WIRELESS COMMUNICATION PRACTISE

EXPERIMENT - 01 (BER of BPSK in fadding channel)

Manas Kumar Mishra (ESD18I011)

18-JANUARY-2022

Lab In-charge:	Dr. Premkumar Karumbu
Organization:	IIITDM Kancheepuram

1 Aim:

To analyze the fadding channel. To evaluate performance of BPSK from BER (probability of error) vs SNR ratio in fadding channel. Explore the other possibilities with BPSK in fadding channel.

2 Software:

To perform this experiment, c++ language and gnuplot (open source) have been used. Data have been generated by the program in c++ language and plots are made by gnuplot.

3 Theory

3.1 Wireless channel and model

3.1.1 Wireless and wired

From basic understanding of electrical signal and electromagnetic waves, data (*modulated signal*) propagate in wired medium, that implies it always has certain path and direction inside the medium. Unlike wired medium, in wireless, data (*modulated signal*) radiates into the space (free space/air), that implies it do not have fixed path and direction inside the medium. This issue creates problems for receiver.

3.1.2 Multipath

Consider no noise case, if transmitter transmits $x(t)$ signal in wireless channel, then receiver receives $y(t)$ as

$$y(t) = \sum_i \alpha_i(t) x(t - \tau_i(t))$$

where, α_i and τ_i are amplitudes and time delays respectively, of the received signal through i^{th} path. This is the basic model for multi-path issue. (*Note:- Here, transmitter and receiver are assumed to be in non-motion, hence, no Doppler shift has been considered here*)

3.1.3 Multipath modelling

Consider transmitted signal $x(t)$ as cosine signal (most basic signal).

$$
\begin{aligned}
x(t) &= cos(\omega t) \\
x(t) &= Re\{\ e^{j\omega t}\} && \text{(Re implies real part)} \\
\implies \quad y(t) &= \sum_i Re\{\alpha_i(t)\ e^{j\omega(t-\tau_i(t))}\} \\
&= \sum_i Re\{\alpha_i(t)e^{-j\omega\tau_i(t)}\ e^{j\omega t}\} && (1)
\end{aligned}
$$

Here, $\alpha_i e^{-j\omega\tau_i(t)}$ can be considered as the overall impairment to the transmitted signal by wireless channel. Consider channel is slow changing channel, that implies environment of channel is changing slowly with respect to change in signal. In other words, bit duration is very small as compare to rate of change in channel. For example if data rate is 1Mbps implies 10^{-6}sec is bit duration (after that bit may change hence signal may change) but dynamic elements in channel (so called reflectors) can not even move in 10^{-6}sec. This type of channel is known as slow fading channel.
Due to slow fadding channel assumption, $\alpha_i(t)$ and $\tau_i(t)$ can be considered as constant in every fix sized time slot.

$$
\begin{aligned}
y(t) &= \sum_i Re\{\alpha_i(t)e^{-j\omega\tau_i(t)}\ e^{j\omega t}\} && \text{(from 1)} \\
y_k(t) &= \sum_i Re\{\alpha_i e^{-j\omega\tau_i}\ e^{j\omega t}\} && \text{(due to slow fadding assumption)} \\
consider, \quad -j\omega\tau_i &= \phi_i && (\phi_i \in [-\pi,\pi]) \\
&= \sum_i Re\{\alpha_i e^{\phi_i}\ e^{j\omega t}\} \\
&= Re\{\sum_i \alpha_i e^{\phi_i}\ e^{j\omega t}\}
\end{aligned}
$$

In practical case, there are infinite number of reflectors, that implies i goes from zero to infinity. If $\alpha_i e^{\phi_i}$ consider as a independent and identical random variable for every sample of $y(t)$ like y_k, then it would converge to the Gaussian random variable. (Central limit theorem).

$$\sum_i \alpha_i e^{\phi_i} = R + jI \tag{2}$$

Since, in equation-2 LHS is complex quantity, therefore RHS is modelled as complex random variable, where real and imaginary part is gaussian random variable.

$$
\begin{aligned}
R &= \sum_i \alpha_i cos(\phi_i) \\
I &= \sum_i \alpha_i sin(\phi_i) \\
R &\sim \mathcal{N}(0,\sigma^2) \\
I &\sim \mathcal{N}(0,\sigma^2)
\end{aligned}
$$

Finally, received signal in k^{th} sample of $y(t)$ can be written as

$$
\begin{aligned}
y_k(t) &= Re\{\sum_i \alpha_i e^{\phi_i}\, e^{j\omega t}\} \\
y_k(t) &= Re\{(R+jI)\, e^{j\omega t}\} \\
y_k(t) &= R\, \cos(\omega t) - I\, \sin(\omega t)
\end{aligned}
$$

By complex analysis, one can write $y_k(t)$ is polar form.

$$
\begin{aligned}
y_k(t) &= Re\{M_k e^{j\theta_k}\, e^{j\omega t}\} \\
y_k(t) &= M_k\, Re\{e^{j\theta_k} e^{j\omega t}\}
\end{aligned}
$$

To remove the phase term, there is process called co-phasing. After co-phasing.

$$
\begin{aligned}
y_k(t) &= M_k\, Re\{e^{j\omega t}\} \\
y_k(t) &= M_k\, x_k(t) && (k^{th} \text{ sample of x(t)}) \\
Y_k &= M_k\, X_k && (3) \\
where, \quad M_k &= \sqrt{R^2 + I^2}
\end{aligned}
$$

Now, problem is to find statistical properties of M_k

3.1.4 Rayleigh distribution

Consider X and Y are two independent zero mean but same variance gaussian random variables.

$$
\begin{aligned}
& X \sim \mathcal{N}(0, \sigma^2) \\
& Y \sim \mathcal{N}(0, \sigma^2) \\
& X \perp\!\!\!\perp Y \\
let \quad & Z = X + jY \ = M\, e^{j\theta} \\
where, \quad & M = \sqrt{X^2 + Y^2} \\
& \theta = tan^{-1}\left(\frac{Y}{X}\right) \\
\because\, & X \perp\!\!\!\perp Y \\
\therefore\, f_{X,Y}(x,y) &= f_X(x)\, f_Y(y) \\
&= \frac{1}{2\pi\sigma^2} \exp\left(\frac{-1}{2\sigma^2}(x^2 + y^2)\right)
\end{aligned}
$$

Convert the $f_{X,Y}(x,y)$ into $f_{M,\theta}(m, \Theta)$, for that use concepts of transformation of random variables, jacobian matrix.

$$
\begin{aligned}
consider\ \ \theta &= tan^{-1}\left(\frac{Y}{X}\right) \\
\implies Y &= X\ tan(\theta) \\
\because,\ \ M &= \sqrt{X^2+Y^2} && \text{(M}\geq\text{0)} \\
\therefore,\ \ M &= X\ sec(\theta) && \text{(X and } sec(\theta)\geq 0 \text{ or X and } sec(\theta)\leq 0\text{)} \\
\implies\ \ X &= M\ cos(\theta) \\
\implies\ \ Y &= M\ sin(\theta)
\end{aligned}
$$

Note:- This is only valid of principle values of $tan^{-1}(x)\ \ \forall\ \ x$ *i.e.* $(-\pi/2,\pi/2)$ *and M≥0, But X and Y can be any value positive and negative. This gives two solutions.*
First as $X = M\ cos(\theta)$ and $Y = M\ sin(\theta)$
second as $X = -M\ cos(\theta)$ and $Y = -M\ sin(\theta)$
Now jacobian

$$
\begin{aligned}
|J| &= \begin{vmatrix} \frac{\partial X}{\partial M} & \frac{\partial X}{\partial \theta} \\ \frac{\partial Y}{\partial M} & \frac{\partial Y}{\partial \theta} \end{vmatrix} \\
|J| &= \begin{vmatrix} cos(\theta) & -Msin(\theta) \\ sin(\theta) & Mcos(\theta) \end{vmatrix} \\
|J| &= M
\end{aligned}
$$

Both solution leads to same jacobian value. That mean, jacobian for entire principle value of θ is 2M.
Transformation of random variable

$$
\begin{aligned}
f_{M,\theta}(m,\Theta) &= |J| f_{X,Y}(x,y)|_{X=Mcos(\theta),Y=Msin(\theta)} \\
f_{M,\theta}(m,\Theta) &= \frac{2M}{2\pi\sigma^2}\exp\left(\frac{-1}{2\sigma^2}(M^2)\right)
\end{aligned}
$$

$$
\begin{aligned}
f_M(m) &= \int_{-\pi/2}^{\pi/2}\frac{2M}{2\pi\sigma^2}\exp\left(\frac{-1}{2\sigma^2}(M^2)\right)d\theta \qquad M\geq 0 \\
f_M(m) &= \frac{M}{\sigma^2}\exp\left(\frac{-M^2}{2\sigma^2}\right) \qquad M\geq 0
\end{aligned}
$$

This distribution is known as Rayleigh distribution of parameter σ^2. Similarly,

$$
\begin{aligned}
f_\theta(\Theta) &= \int_0^\infty \frac{2M}{2\pi\sigma^2}\exp\left(\frac{-1}{2\sigma^2}(M^2)\right)dM \\
&= \frac{1}{\pi\sigma^2}\int_0^\infty M\ \exp\left(\frac{-M^2}{2\sigma^2}\right)dM \\
let\ \ u &= \frac{M^2}{2\sigma^2} \implies du = M/\sigma^2 \\
&= \frac{1}{\pi}\int_0^\infty \exp(-u)\,du \\
&= \frac{1}{\pi}
\end{aligned}
$$

That means $\theta \sim uniform(-\pi/2, \pi/2)$
Since $\theta \in (-\pi, \pi)$, extend the uniformity from $(-\pi/2, \pi/2)$ to $(-\pi, \pi)$.

$$f_\theta(\Theta) = \frac{1}{2\pi} \quad ; \quad \theta \in (-\pi, \pi)$$

Warning:
As a responsible engineer and author, i must confess that i do not know whether can we extend the uniform random variable. Maybe whatever i did for θ in last line is wrong. i will not use this in further parts of this report.

$$\begin{aligned} f_{M,\theta}(m,\Theta) &= \left(\frac{1}{\pi}\right)\frac{M}{\sigma^2}\exp\left(\frac{-M^2}{2\sigma^2}\right) \quad ; \quad M \geq 0 \\ &= f_\theta(\Theta) f_M(m) \\ \therefore, \;\; \theta &\perp\!\!\!\perp M \end{aligned}$$

For wireless channel, M is Rayleigh with parameter $1/2$.

$$\begin{aligned} Y_k &= M_k X_k \\ M_k &\sim Rayleigh(1/2) \end{aligned}$$

3.2 BPSK in Rayleigh Fading channel

3.2.1 ML detector

$$\begin{aligned} f_{Y|B,M}(y|0,m) &\gtrless^{B^o=0}_{B^o=1} f_{Y|B,M}(y|1,m) \\ f_Y(m\sqrt{E}+\eta = y) &\gtrless^{B^o=0}_{B^o=1} f_Y(-m\sqrt{E}+\eta = y) \\ f_N(\eta = y - m\sqrt{E}) &\gtrless^{B^o=0}_{B^o=1} f_N(\eta = y + m\sqrt{E}) \\ \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{-1}{2\sigma^2}(y-m\sqrt{E})^2\right) &\gtrless^{B^o=0}_{B^o=1} \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{-1}{2\sigma^2}(y+m\sqrt{E})^2\right) \\ (y-m\sqrt{E})^2 &\gtrless^{B^o=1}_{B^o=0} (y+m\sqrt{E})^2 \\ 2y\, m\sqrt{E} &\gtrless^{B^o=0}_{B^o=1} 0 \\ y &\gtrless^{B^o=0}_{B^o=1} 0 \end{aligned}$$

Ml detector is same as normal BPSK detector.

3.2.2 Probability of error

$$
\begin{aligned}
P(Error) &= \sum_{i=0}^{1} P(Error, B=i) \\
&= \sum_{i=0}^{1} P(Error|B=i)P(B=i) \\
consider\ \ P(Error|B=0) &= P(B^o=1|B=0) \\
&= \int_0^\infty P(B^o=1|B=0,M=m)f_M(m)dm \\
&= \int_0^\infty P(y<0|B=0,M=m)f_M(m)dm \\
&= \int_0^\infty P(m\sqrt{E}+\eta<0)f_M(m)dm \\
&= \int_0^\infty P(\eta<-m\sqrt{E})f_M(m)dm \\
&= \int_{m=0}^\infty \int_{x=m\sqrt{E}}^\infty \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{x^2}{2\sigma^2}\right) dx\, 2m\, exp(-m^2)dm \\
Change\ \ the\ \ variables\ \ &= \int_{x=0}^\infty \int_{m=0}^{x/\sqrt{E}} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{x^2}{2\sigma^2}\right)\ 2m\, exp\left(-m^2\right) dm\, dx \\
&= \int_{x=0}^\infty \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{x^2}{2\sigma^2}\right)\ 2\int_{m=0}^{x/\sqrt{E}}\ m\, exp\left(-m^2\right) dm\, dx \\
&= \int_{x=0}^\infty \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{x^2}{2\sigma^2}\right)\ \left[1-exp\left(-\frac{x^2}{E}\right)\right] dx \\
&= \frac{1}{2} - \int_{x=0}^\infty \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{x^2}{2\sigma^2}-\frac{x^2}{E}\right) dx \\
&= \frac{1}{2} - \int_{x=0}^\infty \frac{1}{\sqrt{2\pi}\sigma} exp\left(-x^2\left[\frac{1}{2\sigma^2}+\frac{1}{E}\right]\right) dx \\
let\ \frac{s}{\sqrt{2}} &= x\sqrt{\frac{1}{2\sigma^2}+\frac{1}{E}} \\
dx &= \frac{ds}{\sqrt{2}\sqrt{\frac{1}{2\sigma^2}+\frac{1}{E}}} \\
&= \frac{1}{2} - \int_{x=0}^\infty \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{s^2}{2}\right) \frac{ds}{\sqrt{2}\sqrt{\frac{1}{2\sigma^2}+\frac{1}{E}}} \\
&= \frac{1}{2} - \frac{1}{2\sqrt{2}\sigma\left(\sqrt{\frac{1}{2\sigma^2}+\frac{1}{E}}\right)} \\
&= \frac{1}{2}\left[1-\frac{1}{\left(\sqrt{1+\frac{2\sigma^2}{E}}\right)}\right]
\end{aligned}
$$

Now assume, $\gamma = \frac{E}{\sigma^2}$ Hence,

$$P(B^o = 1|B = 0) = \frac{1}{2}\left[1 - \frac{1}{\left(\sqrt{1+\frac{2}{\gamma}}\right)}\right]$$

$$similarly, \;\; P(B^o = 0|B = 1) = \frac{1}{2}\left[1 - \frac{1}{\left(\sqrt{1+\frac{2}{\gamma}}\right)}\right]$$

$$\therefore \;\; P(Error) = \frac{1}{2}\left[1 - \frac{1}{\left(\sqrt{1+\frac{2}{\gamma}}\right)}\right]$$

For large values of γ, approximation is possible is in P(Error)

$$\left(\sqrt{1+\frac{2}{\gamma}}\right) \approx 1 + \frac{1}{\gamma} \qquad \text{(Using taylor series expension)}$$

$$\begin{aligned} P(Error) &\approx \frac{1}{2}\left(1 - \frac{1}{1+\frac{1}{\gamma}}\right) \\ &\approx \frac{1}{2}\left(1 - \frac{\gamma}{\gamma+1}\right) \\ &\approx \frac{1}{2(1+\gamma)} && \text{(a simple approximation for large } \gamma\text{)} \\ &\approx \frac{1}{2\gamma} && \text{(One more simple approximation)} \end{aligned}$$

4 Pseudo code

- S1. Generate random bit stream of length equal to million.
- S2. Map 0 to $-\sqrt{E}$ and 1 to $\sqrt{E}$.
- S3. Generate two gaussian noise vector of zero mean and variance as half, treat as real part and imaginary part of the complex random variable.
- S4. Compute Rayleigh coefficient by using above random variable.
- S5. Generate gaussian noise component.
- S6. Apply channel model equation $Y_k = H_k X_K + N_k$ and get the received signal.
- S7. Apply ML rule and decode the received signal.
- S8. Count errors
- S9. Repeat S1 to S8 for many SNR values.
- S10. Store data (SNR, Error count) in .dat file.
- S11. Compute three theoretical probability of error (one exact and two approx value). Store into .dat file
- S12. Plot the result using gnuplot in log scale.

5 Code and Results

Main code for Rayleigh fading channel, for bpsk a header file has been designed to reduce the length of the program.

5.1 Code for Rayleigh fading channel

```
////////////////////////////////////////////////////////////
//////////////////////////////////////
// Author:- MANAS KUMAR MISHRA
// Organisation:- IIITDM KANCHEEPURAM
// Topic:- Performance of BPSK in Rayleigh fading channel
////////////////////////////////////////////////////////////
//////////////////////////////////////

#include <iostream>
#include <cmath>
#include <iterator>
#include <random>
#include <chrono>
#include <time.h>
#include <fstream>
#include "BPSK.h"

#define one_million 1000000

using namespace std;


// Function for printing the vector on the console output.
void PrintVectorDouble(vector<double> vectr)
{
    std::copy(begin(vectr), end(vectr),  std::ostream_iterator<double>(std::cout, "
"));
    cout<<endl;
}


// Function for generating binary bits at source side. Each bit is equiprobable.
// Input is nothing
// Output is a vector that contains the binary bits of length one_million*1.
vector<double> sourceVector()
{
    vector<double> sourceBits;

    // Use current time as seed for random generator
    srand(time(0));

    for(int i = 0; i<one_million; i++){
        sourceBits.insert(sourceBits.end(), rand()%2);
    }

    return sourceBits;
```

```
}


// Function for Rayleigh fading cofficients
// Inputs are two vectors one is the gaussion noise as real
//part and second as Gaussian noise as imazinary part
// Output is a vector that contain rayliegh noise coff, sqrt(real_part^2 + imz_part^2)
vector<double> RayleighFadingCoff(vector<double> realGaussian,
                                  vector<double> ImziGaussian)
{
    vector<double> rayleighNoise;
    double temp;

    for(int times=0; times<realGaussian.size(); times++){

        temp = sqrt(pow(realGaussian[times], 2)+pow(ImziGaussian[times], 2));
        rayleighNoise.insert(rayleighNoise.end(), temp);
    }

    return rayleighNoise;
}


// function for modelling wireless channel
// Inputs are the transmitted signal (X_k), rayleigh
//cofficient (H_k) and AWGN (N_k) component
// Output is the H_k*X_k+N_k
vector<double> ChannelOperation(vector<double> TransSignal,
                                vector<double> RayleighCoff,
                                vector<double> gnoise)
{
    vector<double> channelResult;
    double temp;
    for(int i =0; i<TransSignal.size(); i++){
        temp = (RayleighCoff[i]*TransSignal[i]) + gnoise[i];
        channelResult.insert(channelResult.end(), temp);
    }

    return channelResult;
}


// Function to count number of errors in the received bits.
// Inputs are the sourcebits and decodedbits
// OUtput is the number of error in received bits.
// error: if sourcebit  != receivebit
double errorCalculation (vector<double> sourceBits, vector<double> decodedBits)
{
    double countError =0;
    for(int i =0; i<sourceBits.size();i++){
        if(sourceBits[i]!= decodedBits[i]){
            countError++;
        }
    }

    return countError;
```

```
}


// Function to store the data in the file (.dat)
// Input is the SNR per bit in dB and calculated probability of error
// Output is the nothing but in processing it is creating a file and writing data into it.
void datafile(vector<double> xindB, vector<double> Prob_error)
{
    ofstream outfile;

    outfile.open("FaddingChan1.dat");

    if(!outfile.is_open()){
        cout<<"File opening error !!!"<<endl;
        return;
    }

    for(int i =0; i<xindB.size(); i++){
        outfile<< xindB[i] << " "<<"\t"<<" "<< Prob_error[i]<< endl;
    }

    outfile.close();
}



vector<double> Errorfunction(vector <double> SNR_dB)
{
    vector <double> Qvalue;

    double po, normalValue, inter;
    for (int k =0; k<SNR_dB.size(); k++){
        normalValue = pow(10, (SNR_dB[k]/10));
        // inter = 1+(2/normalValue);
        // inter = sqrt(inter);
        // po = 0.5*(1-(1/inter));
        po = 1/(2*(normalValue));
        Qvalue.insert(Qvalue.end(), po);
    }

    return Qvalue;
}

// Function to store the data in the file (.dat)
// Input is the SNR per bit in dB and calculated Qfunction values
// Output is the nothing but in processing it is creating a file and writing data into it.
void ErrorValueInFile(vector <double> SNR, vector <double> Qvalue)
{
    ofstream outfile;

    outfile.open("FaddingChan_Qvalue1.dat");

    if(!outfile.is_open()){
        cout<<"File opening error !!!"<<endl;
        return;
    }
```

```

    for(int i =0; i<SNR.size(); i++){
        outfile<< SNR[i] << "  "<<"\t"<<"  "<< Qvalue[i]<< endl;
    }

    outfile.close();
}


int main()
{
    // source defination
    vector<double> sourceBits;

    // Mapping of bits to symbols;
    vector<double> transmittedSymbol;

    // Noise definition
    vector<double> gnoise;

    vector<double> realGaussian;
    vector<double> imziGaussian;

    vector<double> RayleighNoise;

    vector<double> receiveSignal;

    vector<double> decodedBits;

    double sigmaSquare = 0.5;
    double stddevRayleigh = sqrt(sigmaSquare);
    double N_o =4;
    double p, stdnoise;
    double counterror, P_error;

    vector<double> SNR_dB;
    for(float i =0; i<=35; i=i+0.5)
    {
        SNR_dB.insert(SNR_dB.end(), i);
    }

    vector<double> energyOfSymbol;
    vector<double> Prob_error;
    double normalValue;

    for(int i =0; i<SNR_dB.size(); i++){

        normalValue = pow(10, (SNR_dB[i]/10));
        energyOfSymbol.insert(energyOfSymbol.end(), N_o*normalValue);
    }


    for(int step =0; step <energyOfSymbol.size(); step++){

        sourceBits = sourceVector();

```

```
        transmittedSymbol=bit_maps_to_symbol_of_energy_E(sourceBits, energyOfSymbol[step],
                                                         one_million);

        realGaussian = GnoiseVector(0.0, stddevRayleigh, one_million);
        imziGaussian = GnoiseVector(0.0, stddevRayleigh, one_million);

        RayleighNoise = RayleighFaddingCoff(realGaussian, imziGaussian);

        stdnoise = sqrt(N_o);
        gnoise = GnoiseVector(0.0, stdnoise, one_million);

        receiveSignal = ChannelOperation(transmittedSymbol, RayleighNoise, gnoise);

        decodedBits = decisionBlock(receiveSignal);
        counterror = errorCalculation(sourceBits, decodedBits);

        P_error = counterror/one_million;

        Prob_error.insert(Prob_error.end(), P_error);

        cout<<endl;

        cout<<"Energy of symbol     : "<<energyOfSymbol[step]<<endl;
        cout<<"Count errors         : "<<counterror<<endl;
        cout<<"Probability of error : "<<P_error<<endl;

        cout<<endl;
    }

    datafile(SNR_dB, Prob_error);
    vector<double> qvalue = Errorfunction(SNR_dB);
    ErrorValueInFile(SNR_dB, qvalue);

    return 0;
}
```

Code for BPSK header file

```
////////////////////////////////////////////////////////////
///////////////////////////////////////
// Author:- MANAS KUMAR MISHRA
// Organisation:- IIITDM KANCHEEPURAM
// Topic:- header file BPSK scheme
////////////////////////////////////////////////////////////
///////////////////////////////////////
#include <cmath>
#include <iterator>
#include <random>
#include <chrono>
#include <time.h>

using namespace std;

// Function for mapping bits to symbol.
// Input is a binary bit vector. Here 0---> -(sqrt(Energy)) and 1---> (sqrt(Energy))
// Output is a vector that contains transmitted symbols.
```

```
vector<double> bit_maps_to_symbol_of_energy_E(vector<double> sourceBits,
                                              double energyOfSymbol,
                                              const int one_million)
{
    vector<double> transmittedSymbol;

    for(int i=0; i<one_million; i++){
        if(sourceBits[i]== 0){
            transmittedSymbol.insert(transmittedSymbol.end(), -sqrt(energyOfSymbol));
        }
        else{
            transmittedSymbol.insert(transmittedSymbol.end(), sqrt(energyOfSymbol));
        }

    }

    return transmittedSymbol;
}


// Function for generating random noise based on gaussian distribution N(mean, variance).
// Input mean and standard deviation.
// Output is the vector that contain gaussian noise as an element.
vector<double> GnoiseVector(double mean, double stddev, const int one_million)
{
    std::vector<double> data;

    // construct a trivial random generator engine from a time-based seed:
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator (seed);

    std::normal_distribution<double> dist(mean, stddev);

    // Add Gaussian noise
    for (int i =0; i<one_million; i++) {
        data.insert(data.end(),dist(generator));
    }

    return data;
}


// Function for modeling additive channel.Here gaussian noise adds to the transmitted bit.
// Inputs are the transmitted bit and gaussian noise with mean 0 and variance 1.
// Output is the receive bits.
vector<double> receiveBits(vector<double> transBit, vector<double> gnoise)
{
    vector<double> recievebits;

    for(int j =0; j<transBit.size(); j++){
        recievebits.insert(recievebits.end(), transBit[j]+gnoise[j]);
    }

    return recievebits;

}
```

```cpp


// Function for deciding the bit value from the received bits
// Input is the received bits.
// Output is the decoded bits.
// Decision rule :- if receiveBit >0 then 1 otherwise 0 (simple Binary detection)
vector<double> decisionBlock(vector<double> receiveBits)
{
    vector<double> decodedBits;

    for(int i =0; i<receiveBits.size(); i++){
        if(receiveBits[i]>0){
            decodedBits.insert(decodedBits.end(), 1);
        }
        else{
            decodedBits.insert(decodedBits.end(), 0);
        }
    }

    return decodedBits;
}
```

5.2 Results

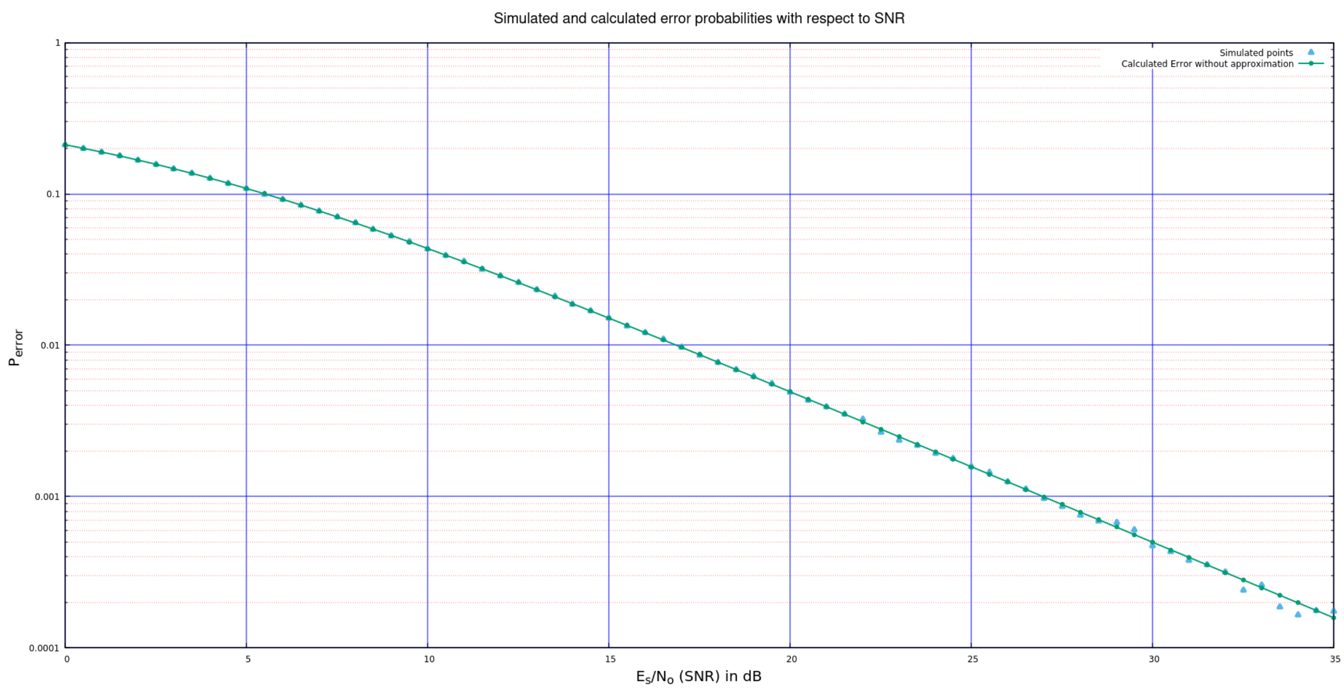

Figure 1: Simulated and calculated (theoretical) results

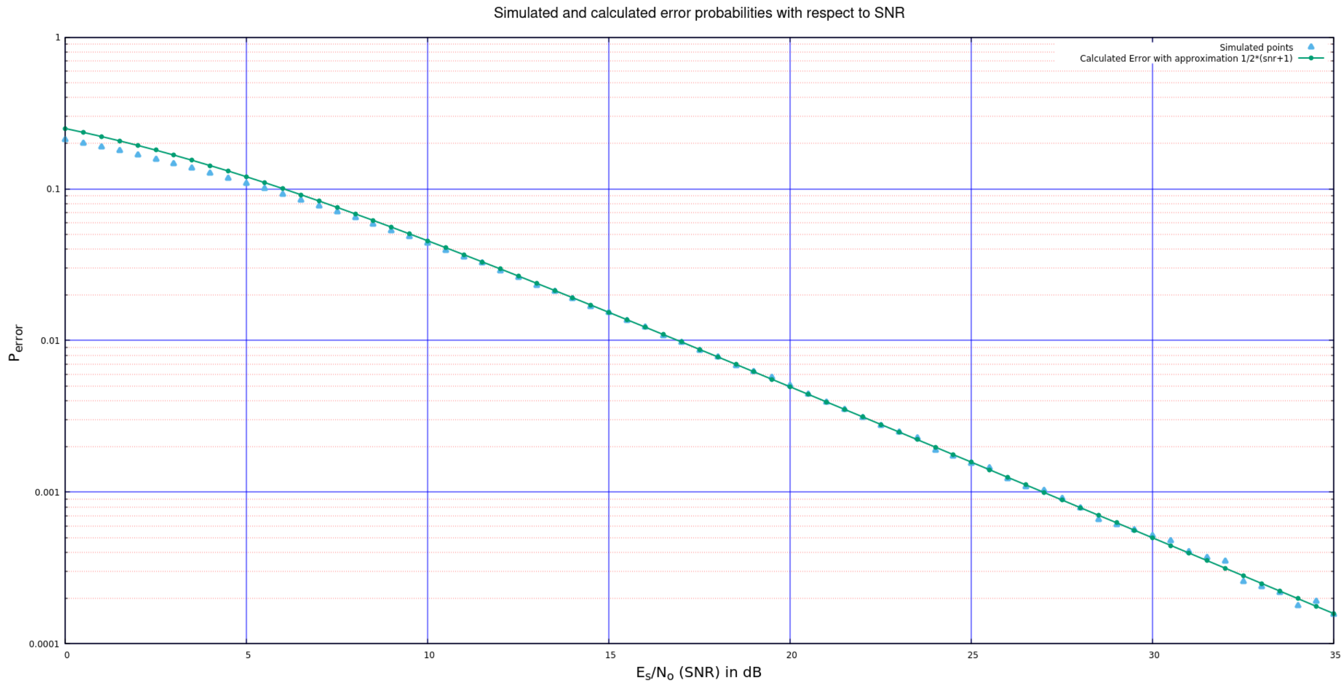

Figure 2: Simulated and calculated results (using approximation)

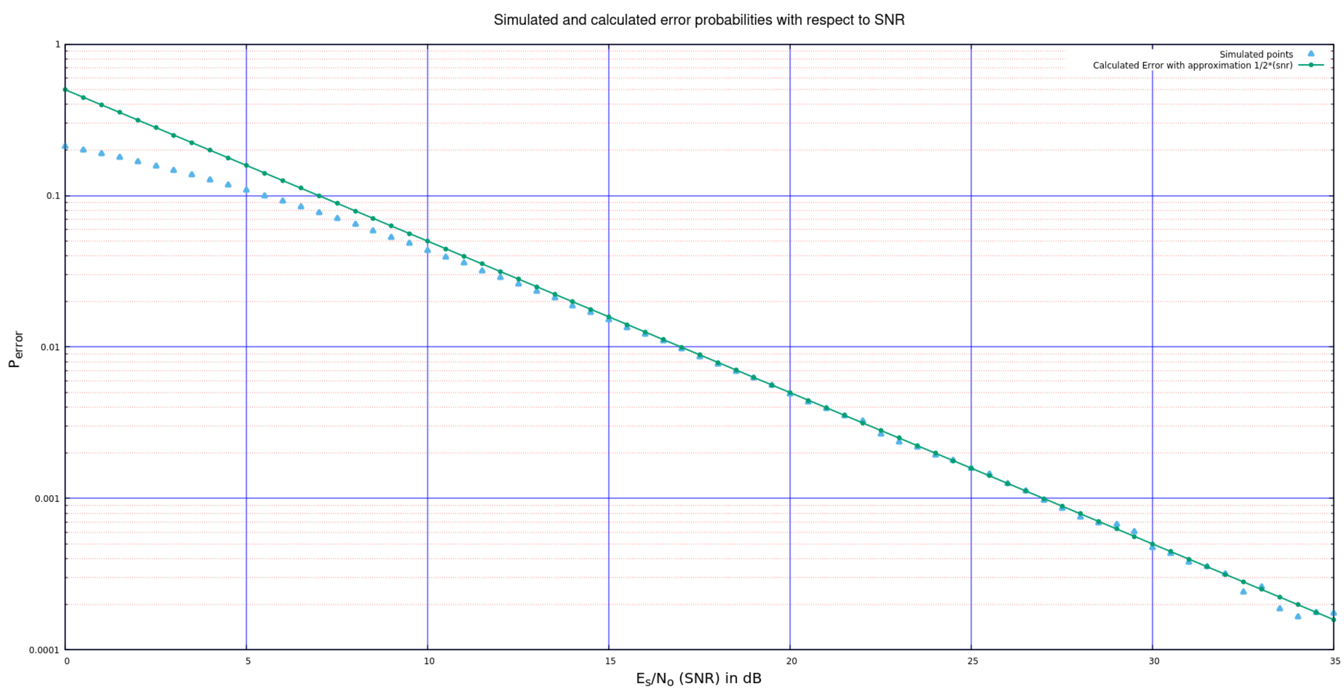

Figure 3: Simulated and calculated results (using approximation)

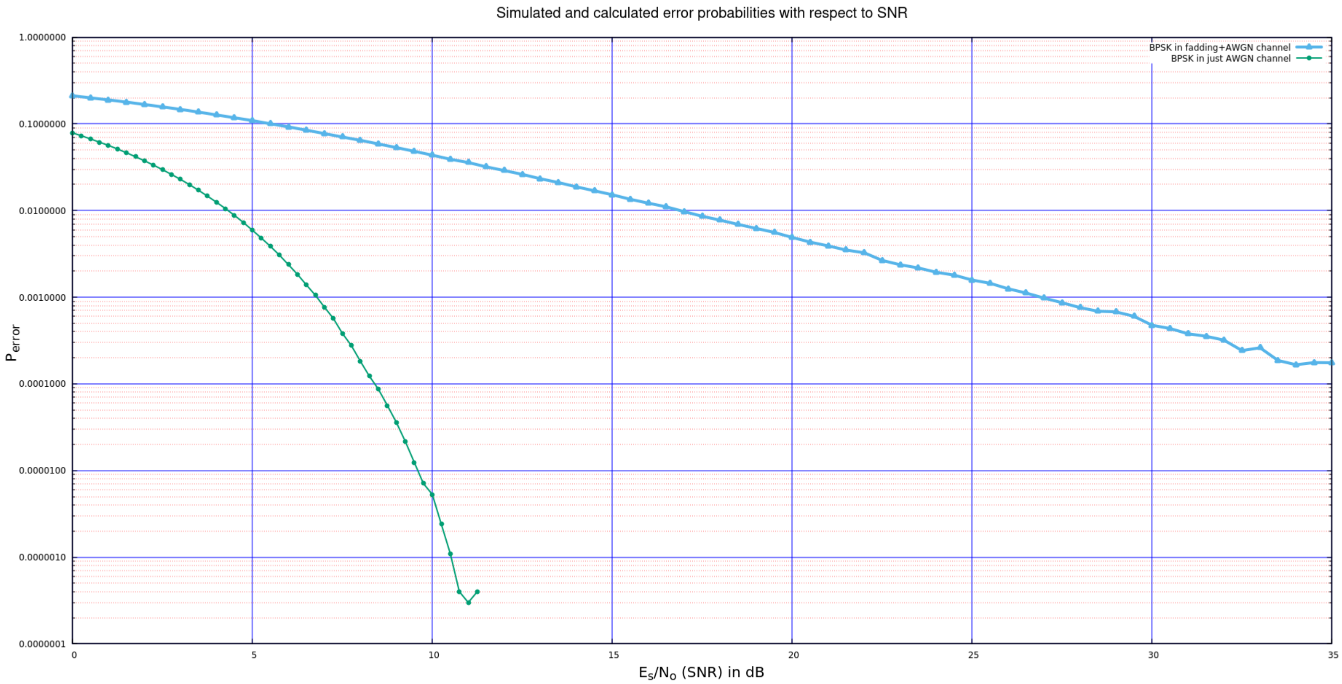

Figure 4: BPSK in just AWGN and BPSK in fadding channel

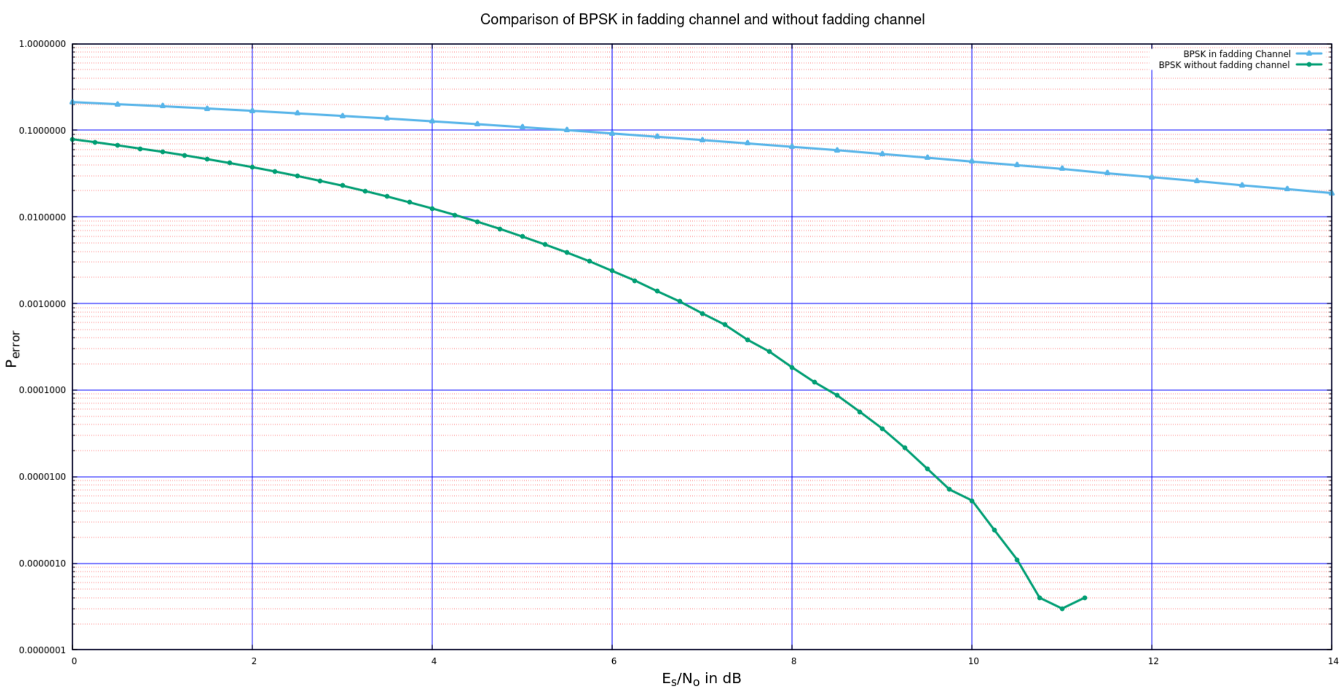

Figure 5: BPSK in just AWGN and BPSK in fadding channel for low values of SNR

5.3 Something extra

As anyone can see, fadding channel has very poor probability of error for given SNR value, as compare to normal AWGN channel. Now curiosity is that how to go close to the AWGN channel values in fadding

channel. In other words, is there any way to shift fadding channel probability of error close to AWGN channel value.
Answer is yes, it is possible with loss of data rate. If transmitter transmits same bit L consecutive times (it need not be consecutive, rather transmitter can transmit sequence of bits L times repeatedly), then receiver receive the same data on different signals. Due to that receiver can improve the probability of error.
Based on above hypothesis, given code has been modified for same consecutive two and three bits ($L = 2$ *and* $L = 3$). Results are given below. And it satisfies the above hypothesis.

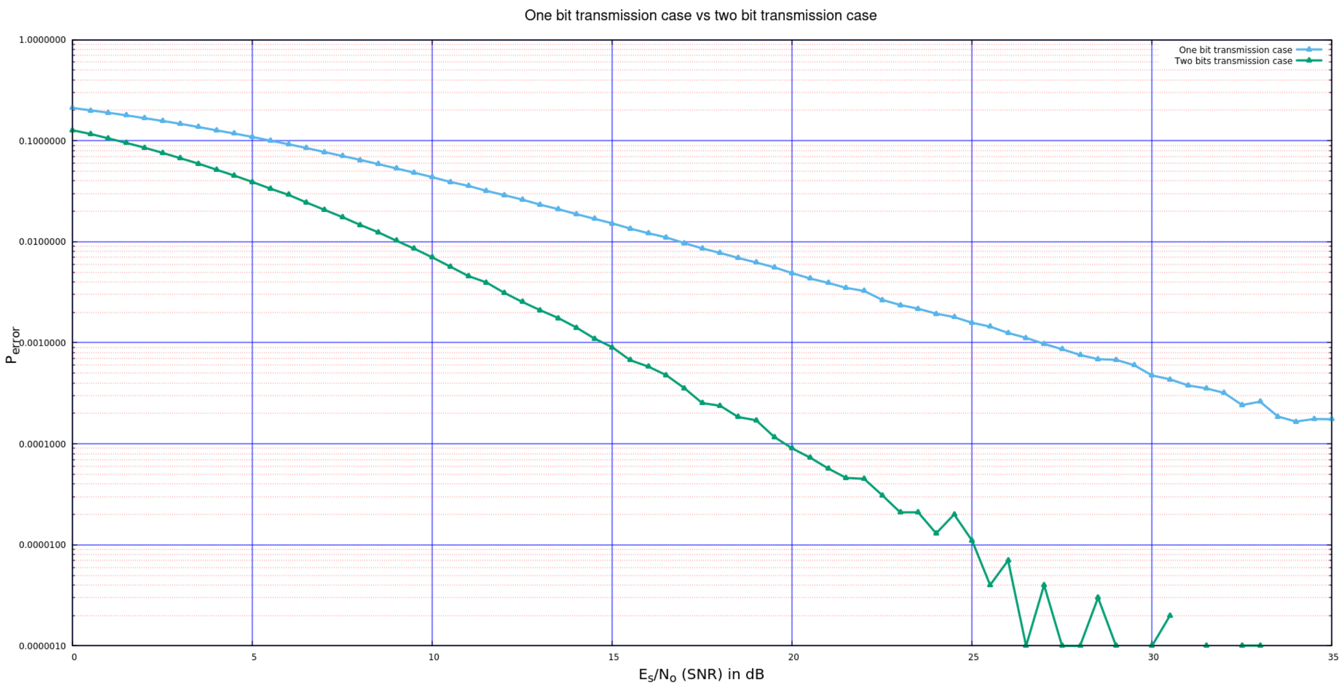

Figure 6: Only one bit and two same bits transmission

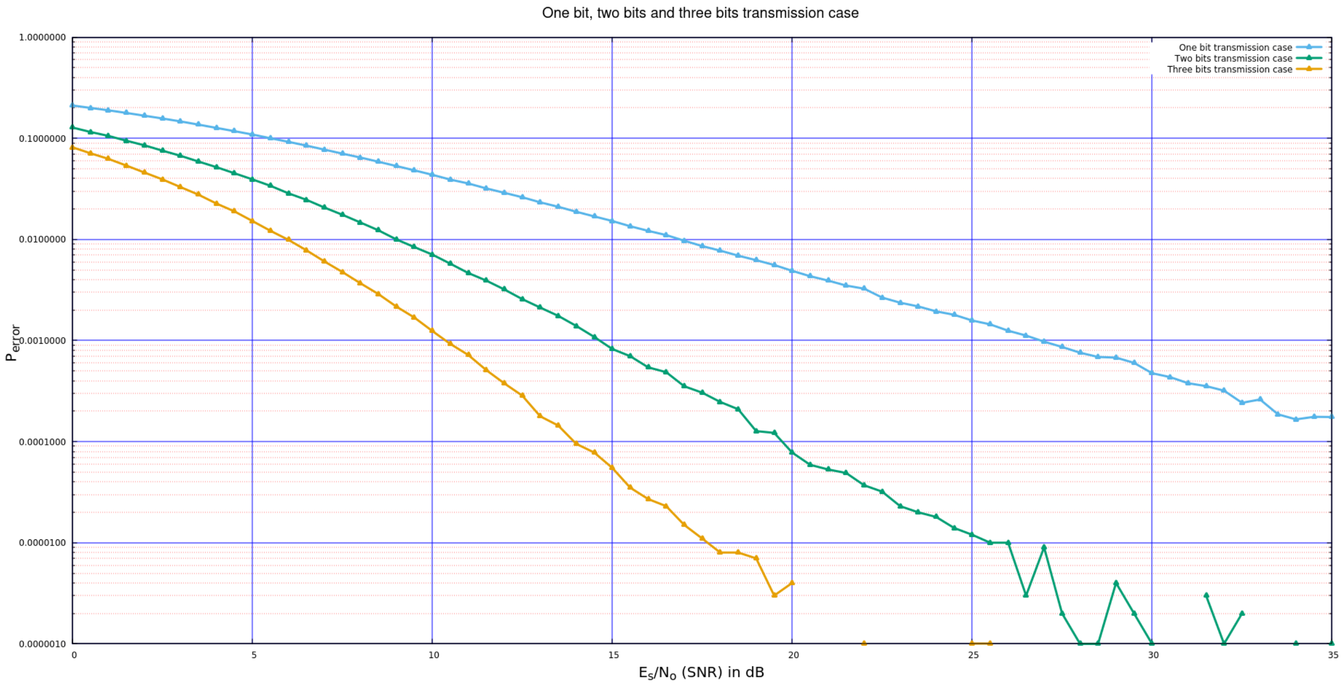

Figure 7: Only one bit, two same bits and three same bits transmission

6 Inference

Inferences list:

Based on the Comparison of AWGN and fadding

In fadding channel, probability of error approximately follows linear tend in downward direction (in log scale), but in AWGN channel falling tend is approximately exponential. That mean, as we increase SNR, AWGN channel improves a lot but fadding channel improve slowly.

Based on the approximation

In fadding channel case, probability of error for BPSK is a very complex term (relative to BPSK in AWGN), but it can be approximated by Taylor series, result as pretty well figure 2, and figure 3. For large SNR, actual data and approximated probability of error are indistinguishable. Hence, one can use approximated result for BPSK in fadding channel.

Based on the equal gain combining

As we can see in figure 6 and figure 7, by transmitting same bit multiple times, one can reduce the probability of error drastically for a given SNR in fadding channel.

7 Result/Conclusion

7.1 What did I learn?

1. I derived the fadding channel model.
2. I derived the BPSK scheme in fadding channel model (ML rule and Probability of error). I tested that results through c++ program.
3. I tested with same bit multiple times to improve performance.
4. I learned about overall performance of fadding channel through BPSK scheme.