WIRELESS COMMUNICATION PRACTISE

EXPERIMENT - 06 (Alamouti code)

Manas Kumar Mishra (ESD18I011)

08-MARCH-2022

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

1 Aim:

To analyze space time code proposed by Alamouti. With two transmitted antennas and one receiver antenna perform this experiment.

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 What is CSIT?

CSIT (Channel State Information), implies channel information is known at transmitter. Based on that information transmitter transmits the signals through channel. Now, question is, how transmitter can have information of channel (fadding coefficients) before transmitting the signal? There has to be some kind of feedback mechanism between receiver and transmitter and assumption of slow fadding channel. Transmitter estimate the fadding coefficient and send back to transmitter. But it require more processing.

3.2 Space time codes

For targeting the CSIT issue, space time codes are one option. One space time code was proposed by Alamouti [1998] for targeting CSIT issue. In that, assume two transmitter antennas transmitting at same time. Transmitters transmit S_1 and S_2 at one time instant followed by $-S_2^*$ and S_1^* in next time instant refer figure1.
Here assumptions are

1. Channel is not changing to two consecutive symbol duration.
2. Channel is frequency flat (Non-selective channel).

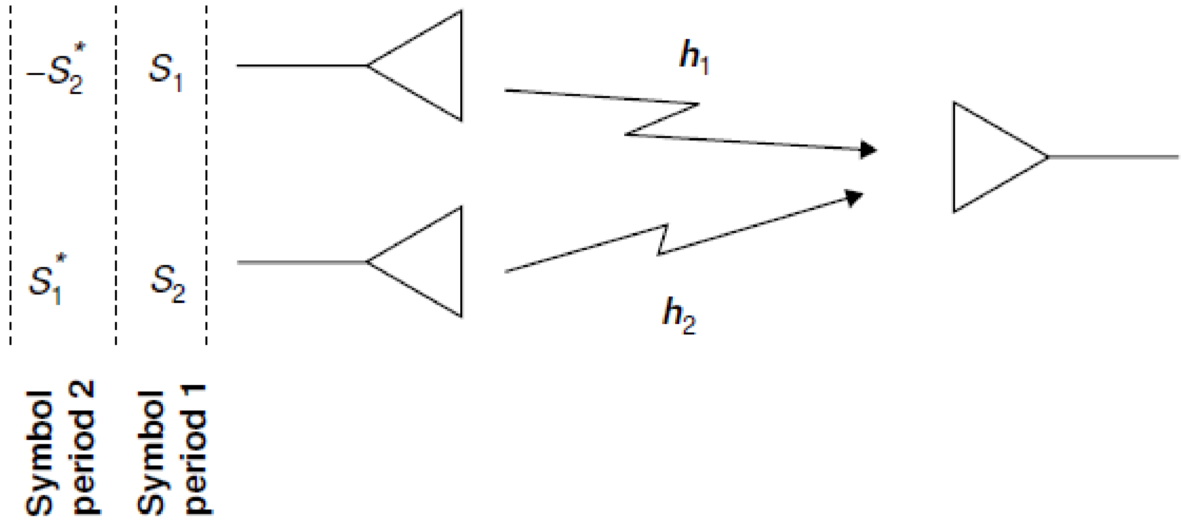

Figure 1: Alamouti scheme

3.3 How does it work?

Let y_1 and y_2 are received signal at two consecutive time instants.

$$y_1 = h_1\, S_1 + h_2\, S_2 + \eta_1$$
$$y_2 = h_2\, S_1^* - h_1\, S_2^* + \eta_2$$

Take conjugate on y_2 and rearrange the y_1 and y_2.

$$y_1 = h_1\, S_1 + h_2\, S_2 + \eta_1$$
$$y_2^* = h_2^*\, S_1 - h_1^*\, S_2 + \eta_2^*$$

In matrix form.

$$\begin{bmatrix} y_1 \\ y_2^* \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2^* \end{bmatrix}$$

At receiver, let there is a matrix $\bar{A}$ such that

$$A = \begin{bmatrix} h_1^* & h_2 \\ h_2^* & -h_1 \end{bmatrix}$$

Multiply A matrix with both side.

$$A \begin{bmatrix} y_1 \\ y_2^* \end{bmatrix} = A \begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} + A \begin{bmatrix} \eta_1 \\ \eta_2^* \end{bmatrix}$$
$$\begin{bmatrix} \tilde{y_1} \\ \tilde{y_2^*} \end{bmatrix} = \begin{bmatrix} (|h_1|^2 + |h_2|^2) S_1 \\ (|h_1|^2 + |h_2|^2) S_2 \end{bmatrix} + \begin{bmatrix} \tilde{\eta_1} \\ \tilde{\eta_2^*} \end{bmatrix}$$

where,

$$\tilde{y_1} = h_1^* \, y_1 + h_2 \, y_2$$
$$\tilde{y_2} = h_2^* \, y_1 - h_1 \, y_2$$

And $\tilde{\eta_1}$ and $\tilde{\eta_2^*}$ are gaussian noise with mean 0 and variance $(h_1^2 + h_2^2)\sigma^2$.
By ML decision rule assuming BPSK case, decision rule is same as normal BPSK case.

3.3.1 Probability of error

For, one antenna in fadding channel has

$$P_e \approx \frac{K}{SNR} \qquad \text{(For large value of SNR and K is a real constant)}$$

Similarly, one can get a intuitive expression two transmitting antenna when both transmitting antennas are transmitting independently

$$Pe_2 \approx \frac{K}{(SNR)^2} \qquad \text{(For large value of SNR)}$$

In my case, experimentally I found that K = 0.8 (not k=3) is satisfying the results. There is no proof for that, it just an approximation.

4 Pseudo code

S1. Generate random bit stream of length equal to million.
S2. Map 0 to $-\sqrt{E}$ and 1 to $\sqrt{E}$ and store in a dynamic vector (Modulation).
S3. Generate two gaussian noise vectors of zero mean and variance as half, treat as real part and imaginary part of the complex random variable (Size of this vector should be half of source vector).
S4. Compute Rayleigh coefficient by using above random variables.(To generate h_1)
S5. Repeat s3 and s4, to generate the fadding coefficients for second antenna (To generate h_2).
S6. Generate gaussian noise components.
S7. Generate transmitted signal (Tx_1 and Tx_2).
S8. Add gaussian noise.
S9. For receiver side simulations take linear combinations of the Tx_1 and Tx_2 to simulate the matrix multiplication with A.(Generate $\tilde{y_1}$ and $\tilde{y_2}$)
S10. Apply ML rule on $\tilde{y_1}$ and $\tilde{y_2}$ and decode the received signal.
S11. Count errors
S12. Repeat S1 to S11 for many SNR values.
S13. Store data (SNR, Error count) in .dat file.
S14. Compute theoretical probability of error. Store into .dat file
S15. Plot the result using gnuplot in semilog scale.

5 Code and result

To support matrix operation a new header file has been designed and added to the code. Similarly BPSK channel a header file has been used.

5.1 Alamouti code (Space time codes)

```
//////////////////////////////////////////////////////////////
///////////////////////////////////
// Author:- MANAS KUMAR MISHRA
// Organisation:- IIITDM KANCHEEPURAM
// Topic:- Alamouti code (Space time codes)
//////////////////////////////////////////////////////////////
///////////////////////////////////

#include <iostream>
#include <cmath>
#include <iterator>
#include <random>
#include <chrono>
#include <time.h>
#include <fstream>
#include "BPSK.h"
#include "MyMatrixOperation.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 fadding 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 multiplying fading coff to particular antenna
// Inputs are the Transmitted energy and Rayleigh fadding coff
// Output is the multiplication of element by element fadding and energy
vector<double> RayleighOperation(vector<double> TxEnergy,
                                 vector<double> RayleighFaddingCoff)
{
    vector<double> Resultant;

    for(int j =0; j<TxEnergy.size(); j++){
        Resultant.insert(Resultant.end(), TxEnergy[j]*RayleighFaddingCoff[j]);
    }

    return Resultant;
}


// Function for gaussian noise for each tx bit to particular antenna
// Inputs are the Transmitted energy and Gnoise
// Output is the addition of element by element Gnoise and Tx
vector<double> GaussianNoiseAdd(vector<double> TxEnergy,
                                 vector<double> Gnoise)
{
    vector<double> Resultant;

    for(int j =0; j<TxEnergy.size(); j++){
        Resultant.insert(Resultant.end(), TxEnergy[j]+Gnoise[j]);
    }

    return Resultant;
}


// 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, char strName[])
{
    ofstream outfile;

    string filename = strName;

    outfile.open(filename +"."+"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> CalculatedError(vector<double> SNR_dB, double L)
{
    vector<double> ProbError;

    double po, normalValue, inter;
    for (int k =0; k<SNR_dB.size(); k++){
        normalValue = pow(10, (SNR_dB[k]/10));
        po = 0.8/pow(normalValue,2);
        ProbError.insert(ProbError.end(), po);
    }

    return ProbError;
}


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

    // Mapping of bits to symbols;
```

```
    vector<double> transmittedSymbol;

    // Noise definition
    vector<double> gnoise;

    //Rayleigh noise (Real guass and Img Guass)
    vector<double> realGaussian1;
    vector<double> imziGaussian1;
    vector<double> RayleighCoff1;
    vector<double> realGaussian2;
    vector<double> imziGaussian2;
    vector<double> RayleighCoff2;

    vector<double> Tx1, Tx2;
    vector<double> Rx1, Rx2;

    //Alamouti decoder
    vector<double> Dec1, Dec2;

    vector<double> decodedBits1;
    vector<double> decodedBits2;

    vector<double> mergeDecoded;

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

    double sigmaSquare = 0.5;
    double stddevRayleigh = sqrt(sigmaSquare);

    vector<double> SNR_dB;
    for(float i =0; i<=25; 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();    //Source bit streame

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

```
        realGaussian1 = GnoiseVector(0.0, stddevRayleigh, one_million/2);
        imziGaussian1 = GnoiseVector(0.0, stddevRayleigh, one_million/2);

        RayleighCoff1 = RayleighFaddingCoff(realGaussian1, imziGaussian1);

        realGaussian2 = GnoiseVector(0.0, stddevRayleigh, one_million/2);
        imziGaussian2 = GnoiseVector(0.0, stddevRayleigh, one_million/2);

        RayleighCoff2 = RayleighFaddingCoff(realGaussian2, imziGaussian2);


        /*
        Tx1 = h1*s1+h2*s2
        Tx2 = h2*s1-h1*s2
        */
        for(int i =0; i<transmittedSymbol.size(); i=i+2 ){
            Tx1.insert(Tx1.end(), RayleighCoff1[i/2]*transmittedSymbol[i] +
            RayleighCoff2[i/2]*transmittedSymbol[i+1]);
            Tx2.insert(Tx2.end(),RayleighCoff2[i/2]*transmittedSymbol[i] -
            RayleighCoff1[i/2]*transmittedSymbol[i+1]);
        }

        gnoise = GnoiseVector(0.0, stdnoise, one_million);

        /*
        Rx1 = Tx1+gnoise;
        Rx2 = Tx2+gnoise;
        */
        for(int j =0; j<gnoise.size(); j =j+2){
            Rx1.insert(Rx1.end(), Tx1[j/2]+gnoise[j]);
            Rx2.insert(Rx2.end(), Tx2[j/2]+gnoise[j+1]);
        }

        for(int k =0; k<Tx1.size(); k++){
            Dec1.insert(Dec1.end(), RayleighCoff1[k]*Rx1[k]+RayleighCoff2[k]*Rx2[k]);
            Dec2.insert(Dec2.end(), RayleighCoff2[k]*Rx1[k]-RayleighCoff1[k]*Rx2[k]);
        }


        decodedBits1= decisionBlock(Dec1);
        decodedBits2 = decisionBlock(Dec2);

        for(int j =0; j<decodedBits1.size(); j++){
            mergeDecoded.insert(mergeDecoded.end(), decodedBits1[j]);
            mergeDecoded.insert(mergeDecoded.end(), decodedBits2[j]);
        }

        counterror = errorCalculation(sourceBits, mergeDecoded);
        P_error = counterror/one_million;
        Prob_error.insert(Prob_error.end(), P_error);

        cout<<endl;
        cout<<"Error count          :"<<counterror<<endl;
        cout<<"Probability of error :"<<P_error<<endl;
        cout<<endl;

```

```
        Tx1.clear();
        Tx2.clear();
        Rx1.clear();
        Rx2.clear();
        Dec1.clear();
        Dec2.clear();
        mergeDecoded.clear();

    }

    char NameofFile1[30] = "Alc1";
    char NameofFile2[30] = "Alcerr1";
    int L =2;

    datafile(SNR_dB, Prob_error, NameofFile1);

    vector<double> Error = CalculatedError(SNR_dB, 2);

    datafile(SNR_dB, Error, NameofFile2);

    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;

}


// 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;
}
```

Matrix operation header file

```
#include <iostream>
#include <vector>
#include <iterator>

using namespace std;

// Function of error message in matrices multiplications.
void errorMSG(){
    cout<<endl;
    cout<<"! Matrices size are not proper for multiplication !!! :-("<<endl;
    cout<<endl;
}

// Function for printing the matrix on console.
void PrintMat(vector<vector<double> > & MAT)
{
    for(int j =0; j<MAT.size();j++){
        for(int k =0; k<MAT[j].size(); k++){
            cout<<MAT[j][k]<< "    ";
        }
        cout<<endl;
    }
}


// Function for taking transpose of the given matrix.
// Input is the matrix for some m*n dimension.
// Output is the matrix with n*m dimension having transpose of actual matrix
vector<vector <double> > Transpose_MAT(vector<vector <double> > MAT)
{
    vector<vector<double> > TransMAT;
    vector <double> interMAT;

    for(int i =0; i<MAT[0].size(); i++){
        for(int j =0; j<MAT.size(); j++){
            interMAT.insert(interMAT.end(), MAT[j][i]);
        }
        TransMAT.push_back(interMAT);
        interMAT.clear();
    }

    return TransMAT;
}

```

```
// Function to fix the issue of vector and matrixes
// Input is the vector signal
// Output is the Matrix that contain vector as it's first raw
vector<vector<double>> convertVectorToMatrix(vector<double> Vec)
{
    vector<vector<double>> Mat;
    Mat.insert(Mat.end(), Vec);
    return Mat;
}


// Function for Multiplying two matrixs element by elements
// Input are two matrix of same dimension
// Output is another matrix that contain each element
//  as multiplication of each element.
vector<vector<double>> ElementWiseMultiplication(vector<vector<double>> Mat1,
                                                 vector<vector<double>> Mat2)
{
    vector<vector<double>> MatResult;

    vector<double> multi;

    for(int i =0; i<Mat1.size();i++){
        for(int j=0; j<Mat1[0].size(); j++){
            multi.insert(multi.end(), Mat1[i][j]*Mat2[i][j]);
        }

        MatResult.insert(MatResult.end(), multi);
        multi.clear();
    }

    return MatResult;
}


// Function for Adding two matrixs element by elements
// Input are two matrix of same dimension
// Output is another matrix that contain each element
//  as Addition of both element.
vector<vector<double>> ElementWiseAddition(vector<vector<double>> Mat1,
                                           vector<vector<double>> Mat2)
{

    vector<vector<double>> MatResult;

    vector<double> Add;

    for(int i =0; i<Mat1.size();i++){
        for(int j=0; j<Mat1[0].size(); j++){
            Add.insert(Add.end(), Mat1[i][j]+Mat2[i][j]);
        }

        MatResult.insert(MatResult.end(), Add);
        Add.clear();
    }

```

```
    return MatResult;
}

// function for sum of two vectors
// Inputs are two vectors
// Output is the sum of two vectors.
vector<double> VectorSum(vector<double> Vec1, vector<double> Vec2)
{
    vector<double> SumResult;

    if(Vec1.size()==Vec2.size()){
        for(int k=0; k<Vec1.size(); k++){
            SumResult.insert(SumResult.end(), Vec1[k]+Vec2[k]);
        }

        return SumResult;
    }else{
        cout<<"Error !!!! Vector size should be same!!!"<<endl;
        return SumResult;
    }
}
```

5.2 Results

5.2.1 Images

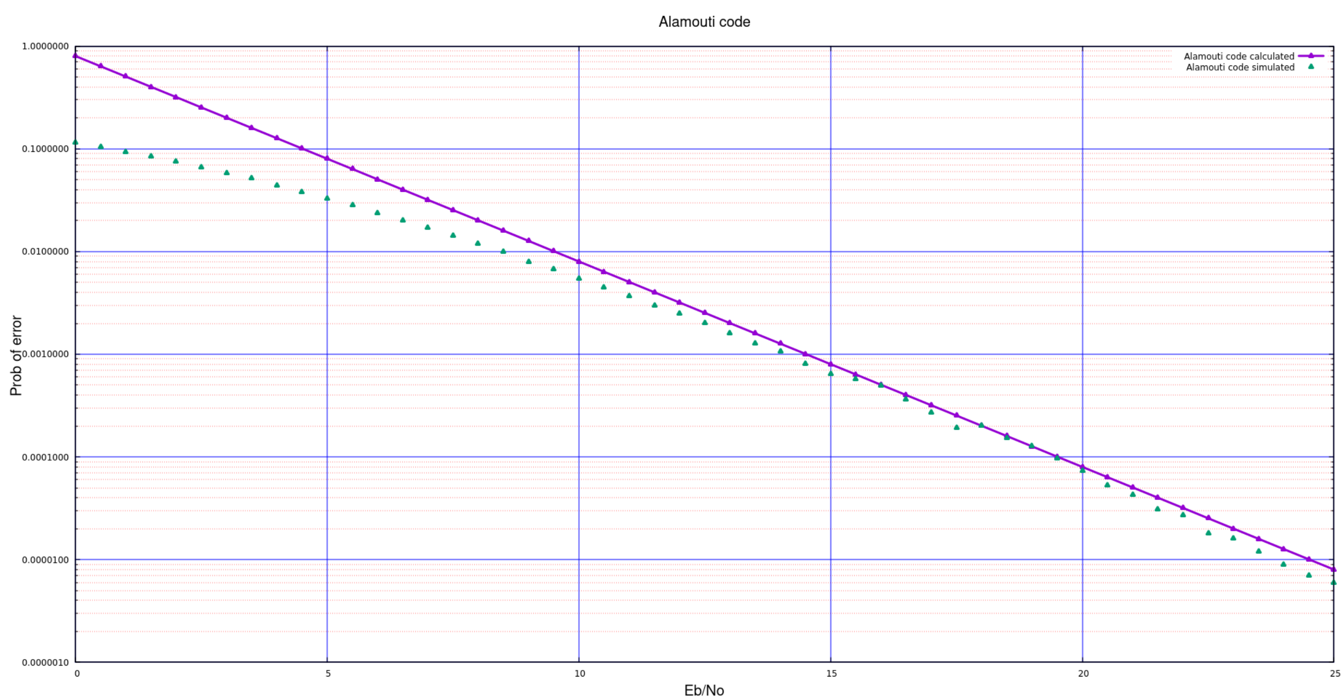

Figure 2: Alamouti code simulation

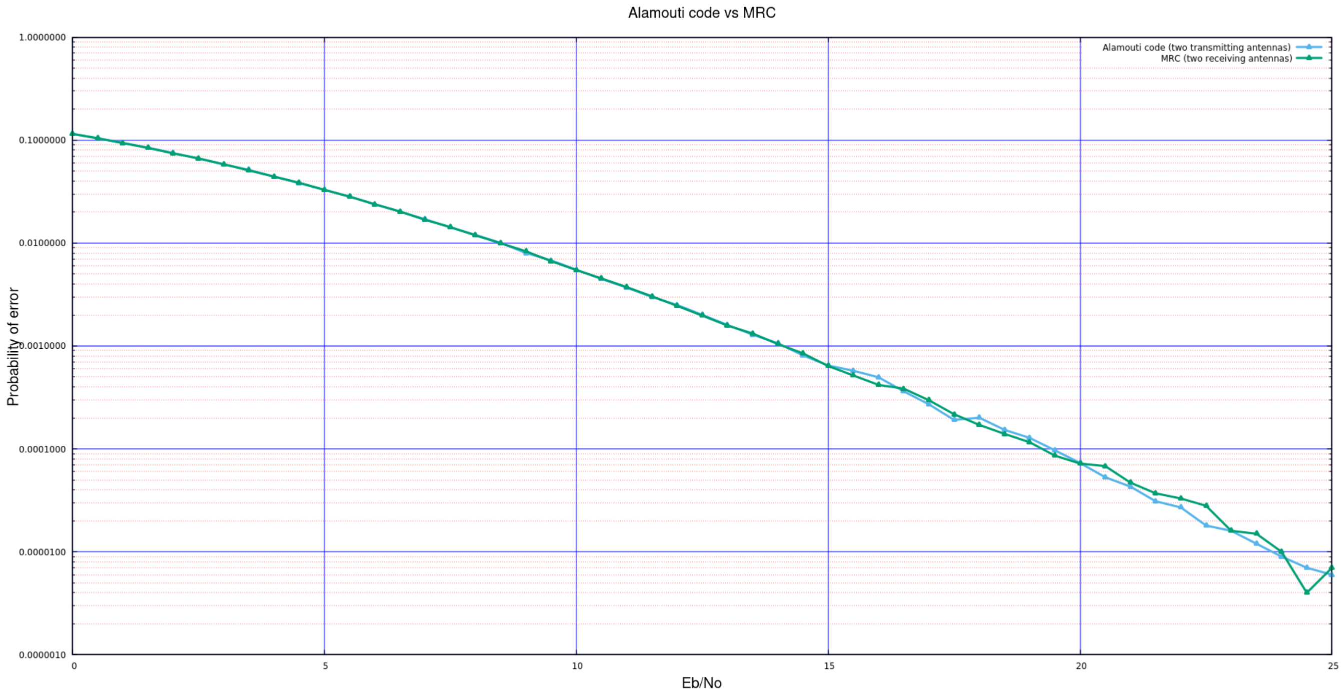

Figure 3: Alamouti code and MRC

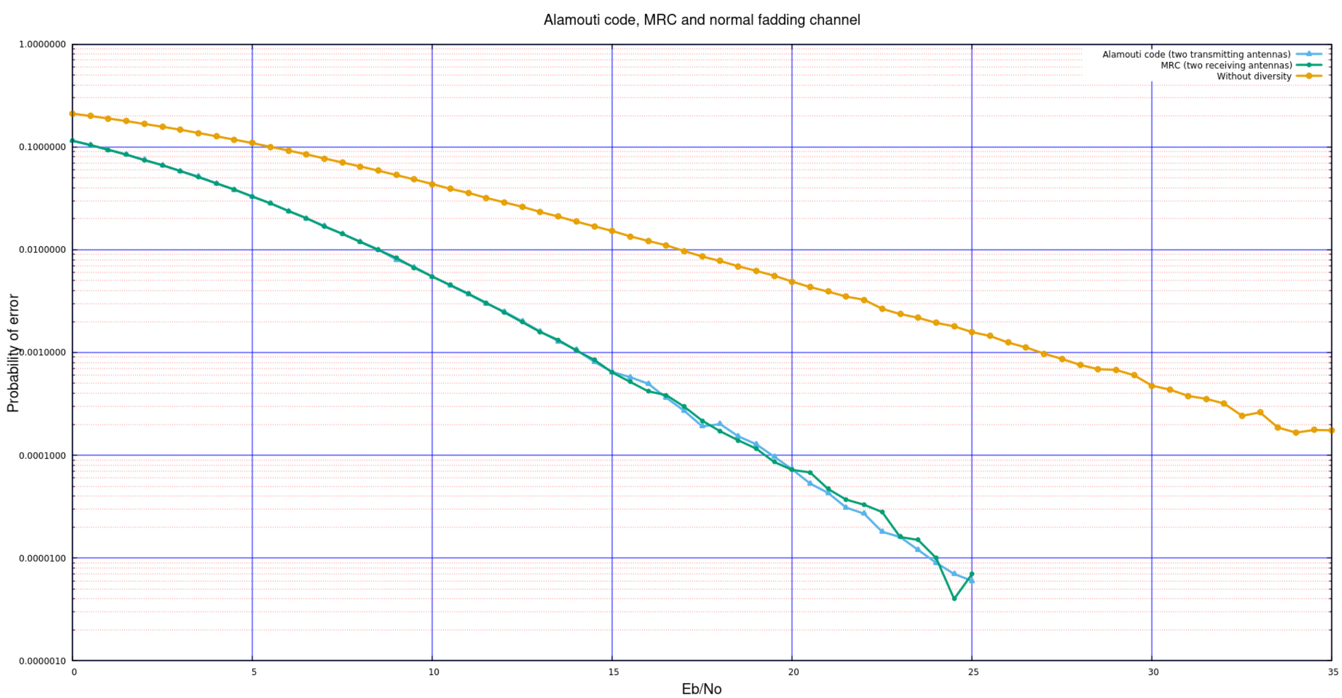

Figure 4: Alamouti code, MRC and Channel without fadding

6 Inferences

Inferences list:

Based on the Alamouti scheme and MRC

From figure3, we can see that Alamouti code gives same performance as MRC, without using multiple receiver antenna. It implies, if we use two transmit antenna and two receive antenna then we can improve a lot through Almouti coding scheme. That means, good performance without compromising data rate reduction.

Based on the diversity gain and antenna gain

From figure4, we can see that Alamouti code has same diversity gain and antenna gain as MRC. That is not surprising, because, in MRC, we were using two receiver antennas and doing something on both side signals (received signal from both antennas). Similarly, in Alamouti code, we are using two received signals that are separated in time.

7 Result/Conclusion

7.1 What did I learn?

1. I understood the Alamouti scheme.
2. I understood how can we improve the performance of fadding channel without CSIT and without receiver diversity.
3. Experimentally proof that Alamouti code has same performance as MRC .