EEE 431

Project 2

Report

Outline

Introduction

In this project, several digital communication systems are studied. In the first part of the project, digital modulation types are simulated for calculating probability of error. The types simulated are binary PSK, on-off signaling, M-ary PAM, and M-PSK respectively. In each part, modulation with respect to these types are performed over random signals and suitable decision rules are applied to receive those signals. Then, the error probabilities are compared with the theoretical values. In the second part, baseband signaling is practiced over a random signal for binary PAM modulation type. Signal is matched with a pulse and sampled, and then again the error probability is checked. Also the effect of delay over sampling is observed in this part.

Work Done

Part I

a)

In part I part a, a simulation for the probability of error for BPSK is built. 100000 bits to be transmitted are produced in MATLAB as follows:

```
x_binary=randi([0 M-1],10^5,1);
```

where *randi* function is a random integer generator and M equals 2 in this binary case.

In BPSK modulation, the bit 0 is coded as "-1" and bit 1 is coded as "1". That modulation scheme is applied as follows:

```
for i=1:10^5
    if x_binary(i)==0
        y_binary(i)=-1;
    else
        y_binary(i)=1;
    end
end
```

After this coding, white Gaussian noise has to be added to the signal. White noise is basically a normally distributed random number with 0 mean and $N_0/2$ variance which is added to the signal as follows:

```
for i=1:10^5
    y_binary_noisy(i)=y_binary(i)+randn(1)*sqrt(N0/2);
end
```

Here, noise depends on signal power and signal to noise ratio as N_0 is the ratio of these two. Hence, N_0 is calculated for a specific value of SNR in dB scale as follows:

```
power_binary=1/M*sum((-1)^2+(1)^2);
SNR_binary_dB=2;%2%9
SNR_binary=10^(SNR_binary_dB/10);
SNR_binary_bit=SNR_binary/log2(M);
N0=power_binary/SNR_binary_bit;
```

Signal power for BPSK is equal to 1 and SNR per bit is equal to real SNR value in binary case as $\log_2 2$ is equal to 1 as well.

After the transmission, in order to receive the signal ML rule is applied as follows:

```
number_of_error_binary=0;
for i=1:10^5
    if real(y_binary_noisy(i))<0
        m_hat_binary(i)=0;
    else
        m_hat_binary(i)=1;
    end
    if x_binary(i)~=m_hat_binary(i)
        number_of_error_binary=number_of_error_binary+1;
    end
end
```

Here, if the noisy signal is smaller than the midpoint 0, received signal named as *m_hat* is equaled to 0 and if it is greater than or equal to 0, it is equaled to 1. Then, the initial values that are transmitted is compared with the received values one by one and if there is an inequality a number of error is count.

Probability of error in binary case is calculated by dividing total number of errors to total number of values transmitted.

```
P_error_binary=number_of_error_binary/10^5;
```

And, the theoretical error probability is calculated by the following formula:

```
P_error_binary_theo=1/2*erfc(sqrt(2*SNR_binary_bit)/sqrt(2));
```

Simulation error and theoretical error are found as follows after running this simulation for SNR value of 2 dB:

```
P_error_binary =

    0.0373

P_error_binary_theo =

    0.0375
```

The same values respectively are found for 6 dB as follows:

```
P_error_binary =

    0.0025

P_error_binary_theo =

    0.0024
```

And for 9 dB:

```
P_error_binary =

    3.0000e-05

P_error_binary_theo =

    3.3627e-05
```

Here, it can be observed that when signal power over noise power increases, probability of error decreases drastically. Also simulation error and theoretical error values are too close and consistent for every value.

b)

In part I part b, a simulation for the probability of error for on-off signaling is built. 100000 bits to be transmitted are produced in MATLAB as follows:

```
x_onoff=randi([0 M-1],10^5,1);
```

where *randi* function is a random integer generator and M equals 2 in this case again.

In on-off modulation, the bit 0 is coded as "0" and bit 1 is coded as "1". Therefore there is no difference between the input and output of the modulation:

```
y_onoff=x_onoff;
```

After this coding, white Gaussian noise has to be added to the signal. White noise is basically a normally distributed random number with 0 mean and $N_0/2$ variance which is added to the signal in the same way:

```
for i=1:10^5
    y_onoff_noisy(i)=y_onoff(i)+randn(1)*sqrt(N0/2);
end
```

Here, noise depends on signal power and signal to noise ratio as N_0 is the ratio of these two. Hence, N_0 is calculated for a specific value of SNR in dB scale as follows:

```
power_onoff=1/M*sum((0)^2+(1)^2);
SNR_onoff_dB=5;%5%12
SNR_onoff=10^(SNR_onoff_dB/10);
SNR_onoff_bit=SNR_onoff/log2(M);
N0=power_onoff/SNR_onoff_bit;
```

Signal power for on-off signaling is equal to 0.5 and SNR per bit is also equal to real SNR value in this case as $\log_2 2$ is equal to 1 again.

After the transmission, in order to receive the signal ML rule is applied as follows:

```
number_of_error_onoff=0;
for i=1:10^5
    if real(y_onoff_noisy(i))<0.5
        m_hat_onoff(i)=0;
    else
        m_hat_onoff(i)=1;
    end
    if x_onoff(i)~=m_hat_onoff(i)
        number_of_error_onoff=number_of_error_onoff+1;
    end
end
```

Here, if the noisy signal is smaller than the midpoint 0.5, received signal named as *m_hat* is equaled to 0 and if it is greater than or equal to 0.5, it is equaled to 1. Then, the initial values that are transmitted is compared with the received values one by one and if there is an inequality a number of error is count.

Probability of error in on-off case is calculated by dividing total number of errors to total number of values transmitted.

```
P_error_onoff=number_of_error_onoff/10^5;
```

And, the theoretical error probability is calculated by the following formula:

```
P_error_onoff_theo=1/2*erfc(sqrt(SNR_onoff_bit)/sqrt(2));
```

Simulation error and theoretical error are found as follows after running this simulation for SNR value of 5 dB:

```
P_error_onoff =

    0.0370

P_error_onoff_theo =

    0.0377
```

The same values respectively are found for 9 dB as follows:

```
P_error_onoff =

    0.0022

P_error_onoff_theo =

    0.0024
```

And for 12 dB:

```
P_error_onoff =

    4.0000e-05

P_error_onoff_theo =

    3.4303e-05
```

Here, it can be observed that when signal power over noise power increases, probability of error decreases again. Also simulation error and theoretical error values are too close and consistent for every value. It can be commented that on-off case is very close to binary case in means of error probability as decision rule applied is very similar.

c)

In part c, this time the simulation is built for M-ary PAM. 100000 random values are produced in MATLAB again.

```
x_PAM=randi([0 M-1],10^5,1);
```

where M is changed to 4, 8, and 16 respectively changing the interval of the random integer values.

In M-ary PAM modulation, the following coding is applied:

```
for i=1:10^5
    for m=x_PAM(i)
        y_PAM(i)=2*m-M+1;
    end
```

```
end
```

to distribute for example M=4 values of 0,1,2,3 to -3,-1,1,3 respectively.

After this modulation, white Gaussian noise is added to the signal again.

```
for i=1:10^5
    y_PAM_noisy(i)=y_PAM(i)+randn(1)*sqrt(N0/2);
end
```

Signal power and N_0 are calculated for each value of M for a specific SNR value in dB scale as follows:

```
s=0;
for i=1:M
    s=s+(2*(i-1)-M+1)^2;
end
power_PAM=1/M*s;

SNR_PAM_dB=5;%5%13

SNR_PAM=10^(SNR_PAM_dB/10);
SNR_PAM_bit=SNR_PAM/log2(M);
N0=power_PAM/SNR_PAM;
```

Here, signal power is calculated as 5 for M=4, 21 for M=8, and 85 for M=16. As symbol error is desired, power is not divided to SNR per bit value, but instead divided to real SNR value.

After the transmission, the following ML rule is applied to receive every symbol:

```
for i=1:10^5
    for m=1:M-2
        if real(y_PAM_noisy(i))<-(M-2)
            m_hat_PAM(i)=0;
        elseif real(y_PAM_noisy(i))>=(2*m-M) && real(y_PAM_noisy(i))<(2*m-M+2)
            m_hat_PAM(i)=m;
        elseif real(y_PAM_noisy(i))>=(M-2)
            m_hat_PAM(i)=M-1;
        end
    end
end
```

Here, for example for M equals 4 case, every noisy signal value is checked whether it falls to region smaller than -2, in between -2 and 0, 0 and 2, or greater than 2. Hence, every value received named as *m_hat* is matched to one of the symbols 0,1,2,3 according to its region respectively.

After applying this decision rule, total number of error is counted again by comparing every value of the original signal and received signal as follows:

```
number_of_error_PAM=0;
for i=1:10^5
    if x_PAM(i)~=m_hat_PAM(i)
        number_of_error_PAM=number_of_error_PAM+1;
    end
end
```

Probability of error in M-ary PAM case is calculated by dividing total number of errors to total number of values transmitted.

```
P_error_PAM=number_of_error_PAM/10^5;
```

And, the theoretical error probability is calculated by the following formula:

```
P_error_PAM_theo=(2*(M-1)/M)*1/2*erfc(sqrt(6*log2(M)/(M^2-
1)*SNR_PAM_bit)/sqrt(2));
```

Simulation error and theoretical error for M=4 are found as follows after running this simulation for SNR value of 5 dB:

```
P_error_PAM =

    0.1982

P_error_PAM_theo =

    0.1955
```

The same values respectively are found for 9 dB as follows:

```
P_error_PAM =

    0.0557

P_error_PAM_theo =

    0.0560
```

And for 13 dB:

```
P_error_PAM =

    0.0039

P_error_PAM_theo =

    0.0035
```

For M=8, SNR=8

```
P_error_PAM =

    0.3842

P_error_PAM_theo =

    0.3835
```

SNR=13

```
P_error_PAM =

    0.1471

P_error_PAM_theo =

    0.1470
```

SNR=17

```
P_error_PAM =

    0.0249

P_error_PAM_theo =

    0.0253
```

And for M=16, SNR=10

```
P_error_PAM =

    0.5882

P_error_PAM_theo =

    0.5884
```

SNR=16

```
P_error_PAM =

    0.3122

P_error_PAM_theo =

    0.3123
```

SNR=22

```
P_error_PAM =

    0.0502

P_error_PAM_theo =

    0.0501
```

Here, it can be observed that when symbol number and hence number of decision regions increases, probability of error increases. Also, again, when signal power over noise power increases, probability of error decreases. Simulation error and theoretical error values are found too close in every case in this part as well.

d)

In part d, simulation is built for 4-PSK and 8-PSK. Again 100000 random values are produced in MATLAB.

```
x_PSK=randi([0 M-1],10^5,1);
```

where M is changed to 4 for 4-PSK and 8 for 8-PSK.

In PSK modulation symbol values are coded in phase and carried in cosine and sine components of the signal. Hence, the following code is applied to modulate the symbol values:

```
for i=1:10^5
    for m=x_PSK(i)
        y_PSK1(i)=cos(2*pi/M*m);
        y_PSK2(i)=sin(2*pi/M*m);
    end
end
```

And white Gaussian noise is added as follows:

```
for i=1:10^5
    y_PSK1_noisy(i)=y_PSK1(i)+randn(1)*sqrt(N0/2);
    y_PSK2_noisy(i)=y_PSK2(i)+randn(1)*sqrt(N0/2);
end
```

with respect to the following N_0 calculation:

```
s=0;
for i=1:M
    s=s+(cos(2*pi/M*(i-1)))^2+(sin(2*pi/M*(i-1)))^2;
end
power_PSK=1/M*s;
SNR_PSK_dB=10;
SNR_PSK=10^(SNR_PSK_dB/10);
SNR_PSK_bit=SNR_PSK/log2(M);
N0=power_PSK/SNR_PSK;
```

Here, signal power comes 1 as it is equal to the square of amplitude of the cosine and sine signals in PSK modulation. SNR value is selected as 10 dB for this part. Again, as symbol error is desired, real value of the SNR is used to calculate N_0.

After the transmission, the following ML rule is applied to receive every symbol:

```
for i=1:10^5
    for m=1:M
        if real(arcn(i))>pi/M*(2*m-3) && real(arcn(i))<=pi/M*(2*m-1)
            m_hat_PSK(i)=m-1;
        end
    end
end
```

where *arcn* is the inverse tangent of the noisy sine and cosine components and calculated as follows:

```
for i=1:10^5
    arc(i)=atan2(y_PSK2_noisy(i),y_PSK1_noisy(i));
    if arc(i)<-pi/M
        arcn(i)=arc(i)+2*pi;
    else
        arcn(i)=arc(i);
    end
end
```

Here, inverse tangents are turned back in four quadrants and distributed according to the pi/M shifted regions of the decision rule.

Then, again the received signal is compared with the original signal to count the total number of errors.

```
number_of_error_PSK=0;
for i=1:10^5
    if x_PSK(i)~=m_hat_PSK(i)
        number_of_error_PSK=number_of_error_PSK+1;
    end
end
```

Probability of error in M-PSK is calculated by again dividing total number of errors to total number of values transmitted.

```
P_error_PSK=number_of_error_PSK/10^5;
```

And, the theoretical error probability is calculated by the following formula which is applicable for both values of M:

```
P_error_PSK_theo=2*1/2*erfc(sqrt(2*SNR_PSK)*sin(pi/M)/sqrt(2));
```

Simulation error and theoretical error for 4-PSK are found as follows after running this simulation for constant SNR value of 10 dB:

```
P_error_PSK =

    0.0017

P_error_PSK_theo =

    0.0016
```

The same values respectively are found for 8-PSK as follows:

```
P_error_PAM =

    0.0875

P_error_PAM_theo =

    0.0870
```

Here, it can be observed that when number of levels increases, probability of error increases as well. It is obvious that probability of error decreases by increasing signal to noise ratio. Simulation error and theoretical error are found very close in this part as well.

Part II

In the second part, different from the first part, randomly generated symbols are convolved with a pulse and then white noise is added. After that, the noisy signal is filtered by the matched filter and sampled accordingly. Then, the decision rule is applied and number of errors are counted just like it was done in the first part.

a)

In part a of the part II, a normalized triangular pulse with T=10 is selected as follows:

```
pul=0:0.1:0.9;
pul_norm=pul/norm(pul);
```

Randomly generated and modulated with respect to binary PAM signal is convolved with this pulse to obtain the baseband signal.

```
r=filter(pul_norm,1,upsample(y,10));
```

After that, noise is added to this signal with respect to binary power and SNR value of 2 dB.

Then the signal is filtered with the matched filter produced from the selected pulse as follows:

```
matched=pul(end:-1:1);

r_n=filter(matched,1,r_noisy);
```

Then, it is sampled by 10 as follows:

```
for i=1:10^5
    r_new(i)=r_n(10*i);
end
```

After this filtering procedure, signal is received with respect to ML rule for the binary PAM and number of error is counted.

According to that counting, probability of error comes as follows for 2 dB SNR value:

```
P_error =

    0.0380
```

where theoretical error probability is calculated as follows:

```
P_error_theo =

    0.0375
```

Simulation error and theoretical error are nearly the same. Also, the results are consistent with the ones found in part I part a as expected.

b)

In part b of the part II, a normalized rectangular pulse with T=10 is used instead of triangular pulse.

```
pul=ones(1,10);
pul_norm=pul/norm(pul);
```

Again, randomly generated signal is convolved with this pulse.

```
r=filter(pul_norm,1,upsample(y,10));
```

Then, the noise is added to this signal and again noisy signal is passed through the matched filter.

```
matched=pul(end:-1:1);

r_n=filter(matched,1,r_noisy);
```

Then the sampling is performed.

```
for i=1:10^5
    r_new(i)=r_n(10*i);
end
```

After that, the signal is received by ML rule again and the number of error is counted.

Probability of error for 2 dB SNR value comes as follows:

```
P_error =

    0.0370
```

where theoretical error probability is calculated as follows:

```
P_error_theo =

    0.0375
```

Here, there is no difference with respect to part a as the matched filters in both parts have length of T=10 and are normalized to 1. As the sampling is performed by 10 as well, there comes no change.

c)

In part c, every step is same with the previous parts except the sampling step. In this part, a delay of delta T is introduced while performing sampling. That delayed sampling is performed as follows:

```
delT=1;%10/10%10/5%2*10/5
for i=1:10^5
    r_new(i)=r_n(10*i+delT);
end
```

When the delay is selected as 1, the probability of error is found as follows:

```
P_error =

    0.0665
```

When it is selected as 2, the probability of error is:

```
P_error =

    0.1125
```

And, when it is selected as 4, it is:

```
P_error =

    0.2457
```

where theoretical error probability is calculated for 2 dB SNR value is as follows:

```
P_error_theo =

    0.0375
```

Here, it can be observed that when delay is introduced probability of error increases. Actually, as the sampling was made with respect to T/10=10/10 in this case, when a delay of delT=10/10=1 is applied, probability of error increases almost by itself. When the delay is doubled, probability of error doubles, and when it is quadrupled, it quadruples with respect to error probability of unary delay.

Conclusion

In this project, BPSK, on-off signaling, M-ary PAM for M values of 4, 8, 16, and 4-PSK, 8-PSK modulation schemes are performed. Error probability is compared with theoretical calculations in each case. Also, the matched filter application is studied in this project. In every part, simulation error probabilities are highly fitted to their theoretical values. Therefore, it can be commented that the project's objective is satisfied.

Appendix

Part I

a)

```
%%
M=2;
x_binary=randi([0 M-1],10^5,1);
for i=1:10^5
    if x_binary(i)==0
        y_binary(i)=-1;
    else
        y_binary(i)=1;
    end
end
%y_binary=-1*pskmod(x_binary,M);
power_binary=1/M*sum((-1)^2+(1)^2);
SNR_binary_dB=9;%2%9
SNR_binary=10^(SNR_binary_dB/10);
SNR_binary_bit=SNR_binary/log2(M);
N0=power_binary/SNR_binary_bit;
%y_binary_noisy=awgn(y_binary,SNR_binary_dB);
for i=1:10^5
    y_binary_noisy(i)=y_binary(i)+randn(1)*sqrt(N0/2);
end

number_of_error_binary=0;
for i=1:10^5
    if real(y_binary_noisy(i))<0
        m_hat_binary(i)=0;
    else
        m_hat_binary(i)=1;
    end
    if x_binary(i)~=m_hat_binary(i)
        number_of_error_binary=number_of_error_binary+1;
    end
end

P_error_binary=number_of_error_binary/10^5;

P_error_binary_theo=1/2*erfc(sqrt(2*SNR_binary_bit)/sqrt(2));
```

b)

```
%%
M=2;
x_onoff=randi([0 M-1],10^5,1);
y_onoff=x_onoff;
power_onoff=1/M*sum((0)^2+(1)^2);
SNR_onoff_dB=12;%5%12
SNR_onoff=10^(SNR_onoff_dB/10);
SNR_onoff_bit=SNR_onoff/log2(M);
N0=power_onoff/SNR_onoff_bit;
%y_onoff_noisy=awgn(y_onoff,SNR_onoff_dB);
for i=1:10^5
    y_onoff_noisy(i)=y_onoff(i)+randn(1)*sqrt(N0/2);
```

```
end

number_of_error_onoff=0;
for i=1:10^5
    if real(y_onoff_noisy(i))<0.5
        m_hat_onoff(i)=0;
    else
        m_hat_onoff(i)=1;
    end
    if x_onoff(i)~=m_hat_onoff(i)
        number_of_error_onoff=number_of_error_onoff+1;
    end
end

P_error_onoff=number_of_error_onoff/10^5;

P_error_onoff_theo=1/2*erfc(sqrt(SNR_onoff_bit)/sqrt(2));
```

c)

```
%%
M=16;%4%8%16
x_PAM=randi([0 M-1],10^5,1);
for i=1:10^5
    for m=x_PAM(i)
        y_PAM(i)=2*m-M+1;
    end
end
%y_PAM=pammod(x_PAM,M);
s=0;
for i=1:M
    s=s+(2*(i-1)-M+1)^2;
end
power_PAM=1/M*s;
if M==4
    SNR_PAM_dB=13;%5%13
elseif M==8
    SNR_PAM_dB=17;%8%17
elseif M==16
    SNR_PAM_dB=22;%10%22
end
SNR_PAM=10^(SNR_PAM_dB/10);
SNR_PAM_bit=SNR_PAM/log2(M);
N0=power_PAM/SNR_PAM;
%y_PAM_noisy=awgn(y_PAM,SNR_PAM_dB);
for i=1:10^5
    y_PAM_noisy(i)=y_PAM(i)+randn(1)*sqrt(N0/2);
end

number_of_error_PAM=0;

for i=1:10^5
    for m=1:M-2
        if real(y_PAM_noisy(i))<-(M-2)
            m_hat_PAM(i)=0;
```

```
        elseif real(y_PAM_noisy(i))>=(2*m-M) && real(y_PAM_noisy(i))<(2*m-
M+2)
            m_hat_PAM(i)=m;
        elseif real(y_PAM_noisy(i))>=(M-2)
            m_hat_PAM(i)=M-1;
%         else
%             number_of_error_PAM=number_of_error_PAM+1;
        end
    end
end

for i=1:10^5
    if x_PAM(i)~=m_hat_PAM(i)
        number_of_error_PAM=number_of_error_PAM+1;
    end
end

P_error_PAM=number_of_error_PAM/10^5;

P_error_PAM_theo=(2*(M-1)/M)*1/2*erfc(sqrt(6*log2(M)/(M^2-
1)*SNR_PAM_bit)/sqrt(2));
```

d)

```
%%
M=8;%4%8
x_PSK=randi([0 M-1],10^5,1);
for i=1:10^5
    for m=x_PSK(i)
        y_PSK1(i)=cos(2*pi/M*m);
        y_PSK2(i)=sin(2*pi/M*m);
    end
end
s=0;
for i=1:M
    s=s+(cos(2*pi/M*(i-1)))^2+(sin(2*pi/M*(i-1)))^2;
end
power_PSK=1/M*s;
SNR_PSK_dB=10;
SNR_PSK=10^(SNR_PSK_dB/10);
SNR_PSK_bit=SNR_PSK/log2(M);
N0=power_PSK/SNR_PSK;
%y_PSK_noisy=awgn(y_PSK,SNR_PSK_dB);
for i=1:10^5
    y_PSK1_noisy(i)=y_PSK1(i)+randn(1)*sqrt(N0/2);
    y_PSK2_noisy(i)=y_PSK2(i)+randn(1)*sqrt(N0/2);
end
% for i=1:10^5
%     y_PSK(i)=x_PSK(i)+randn(1)*sqrt(N0/2);
%     for m=x_PSK(i)
%         y_PSK1_noisy(i)=cos(2*pi/M*y_PSK(i));
%         y_PSK2_noisy(i)=sin(2*pi/M*y_PSK(i));
%     end
% end

for i=1:10^5
```

```
    arc(i)=atan2(y_PSK2_noisy(i),y_PSK1_noisy(i));
    if arc(i)<-pi/M
        arcn(i)=arc(i)+2*pi;
    else
        arcn(i)=arc(i);
    end
end

number_of_error_PSK=0;

for i=1:10^5
    for m=1:M
        if real(arcn(i))>pi/M*(2*m-3) && real(arcn(i))<=pi/M*(2*m-1)
        %if real(1+cot(y_PSK2_noisy(i)/y_PSK1_noisy(i)))>pi/M*(2*m-3) &&
real(1+cot(y_PSK2_noisy(i)/y_PSK1_noisy(i)))<=pi/M*(2*m-1)
            m_hat_PSK(i)=m-1;
%         else
%             number_of_error_PSK=number_of_error_PSK+1;
        end
    end
end

for i=1:10^5
    if x_PSK(i)~=m_hat_PSK(i)
        number_of_error_PSK=number_of_error_PSK+1;
    end
end

P_error_PSK=number_of_error_PSK/10^5;

if M==4
    P_error_PSK_theo=1-(1-1/2*erfc(sqrt(2*SNR_PSK_bit)/sqrt(2)))^2;
    %P_error_PSK_theo=2*1/2*erfc(sqrt(SNR_PSK)/sqrt(2));
    %P_error_PSK_theo=2*1/2*erfc(sqrt(2*SNR_PSK_bit)/sqrt(2));
elseif M==8
    P_error_PSK_theo=2*1/2*erfc(sqrt(2*SNR_PSK)*sin(pi/M)/sqrt(2));
end
```

Part II

a)

```
%%
M=2;
x=randi([0 M-1],10^5,1);
for i=1:10^5
    if x(i)==0
        y(i)=-1;
    elseif x(i)==1
        y(i)=1;
    end
end
power=1/M*sum((-1)^2+(1)^2);
SNR_dB=2;
SNR=10^(SNR_dB/10);
SNR_bit=SNR/log2(M);
```

```
N0=power/SNR_bit;

% T=-1:2/10:1;
% w=2;
% pul=tripuls(T,w);
% %plot(T,pul);
pul=0:0.1:0.9;
pul_norm=pul/norm(pul);
matched=pul(end:-1:1);
r=filter(pul_norm,1,upsample(y,10));

for i=1:10^6
    r_noisy(i)=r(i)+randn(1)*sqrt(N0/2);
end

r_n=filter(matched,1,r_noisy);

% for i=1:10^5
%     r_new(i)=r_n(10*i);
% end

number_of_error=0;
for i=1:10^5
    if r_n(10*i)<0
        m_hat(i)=0;
    elseif r_n(10*i)>=0
        m_hat(i)=1;
    end
    if x(i)~=m_hat(i)
        number_of_error=number_of_error+1;
    end
end

P_error=number_of_error/10^5;

P_error_theo=1/2*erfc(sqrt(2*SNR_bit)/sqrt(2));
```

b)

```
%%
M=2;
x=randi([0 M-1],10^5,1);
for i=1:10^5
    if x(i)==0
        y(i)=-1;
    elseif x(i)==1
        y(i)=1;
    end
end
power=1/M*sum((-1)^2+(1)^2);
SNR_dB=2;
SNR=10^(SNR_dB/10);
SNR_bit=SNR/log2(M);
N0=power/SNR_bit;
```

```
% T=0:2/10:2;
% w=2;
% pul=rectpuls(T,w);
% %stem(T,pul);
pul=ones(1,10);
pul_norm=pul/norm(pul);
matched=pul(end:-1:1);
r=filter(pul_norm,1,upsample(y,10));

for i=1:10^6
    r_noisy(i)=r(i)+randn(1)*sqrt(N0/2);
end

r_n=filter(matched,1,r_noisy);

number_of_error=0;
for i=1:10^5
    if r_n(10*i)<0
        m_hat(i)=0;
    elseif r_n(10*i)>=0
        m_hat(i)=1;
    end
    if x(i)~=m_hat(i)
        number_of_error=number_of_error+1;
    end
end

P_error=number_of_error/10^5;

P_error_theo=1/2*erfc(sqrt(2*SNR_bit)/sqrt(2));
```

c)

```
%%
M=2;
x=randi([0 M-1],10^5+5,1);
for i=1:10^5+5
    if x(i)==0
        y(i)=-1;
    elseif x(i)==1
        y(i)=1;
    end
end
power=1/M*sum((-1)^2+(1)^2);
SNR_dB=2;
SNR=10^(SNR_dB/10);
SNR_bit=SNR/log2(M);
N0=power/SNR_bit;

delT=4;%10/10%10/5%2*10/5
% T=-1:delT:1;
% w=2;
% pul=tripuls(T,w);
% %plot(T,pul);
pul=0:0.1:0.9;
pul_norm=pul/norm(pul);
```

```
matched=pul(end:-1:1);
r=filter(pul_norm,1,upsample(y,10));

for i=1:10^6+50
    r_noisy(i)=r(i)+randn(1)*sqrt(N0/2);
end

r_n=filter(matched,1,r_noisy);

number_of_error=0;
for i=1:10^5
    if r_n(10*i+delT)<0
        m_hat(i)=0;
    elseif r_n(10*i+delT)>=0
        m_hat(i)=1;
    end
    if x(i)~=m_hat(i)
        number_of_error=number_of_error+1;
    end
end

P_error=number_of_error/(10^5+5);

P_error_theo=1/2*erfc(sqrt(2*SNR_bit)/sqrt(2));
```