc=CarriModu(Tx,Nfc,sample_rate);
    % common multiple
    rate=Nfc*sample_rate;
    % t=n*Ts/rate;
    % 2*pi*fc*t=2*pi*(Nfc/Ts)*(n*Ts/rate)=2*pi*Nfc*n/rate;
    tx_temp=zeros(1,Nfc*length(Tx));
    for ii=1:length(Tx)
        tx_temp((ii-1)*Nfc+1:ii*Nfc)=Tx(ii);
    end;
    n=0:Nfc*length(Tx)-1;
    angle_carrier=2*pi*Nfc*n/rate;
    Txfc=real(tx_temp).*cos(angle_carrier)-imag(tx_temp).*sin(angle_carrier);
```

7. Add AWGN signals

```
function y=add_noise(x,snr);
  len=length(x);
  power_s=10;                  % the average power of signal
  mean=0;                                % mean
  variance=power_s/(10^(snr/10));              % variance
% function conversion to generate Gauss distribution random data
  % from uniform distribution random data
  rand_uniform=rand(1,len);
  % generate Rayleigh distribution random data to be amplitude
amplitude_rand=sqrt(-2*log(rand_uniform));
angle_rand=2*pi*rand(1,len);
  noise_I=mean + sqrt(variance/2) * amplitude_rand.*cos(angle_rand);   % noise for I
  noise_Q=mean + sqrt(variance/2) * amplitude_rand.*sin(angle_rand);  % noise for Q
  noise=noise_I+j*noise_Q;                    % I+jQ
y=x + noise;                           % add noise to signal
```

8. Filter and down sampling

```
% Filtered by match filter
        filteredI=my_sqrt_filter(real(Rx),R,delay,sample_rate);
        filteredQ=my_sqrt_filter(imag(Rx),R,delay,sample_rate);

function y=down_sample(x,sample_rate,delay);
    y=x(2*delay*sample_rate+1:sample_rate:(length(x)-2*delay*sample_rate));
```

9-1. demodulation method 1

```
function y=De16QAM(x);
    I=real(x);
    Q=imag(x);
    y=zeros(1,length(I)*4);
    y(1:4:end)=(I>0);
    y(2:4:end)=(Q>0);
    y(3:4:end)=(I>-2 & I<2);
    y(4:4:end)=(Q>-2 & Q<2);
```

9-2 demodulation method 2

```
function y=De16QAM2(x);
    I=real(x);
    Q=imag(x);
    IandQ=[3,3,3,3,1,1,1,1,-1,-1,-1,-1,-3,-3,-3,-3;
           3,1,-1,-3,3,1,-1,-3,3,1,-1,-3,3,1,-1,-3];
   tempy=zeros(length(I),2);
   for ii=1:length(I)
       closest=1;
       temp1=ones(1,16)*I(ii);
       temp2=ones(1,16)*Q(ii);
       temp=(temp1-IandQ(1,:)).^2+(temp2-IandQ(2,:)).^2;
       index=find(temp==min(temp));
       tempy(ii,:)=IandQ(:,index)';
    end;
    tempy
    y=zeros(1,length(I)*4);
```

```
for jj=1:length(I)
        if tempy(jj,1)==3
            y(1,(jj-1)*4+1)=1;
            y(1,(jj-1)*4+3)=0;
        elseif tempy(jj,1)==1
            y(1,(jj-1)*4+1)=1;
            y(1,(jj-1)*4+3)=1;
        elseif tempy(jj,1)==-1
            y(1,(jj-1)*4+1)=0;
            y(1,(jj-1)*4+3)=1;
        else
            y(1,(jj-1)*4+1)=0;
            y(1,(jj-1)*4+3)=0;
        end;
        if tempy(jj,2)==3
            y(1,(jj-1)*4+2)=1;
            y(1,(jj-1)*4+4)=0;
        elseif tempy(jj,2)==1
            y(1,(jj-1)*4+2)=1;
            y(1,(jj-1)*4+4)=1;
        elseif tempy(jj,2)==-1
            y(1,(jj-1)*4+2)=0;
            y(1,(jj-1)*4+4)=1;
        else
            y(1,(jj-1)*4+2)=0;
            y(1,(jj-1)*4+4)=0;
        end;
    end;
```

Get the simulated BER

```
function ber=Cal_ber(tx_data,rx_data)
    len=length(rx_data);
    diff=tx_data(1:len) - rx_data;
    error_ones=(diff~=0);
    error_num=sum(error_ones);
```

Thanks!