# Dominik Ciesiołkiewicz 44289 Sprawozdanie Lab 10 - Właściwości toru transmisyjnego

W poniższym zadaniu zastosowałem szum biały, na którego algorytm podany był przez Pana mgr. inż. Wernika w opisie do poniższego laboratorium. Ważniejsze części kodu dotyczące tego zadania (a nie poprzedniego) pozwoliłem sobie zaznaczyć niebieską czcionką, dla ułatwienia sprawdzania. Wykresy znajdują się na końcu sprawozdania.

**Poziomy szumów:**

Małe (<0.30) dla alfa>0.3

Średnie (0.30-0.50) dla 0.1<alfa<0.3

Duże (>0.50) dla alfa<0.1

**Kod:**

#include <iostream>

#include <fstream>

#include <complex>

#include <bitset>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

double pi = 3.14159265359;

int lengthOfString(string str)

{

return str.length();

}

string S2BS(string in, bool choice) //String To Binary Stream

{

string out = "";

int n = in.length();

string bity = "";

if (choice == 1)//LittleEndian

{

for (int i = 0; i < n; i++)

{

int wartosc = int(in[i]);

bity = "";

while (wartosc > 0)

{

if (wartosc % 2)

{

bity += '1';

}

else

{

bity += '0';

}

wartosc = wartosc / 2;

}

out += bity;

}

reverse(out.begin(), out.end());

//cout << out << endl;

return out;

}

else {//BigEndian

for (int i = 0; i < n; i++)

{

int wartosc = int(in[i]);

bity = "";

while (wartosc > 0)

{

if (wartosc % 2)

{

bity += '1';

}

else

{

bity += '0';

}

wartosc = wartosc / 2;

}

reverse(bity.begin(), bity.end());

out += bity;

}

//cout << out << endl;

return out;

}

}

string BS2S(string charset)

{

stringstream strumien(charset);

string result;

while (strumien.good())

{

bitset<7> bity;

strumien >> bity;

char znak = char(bity.to\_ulong());

result += znak;

}

return result;

}

int\* Hamming(int\* d)

{

int G[7][4] = { {1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };

int\* K = new int[7];

for (int i = 0; i < 7; i++)

{

K[i] = 0;

for (int j = 0; j < 4; j++)

{

//cout << G[i][j];

//cout<<d[j]<<endl;

K[i] += G[i][j] \* (d[j]);

}

//cout << endl;

}

/\*cout << "K:" << endl;

for (int i = 0; i < 7; i++)

{

cout << K[i] << endl;

}

cout << endl;

cout << "K modulo 2:" << endl;\*/

for (int i = 0; i < 7; i++)

{

K[i] = K[i] % 2;

//cout << K[i] << endl;

}

//cout << endl;

return K;

}

int\* HammingM(int\* d)

{

int\* KD = new int[7];

KD[2] = d[0];

KD[4] = d[1];

KD[5] = d[2];

KD[6] = d[3];

if (d[0] != 0 && d[0] != 1)

KD[2] = 0;

if (d[1] != 0 && d[1] != 1)

KD[4] = 0;

if (d[2] != 0 && d[2] != 1)

KD[5] = 0;

if (d[3] != 0 && d[3] != 1)

KD[6] = 0;

KD[0] = KD[2] ^ KD[4] ^ KD[6];

KD[1] = KD[2] ^ KD[5] ^ KD[6];

KD[3] = KD[4] ^ KD[5] ^ KD[6];

return KD;

}

int\* HammingSECDEC(string d)

{

int G[7][4] = { {1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };

int\* K = new int[8];

for (int i = 0; i < 7; i++)

{

K[i] = 0;

for (int j = 0; j < 4; j++)

{

K[i] += G[i][j] \* (d[j] - '0');

}

}

/\*cout << "K:" << endl;

for (int i = 0; i < 7; i++)

{

cout << K[i] << endl;

}

cout << endl;

cout << "K modulo 2:" << endl;\*/

for (int i = 0; i < 7; i++)

{

K[i] = K[i] % 2;

//cout << K[i] << endl;

}

//cout << endl;

//Dla SECDEC:

//cout << "Ze sprawdzajacym bitem: " << endl;

int err = 0;

for (int i = 0; i < 7; i++)

{

err += K[i];

}

err = err % 2;

K[7] = err;

/\*for (int i = 0; i < 8; i++)

{

cout << K[i] << endl;

}

cout << endl;\*/

return K;

}

int\* DecHamming(int\* K)

{

int H[3][7] = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };

int\* KD = new int[7];

for (int i = 0; i < 3; i++)

{

KD[i] = 0;

for (int j = 0; j < 7; j++)

{

KD[i] += H[i][j] \* K[j];

}

}

for (int i = 0; i < 3; i++)

{

KD[i] = KD[i] % 2;

}

return KD;

}

int\* DecHammingM(int\* K)

{

int\* KD = new int[4];

KD[0] = K[2];

KD[1] = K[4];

KD[2] = K[5];

KD[3] = K[6];

return KD;

}

int\* DecHammingSECDEC(int\* K)

{

int H[3][7] = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };

int\* KD = new int[7];

cout << "Sprawdzanie p4:" << endl;

int err = 0;

for (int i = 0; i < 7; i++)

{

err += K[i];

}

err = err % 2;

if (err != K[7])

{

cout << "P4 nie jest zgodne. Mamy 50% szans na powodzenie naprawy." << endl << endl;

}

else

{

cout << "P4 jest zgodne" << endl << endl;

}

int p1 = (K[0] + K[2] + K[4] + K[6]) % 2;

int p2 = (K[1] + K[2] + K[5] + K[6]) % 2;

int p3 = (K[3] + K[4] + K[5] + K[6]) % 2;

int n = p1 \* 1 + p2 \* 2 + p3 \* 4 - 1;

cout << "Poprawiony kod odebrany:" << endl;

if (K[n] == 0)

{

K[n] = 1;

}

else

{

K[n] = 0;

}

for (int i = 0; i < 8; i++)

{

cout << K[i] << endl;

}

cout << endl << "Sprawdzanie p4 - ponowne:" << endl;

n = 0;

for (int i = 0; i < 7; i++)

{

n += K[i];

}

n = n % 2;

if (n != K[7])

{

cout << "P4 nie jest zgodne. Sa co najmniej 2 bledne bity. Odrzucamy pakiet." << endl << endl;

return NULL;

}

else

{

cout << "P4 jest zgodne, odkodowujemy:" << endl << endl;

cout << "Informacja odkodowana:" << endl;

cout << K[2] << endl;

cout << K[4] << endl;

cout << K[5] << endl;

cout << K[6] << endl;

}

return K;

}

int\* BitNegation(int\* K, int NoBit)

{

if (K[NoBit] == 0)

K[NoBit] = 1;

else

K[NoBit] = 0;

return K;

}

double ASKF(int m, double t)

{

double A1 = 0;

double A2 = 2.0;

double f = 2.0;

double phi = 0.0;

if (m == 0)

return (A1 \* sin(2 \* pi \* t \* f + phi));

else

return (A2 \* sin(2 \* pi \* t \* f + phi));

}

double FSKF(int m, double t)

{

double A = 1;

double f0 = 1.0;

double f1 = 2.0;

double phi = 1.0;

if (m == 0)

return (A \* sin(2 \* pi \* t \* f0 + phi));

else

return (A \* sin(2 \* pi \* t \* f1 + phi));

}

double PSKF(int m, double t)

{

double A = 1;

double f = 2.0;

double phi0 = 0.0;

double phi1 = pi;

if (m == 0)

return A \* sin(2 \* pi \* t \* f + phi0);

else

return A \* sin(2 \* pi \* t \* f + phi1);

}

double ASKDx(double v, double SinE)

{

return v \* SinE;

}

double ASKDp(double vX, double del)

{

return del \* vX;

}

double ASKDm(double vP, double h)

{

if (vP > h)

return 1;

else

return 0;

}

double\* noise(double\* sig, double\* ret, int len)

{

const static int q = 15;

const static float c1 = (1 << q) - 1;

const static float c2 = ((int)(c1 / 3)) + 1;

const static float c3 = 1.f / c1;

float random = 0.f;

float noisef = 0.f;

float alfa = 0.25;

int nn = 0, NN = len \* 40;

for (int i = 0; i < len \* 40; i++)

{

random = ((float)rand() / (float)(RAND\_MAX + 1));

noisef = (2.f \* ((random \* c2) + (random \* c2) + (random \* c2)) - 3.f \* (c2 - 1.f)) \* c3;

ret[i] = (sig[i] \* alfa) + (noisef \* (1.0 - alfa));

//cout << sig[i] << endl;

nn++;

if (nn >= NN)

{

nn = 0;

}

}

return ret;

}

int main()

{

double Tb = 0.1; //[s]

int fs = 10000; //[Hz]

//WCZYTYWANIE INFORMACJI

cout << "Zdanie zakodowane: ALAMAKOTA" << endl;

string str = S2BS("ALAMAKOTA", 0);

int n = lengthOfString(str);

cout << "Ilosc bitow transmisji: " << n << endl << endl;

int\* tab = new int[n];

for (int i = 0; i < n; i++)

{

if (str[i] == 48)

tab[i] = 0;

else

tab[i] = 1;

}

ofstream saveData("Dane.txt");

cout << "Informacja:" << endl;

for (int i = 0; i < n; i++)

{

cout << tab[i];

}

cout << endl << endl;

saveData << str << endl;

saveData.close();

vector<int> vectorASKD;

vector<int> vectorFSKD;

vector<int> vectorPSKD;

vector<int> dASK;

vector<int> dFSK;

vector<int> dPSK;

vector<int> dASKAMP;

vector<int> dFSKAMP;

vector<int> dPSKAMP;

bool SECDEC = 0;// 0-zwykly kod Hamminga; 1-SECDEC

//KODOWANIE KODEM HAMMINGA

int\* HammingF;

int\* HammingZ;

vector<int> vector;

for (int i = 0; i < n; i += 4)

{

HammingF = HammingM(&tab[i]);

for (int i = 0; i < 7; i++)

vector.push\_back(HammingF[i]);

}

cout << "Dane zakodowane:" << endl;

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

{

cout << vector[i];

}

cout << endl << endl;

//MODULACJA

ofstream ASKf("ASK.txt");

ofstream PSKf("PSK.txt");

ofstream FSKf("FSK.txt");

ofstream time("time.txt");

double diff = 0.025;//bo 1/40

double\* ASK = new double[vector.size() \* 40];

double\* PSK = new double[vector.size() \* 40];

double\* FSK = new double[vector.size() \* 40];

int\* Sinus1 = new int[vector.size() \* 40];

int\* Sinus2 = new int[vector.size() \* 40];

int\* Sinus3 = new int[vector.size() \* 40];

int\* Sinus4 = new int[vector.size() \* 40];

for (int i = 0; i < vector.size() \* 40; i++)

{

ASK[i] = ASKF(vector[int(i \* diff)], i \* diff);

PSK[i] = PSKF(vector[int(i \* diff)], i \* diff);

FSK[i] = FSKF(vector[int(i \* diff)], i \* diff);

ASKf << ASKF(vector[int(i \* diff)], i \* diff) << endl;

PSKf << PSKF(vector[int(i \* diff)], i \* diff) << endl;

FSKf << FSKF(vector[int(i \* diff)], i \* diff) << endl;

//cout << ASK[i] << endl;

time << i << endl;

Sinus1[i] = ASKF(1, i \* diff);

Sinus2[i] = PSKF(1, i \* diff);

Sinus3[i] = FSKF(0, i \* diff);

Sinus4[i] = FSKF(1, i \* diff);

}

//SZUM

ofstream ASKszum("ASK\_Noise.txt");

ofstream PSKszum("PSK\_Noise.txt");

ofstream FSKszum("FSK\_Noise.txt");

double\* ASKSz = new double[vector.size() \* 40];

double\* PSKSz = new double[vector.size() \* 40];

double\* FSKSz = new double[vector.size() \* 40];

ASK = noise(ASK, ASKSz, vector.size());

PSK = noise(PSK, PSKSz, vector.size());

FSK = noise(FSK, FSKSz, vector.size());

for (int i = 0; i < vector.size() \* 40; i++)

{

ASKszum << ASK[i] << endl;

PSKszum << PSK[i] << endl;

FSKszum << FSK[i] << endl;

}

ASKszum.close();

PSKszum.close();

FSKszum.close();

//DEMODULACJA

double tempASK = 0;

double tempFSK = 0;

double tempPSK1 = 0;

double tempPSK2 = 0;

double tempPSK3 = 0;

for (int i = 0; i < vector.size() \* 40; i++)

{

if (i >= 1)

{

tempASK = ASKDp(ASKDx(ASK[i], Sinus1[i]), diff) + ASKDp(ASKDx(ASK[i - 1], Sinus1[i - 1]), diff);

tempFSK = ASKDp(ASKDx(FSK[i], Sinus2[i]), diff) + ASKDp(ASKDx(FSK[i - 1], Sinus2[i - 1]), diff);

tempPSK1 = ASKDp(ASKDx(PSK[i], Sinus3[i]), diff) + ASKDp(ASKDx(PSK[i - 1], Sinus3[i - 1]), diff);

tempPSK2 = ASKDp(ASKDx(PSK[i], Sinus4[i]), diff) + ASKDp(ASKDx(PSK[i - 1], Sinus4[i - 1]), diff);

tempPSK3 = tempPSK2 - tempPSK1;

}

else

{

tempASK = ASKDp(ASKDx(ASK[i], Sinus1[i]), diff);

tempFSK = ASKDp(ASKDx(FSK[i], Sinus2[i]), diff);

tempPSK3 = (ASKDp(ASKDx(PSK[i], Sinus4[i]), diff) - ASKDp(ASKDx(PSK[i], Sinus3[i]), diff));

}

dASK.push\_back(ASKDm(tempASK, 0));

dFSK.push\_back(ASKDm(tempFSK, 0));

dPSK.push\_back(ASKDm(tempASK, 0));

}

for (int i = 5; i < dASK.size(); i += 40)

{

dASKAMP.push\_back(dASK[i]);

dFSKAMP.push\_back(dFSK[i]);

dPSKAMP.push\_back(dPSK[i]);

}

cout << "Demodulacja ASK:" << endl;

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

{

cout << dASKAMP[i];

}

cout << endl << endl;

cout << "Demodulacja FSK:" << endl;

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

{

cout << dFSKAMP[i];

}

cout << endl << endl;

cout << "Demodulacja PSK:" << endl;

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

{

cout << dPSKAMP[i];

}

cout << endl;

//DEKODOWANIE

//DEKODOWANIE ASK

for (int i = 0; i < dASKAMP.size(); i += 7)

{

HammingZ = DecHammingM(&dASKAMP[i]);

for (int i = 0; i < 4; i++)

vectorASKD.push\_back(HammingZ[i]);

}

cout << endl;

cout << "Dane zdekodowane ASK:" << endl;

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

{

cout << vectorASKD[i];

}

cout << endl;

//DEKODOWANIE FSK

for (int i = 0; i < dFSKAMP.size(); i += 7)

{

HammingZ = DecHammingM(&dFSKAMP[i]);

for (int i = 0; i < 4; i++)

vectorFSKD.push\_back(HammingZ[i]);

}

cout << endl;

cout << "Dane zdekodowane FSK:" << endl;

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

{

cout << vectorFSKD[i];

}

cout << endl << endl;

//DEKODOWANIE PSK

for (int i = 0; i < dPSKAMP.size(); i += 7)

{

HammingZ = DecHammingM(&dPSKAMP[i]);

for (int i = 0; i < 4; i++)

vectorPSKD.push\_back(HammingZ[i]);

}

cout << "Dane zdekodowane PSK:" << endl;

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

{

cout << vectorPSKD[i];

}

cout << endl << endl;

//LICZENIE WSKAŹNIKA BER

double BER = 0;

cout << "Wskaznik BER dla ASK: ";

for (int i = 0; i < n; i++)

{

if (tab[i] != vectorASKD[i])

BER++;

}

BER = BER / n;

cout << BER << endl << endl;

//BER = 0;

//cout << "Wskaznik BER dla PSK: ";

//for (int i = 0; i < n; i++)

//{

// if (tab[i] != vectorPSKD[i])

// BER++;

//}

//BER = BER / n;

//cout << BER << endl << endl;

//DANE BINARNE NA ZDANIE:

//string decASK;

//string decFSK;

//string decPSK;

//for (int i = 0; i < vectorASKD.size(); i++)

//{

// decASK += tab[i] + '0';

// decFSK += tab[i] + '0';

// decPSK += tab[i] + '0';

//}

//cout << "Zdanie odtworzone z ASK: " << BS2S(decASK) << endl;

//cout << "Zdanie odtworzone z FSK: " << BS2S(decFSK) << endl;

//cout << "Zdanie odtworzone z PSK: " << BS2S(decPSK) << endl;

//Poziomy szumów:

//Małe (<0.30) dla alfa>0.3

//Średnie(0.30-0.50) dla 0.1<alfa<0.3

//Duże(>0.50) dla alfa<0.1

ASKf.close();

PSKf.close();

FSKf.close();

time.close();

return 1;

}

**Wykresy:**











