Dominik Ciesiołkiewicz 44289 222B – Sprawozdanie Lab 7

Kod:

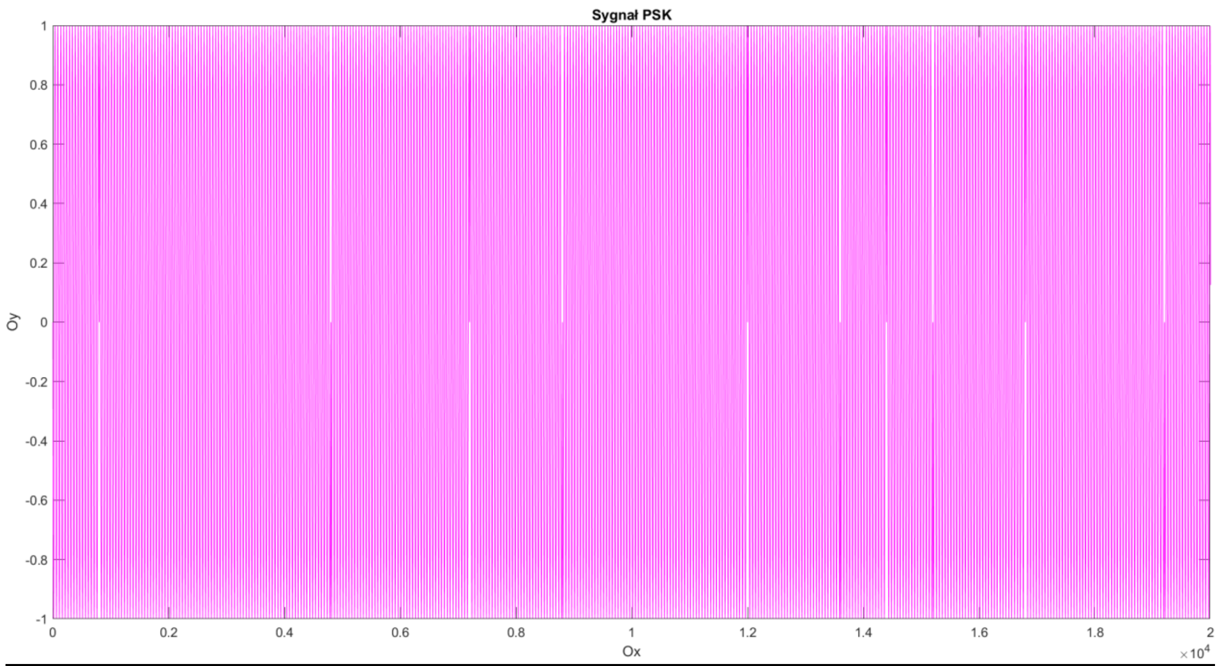
```cpp
#include <iostream>
#include <fstream>
#include <complex>

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;
    }
}

int* Mgenerator(string tab, int size, double Tb, double fs)
{
    ofstream saveM("M.txt");
    int probki = fs * Tb;
    int* m = new int[size * probki * 8];
    int index = 0;
    /*for (int i = 0; i < size/8; i++)
    {
        for (int j = 7; j >= 0; j--)
        {
            for (int k = 0; k < probki; k++)
            {
                if (tab[i]=='1' & (1 << j))
                {
                    m[index] = 1;
                }
                else
                {
                    m[index] = 0;
                }
                saveM << m[index] << endl;
                index++;
            }
        }
    }
    */
    for (int i = 0; i < size; i++)
    {
        if (tab[i] == '1')
        {
            for (int j = 0; j < 8 * probki; j++)
            {
                m[index] = 1;
                saveM << m[index] << endl;
                index++;
            }
        }
        else
        {
            for (int j = 0; j < 8 * probki; j++)
            {
                m[index] = 0;
                saveM << m[index] << endl;
                index++;
            }
        }
    }

    saveM.close();
    return m;
}

complex<double>* DFT(const double* tab, int N)
{
    complex<double>* tab2 = new complex<double>[N];

    for (int k = 0; k < N; k++)
    {
        tab2[k] = 0;
        complex<double> WN = cos(tab[k]) + 1i * sin(tab[k]);

        for (int n = 0; n < N; n++)
        {
            tab2[k] += tab[n] * pow(WN, -k * n);
        }

        //for (int n = 0; n < N; n++)
        //{
        //    tab2[k] += tab[n] * exp(-2 * pi * 1i * (double)k * (double)n / 
(double)N);
        //}

    }

    return tab2;
}

double ton_prosty(double A1, double F, double t)// czy jest w ogóle potrzebny?
{
    return A1 * sin(2 * pi * F * t);
}

double* ASK(int* m, int n, int A1, int A2, double f, double fs, double phi)
{
    double* zA = new double[n];
    for (int i = 0; i < n; i++)
    {
        if (m[i] == 0)
        {
            zA[i] = A1 * sin(2 * pi * f * i / fs + phi);
        }
        else
        {
            zA[i] = A2 * sin(2 * pi * f * i / fs + phi);
        }
    }
    ofstream saveASK("zad2ASK.txt");
    for (int i = 0; i < n; i++)
    {
        saveASK << zA[i] << endl;
    }
    saveASK.close();
    return zA;
}

double* FSK(int* m, int n, int A, int N, double fs, double Tb, double phi)
{
    double* zF = new double[n];
```

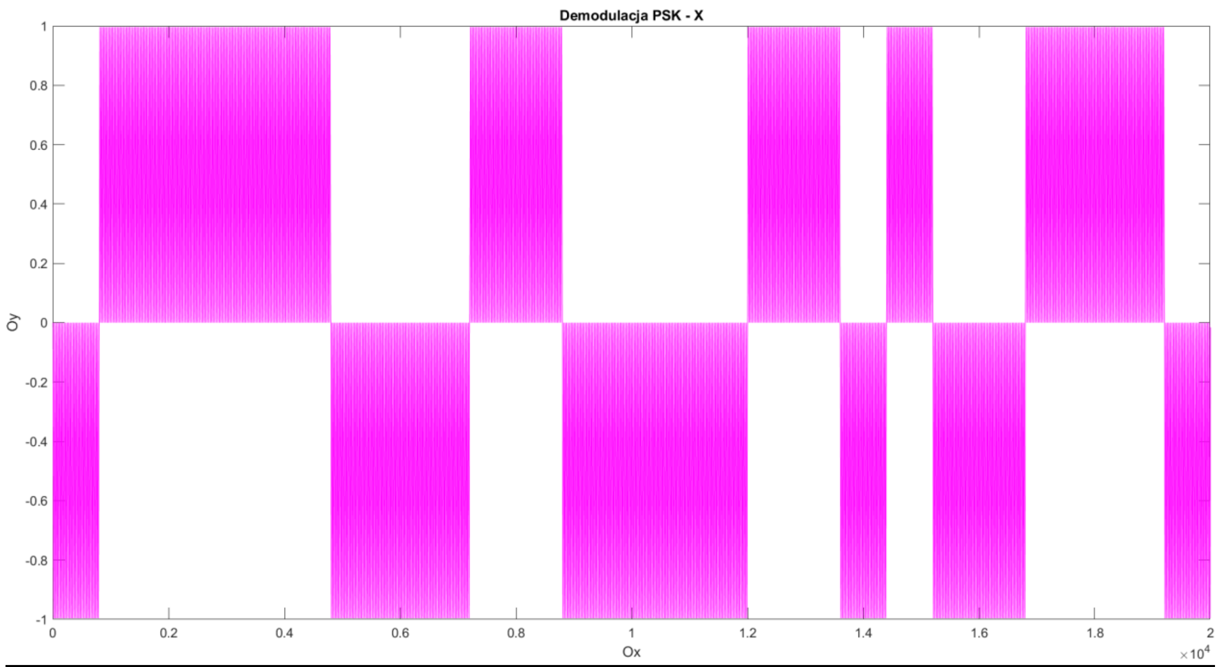
```cpp
    double f0 = (N + 1) / Tb;
    double f1 = (N + 2) / Tb;
    for (int i = 0; i < n; i++)
    {
        if (m[i] == 0)
        {
            zF[i] = A * sin(2 * pi * f0 * i / fs + phi);
        }
        else
        {
            zF[i] = A * sin(2 * pi * f1 * i / fs + phi);
        }
    }
    ofstream saveFSK("zad2FSK.txt");
    for (int i = 0; i < n; i++)
    {
        saveFSK << zF[i] << endl;
    }
    saveFSK.close();
    return zF;
}

double* PSK(int* m, int n, int A, double f, double fs, double Tb)
{
    double* zP = new double[n];
    for (int i = 0; i < n; i++)
    {
        if (m[i] == 0)
        {
            zP[i] = A * sin(2 * pi * f * i / fs + 0);
        }
        else
        {
            zP[i] = A * sin(2 * pi * f * i / fs + pi);
        }
    }
    ofstream savePSK("zad2PSK.txt");
    for (int i = 0; i < n; i++)
    {
        savePSK << zP[i] << endl;
    }
    savePSK.close();
    return zP;
}

void printOut(double* tab, int n, bool sw)
{
    if (sw == 0)
    {
        for (int i = 0; i < n; i++)
        {
            cout << tab[i] << endl;
        }
    }
    else
    {
        ofstream saveASK("zad1ASK.txt");
        for (int i = 0; i < n; i++)
        {
            saveASK << tab[i] << endl;
        }
        saveASK.close();
    }
}

void widmoAmplitudowe(complex<double>* DFTvalues, int size)
{
    double* M = new double[22050];
    double* Mprim = new double[22050];

    ofstream saveM("zad3M.txt");
    ofstream saveMprim("zad3Mprim.txt");

    for (int i = 0; i < size; i++)
    {
        M[i] = sqrt(pow(real(DFTvalues[i]), 2) + pow(imag(DFTvalues[i]), 2));
        saveM << M[i] << endl;
        Mprim[i] = 10 * log10(M[i]);
        saveMprim << Mprim[i] << endl;
    }

    saveM.close();
    saveMprim.close();
}

void szerokoscPasma(double* pasmo, int n) {
    double max = pasmo[0];
    double min = pasmo[0];

    for (int i = 1; i < n; i++)
    {
        if (pasmo[i] < min)
        {
            min = pasmo[i];
        }
        if (pasmo[i] > max)
        {
            max = pasmo[i];
        }
    }

    double szerokosc = max - min;
    cout << szerokosc << endl;
}

double* sinusoid(double f, double phi, double A, double fs, int probki)
{
    double* sinus = new double[probki];
    for (int i = 0; i < probki; i++) {
        sinus[i] = A * sin(2 * pi * i / fs * f + phi);
    }
    return sinus;
}

int* demodulatorASKPSK(double* pasmo, int n, double h, double fs, double f, double A)
{
    //Faza 1:

    ofstream saveDemASK_X("DemASK_X.txt");
    double* Sinus = sinusoid(f, 0, A, fs, n);
    double* x = new double[n];
    for (int i = 0; i < n; i++) {
        x[i] = pasmo[i] * Sinus[i];
        saveDemASK_X << x[i] << endl;
    }
    saveDemASK_X.close();

    //Faza 2 i 3:
    double * pt = new double[n];
    int* mt = new int[n];

    ofstream saveDemASK("DemASK.txt");
    ofstream saveDemASK_P("DemASK_P.txt");
    double calka;
    for (int i = 0; i < n; i++)
    {
        double suma = 0;

        if (i % 625 == 0)
           calka = 0;
        calka += x[i];
        saveDemASK_P << calka << endl;

        //cout << suma << endl;

        if (calka >= h)
        {
            mt[i] = 1;
        }
        else
        {
            mt[i] = 0;
        }

        saveDemASK << mt[i] << endl;
        //cout << mt[i] << endl;
    }
    saveDemASK.close();
    saveDemASK_P.close();

    return mt;
}

int* demodulatorFSK(double* pasmo, int n, double h, double fs, double f1, double f2,
double A)
{
    //Faza 1:
    double* x1 = new double[n];
    double* x2 = new double[n];
    double calka1;
    double calka2;
    double* Sinus1 = sinusoid(f1, 0, A, fs, n);
    double* Sinus2 = sinusoid(f2, 0, A, fs, n);

    ofstream saveDemFSK_X1("DemFSK_X1.txt");
    ofstream saveDemFSK_X2("DemFSK_X2.txt");
    for (int i = 0; i < n; i++) {
        x1[i] = pasmo[i] * Sinus1[i];
        saveDemFSK_X1 << x1[i] << endl;
        x2[i] = pasmo[i] * Sinus2[i];
        saveDemFSK_X2 << x2[i] << endl;
    }
    saveDemFSK_X1.close();
    saveDemFSK_X2.close();

    //Faza 2 i 3:
```

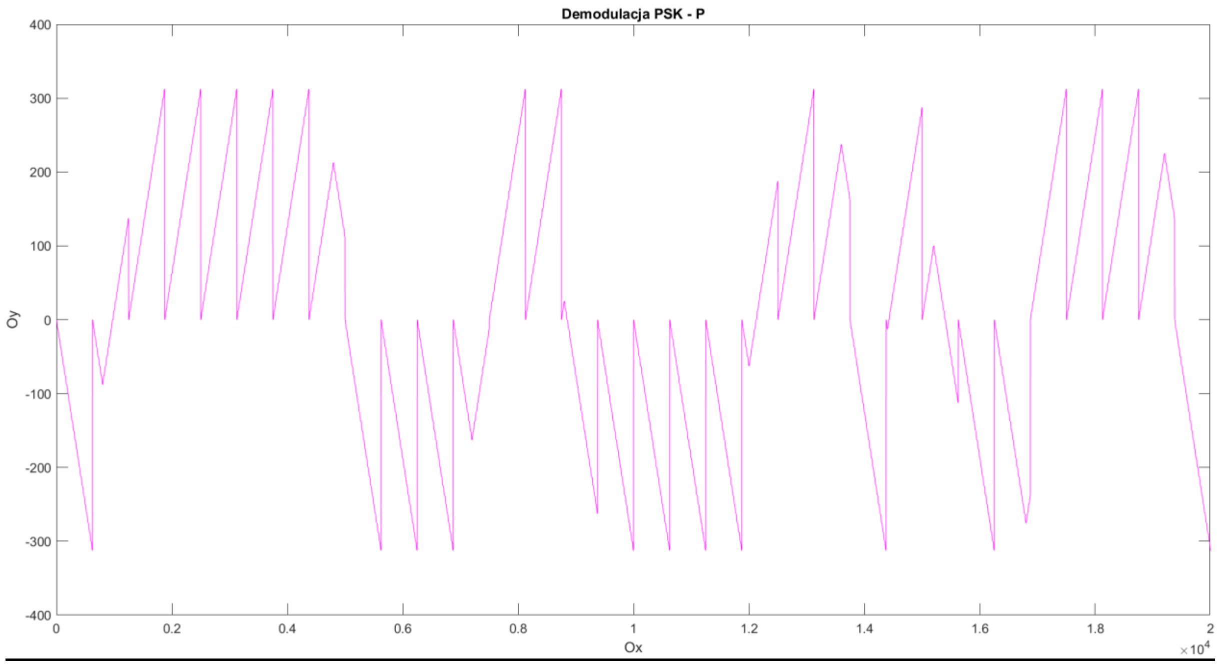
```
    ofstream saveDemFSK("DemFSK.txt");
    ofstream saveDemFSK_P("DemFSK_P.txt");
    double* pt = new double[n];
    int probkiNaBit = 2;
    int* mt = new int[n];

    double p;
    for (int i = 0; i < n; i++)
    {
        double suma = 0;

        if (i % 625 == 0)
        {
            calka1 = 0;
            calka2 = 0;
        }
        calka1 += x1[i];
        calka2 += x2[i];

        p = calka2 - calka1;
        saveDemFSK_P << p << endl;

        if (p >= h)
        {
            mt[i] = 1;
        }
        else
        {
            mt[i] = 0;
        }
        saveDemFSK << mt[i] << endl;
    }

    saveDemFSK.close();
    saveDemFSK_P.close();

    return mt;
}

int main()
{
    string str = S2BS("123A", 1);
    //10000011100111100101100001 - Little Endian
    //S2BS("123A", 0);
    //11000111001011001110000001 - Big Endian

    double phi = 0;
    double Tb = 0.1;//sekundy
    int A1 = 1;
    int A = A1;
    int A2 = 10;
    int N = 2;
    int fs = 1000;
    double f = N * pow(Tb, -1);
    double f1 = (N + 1) / Tb;
    double f2 = (N + 2) / Tb;
    int n = lengthOfString(str);
    int probki = fs * Tb;
    int msize = n * probki * 8;
    int* m = Mgenerator(str, n, Tb, fs);

    double * asktab = ASK(m, msize, A1, A2, f, fs, phi);
    double * fsktab = FSK(m, msize, A, N, fs, Tb, phi);
    double * psktab = PSK(m, msize, A, f, fs, Tb);

    //demodulatorASKPSK(asktab, msize, 400, fs, f, A);
    demodulatorASKPSK(psktab, msize, 0, fs, f, A);
    //demodulatorFSK(fsktab, msize, 0, fs, f1, f2, A);
    return 1;
}
```

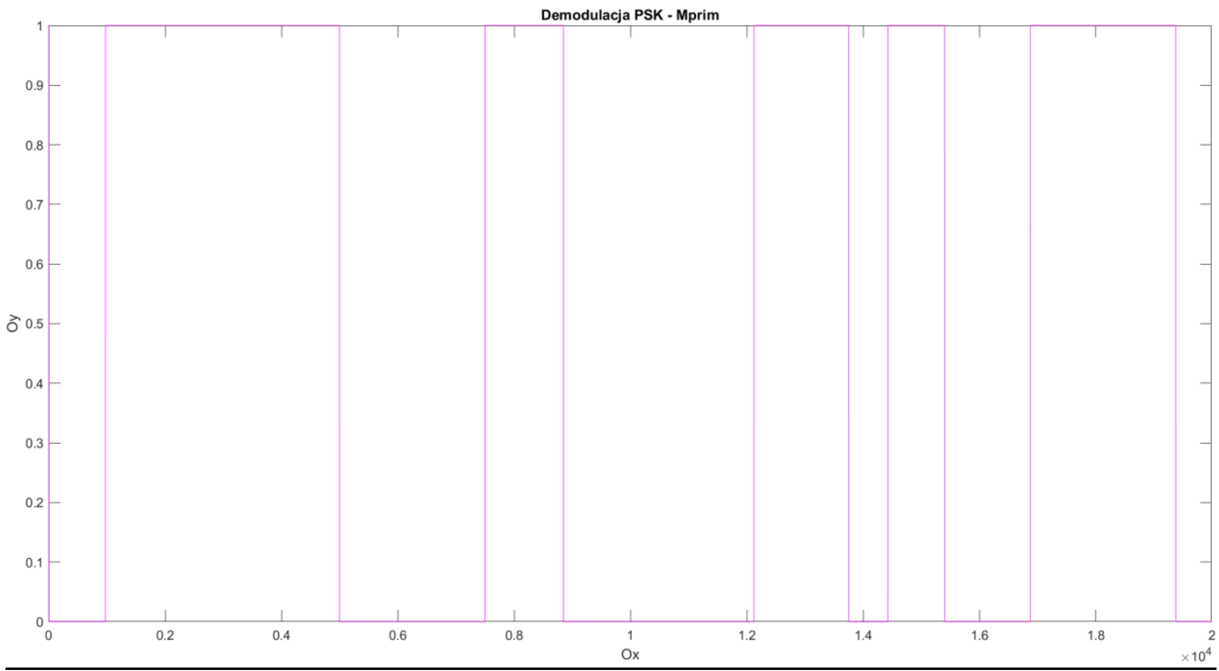
Wykresy:

PSK:

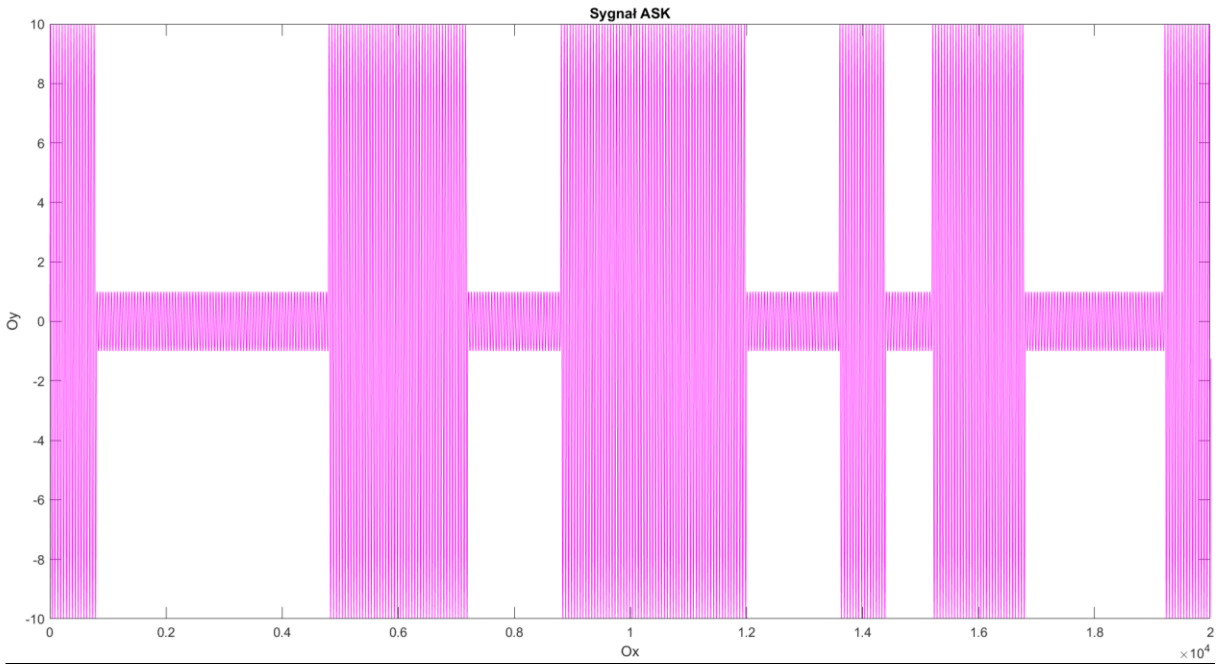


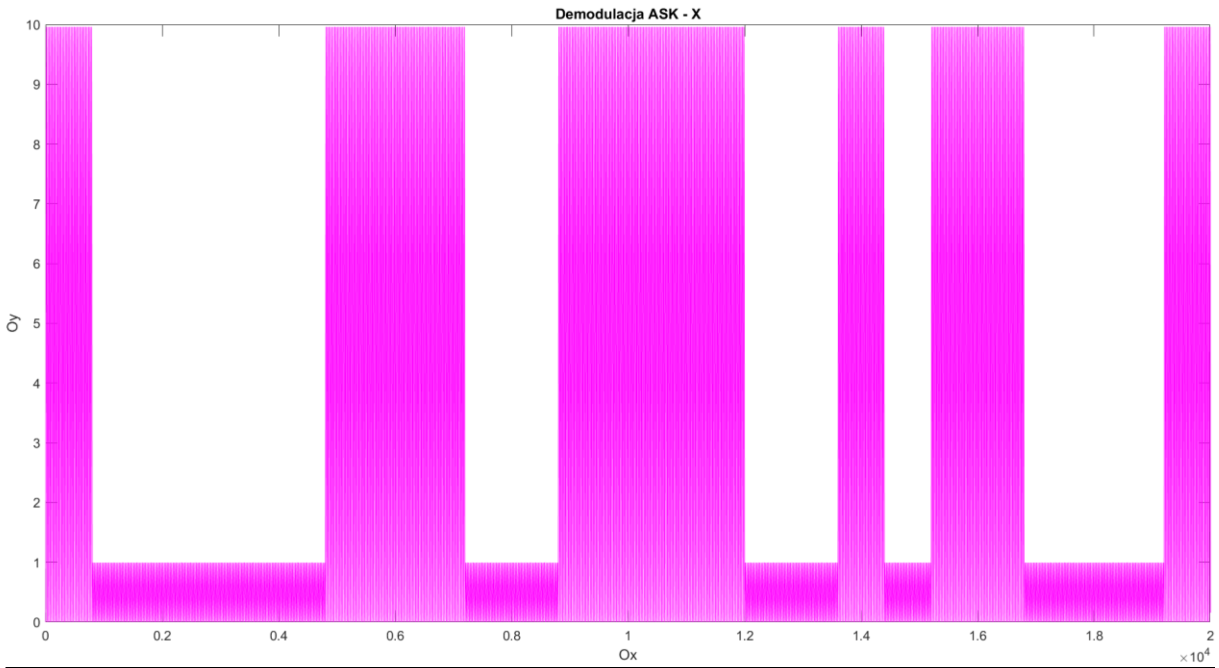


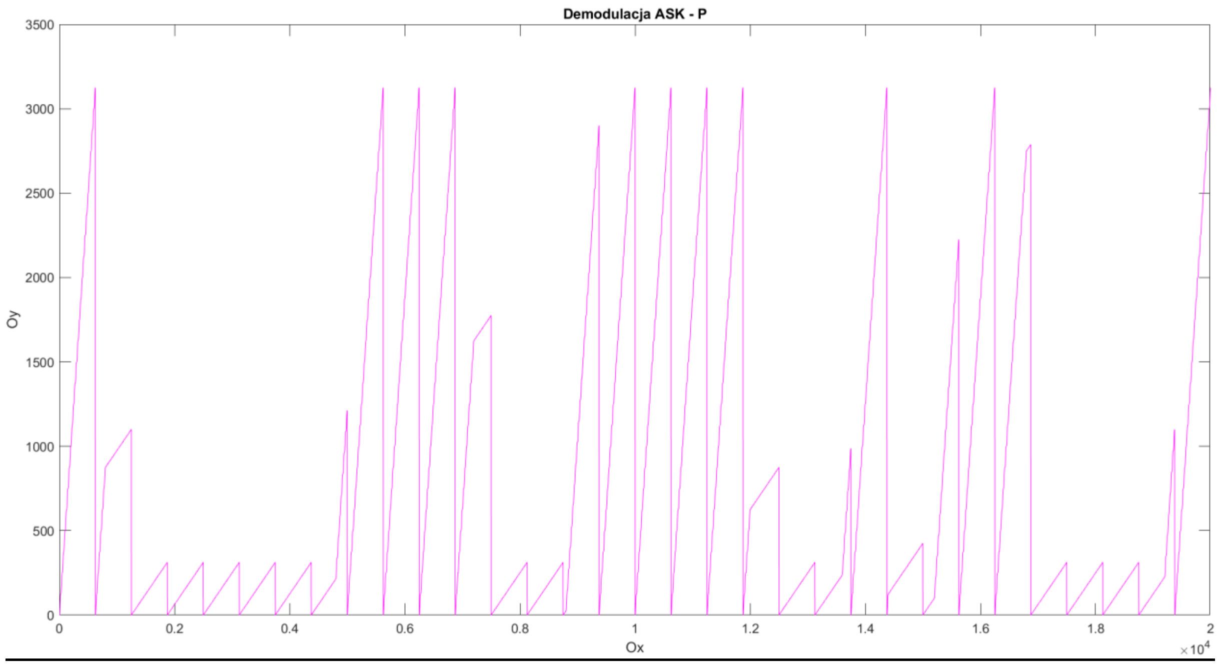

Demodulacja PSK - P
Oy
Ox
×10^4


Demodulacja PSK - Mprim
Oy
Ox
×10^4

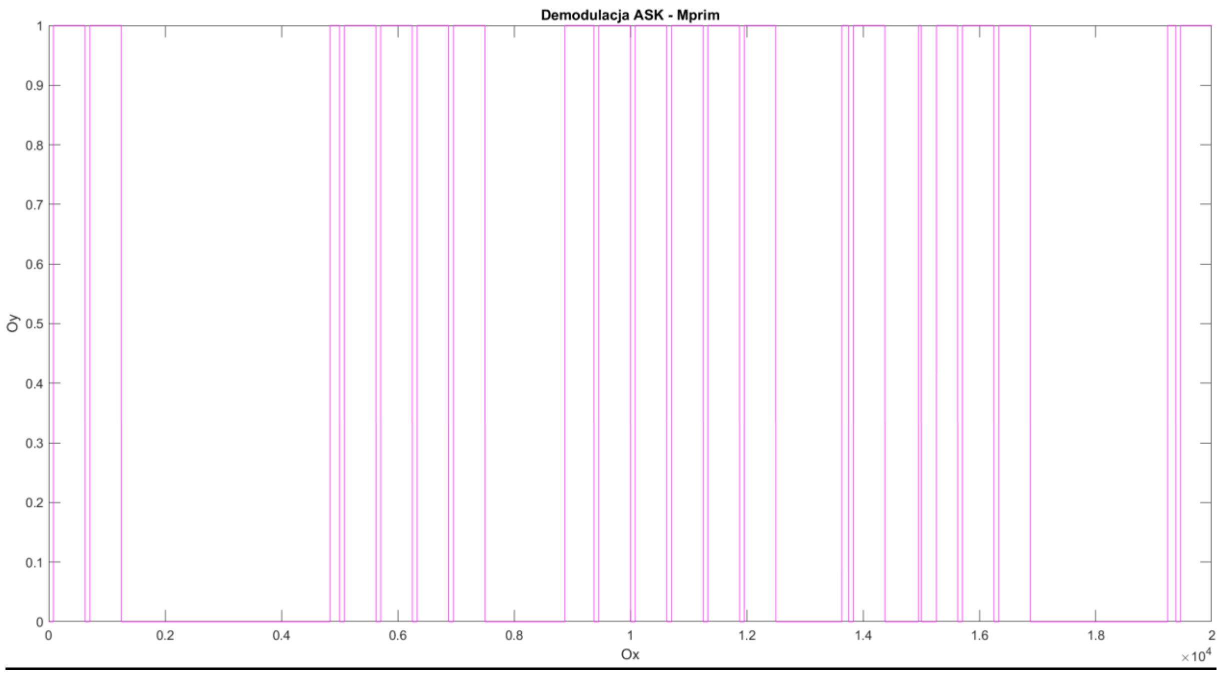
ASK:


Sygnał ASK
Oy
Ox
×10^4
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Demodulacja ASK - X
Oy
Ox
×10^4
10
9
8
7
6
5
4
3
2
1
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

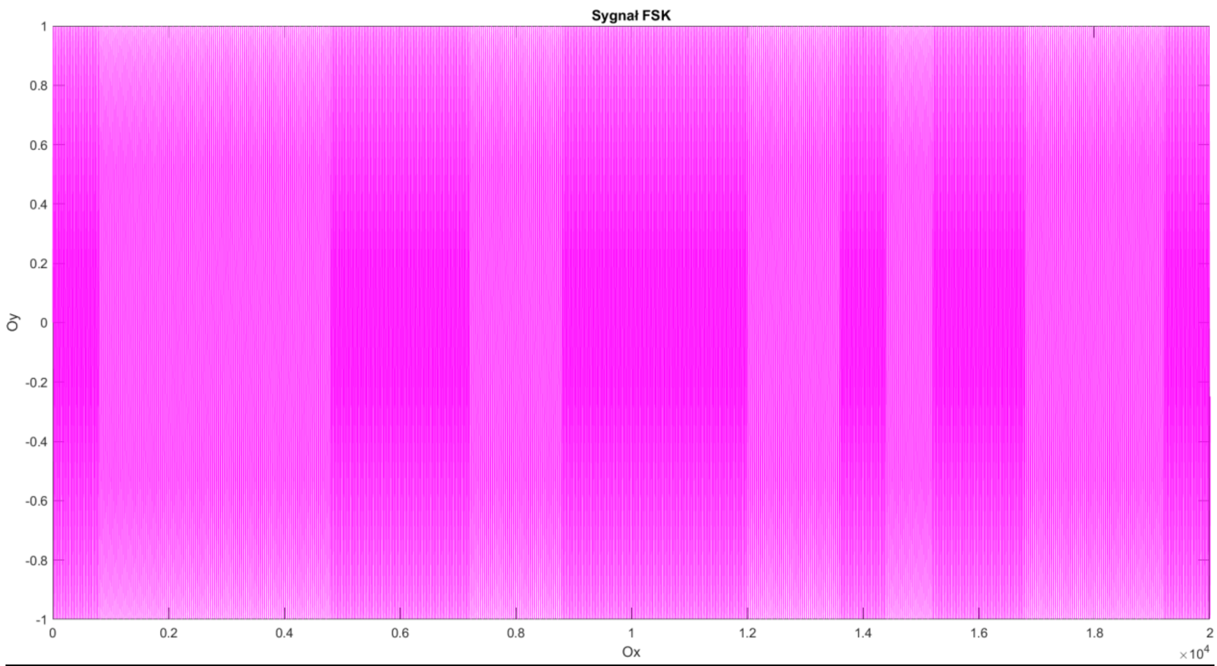


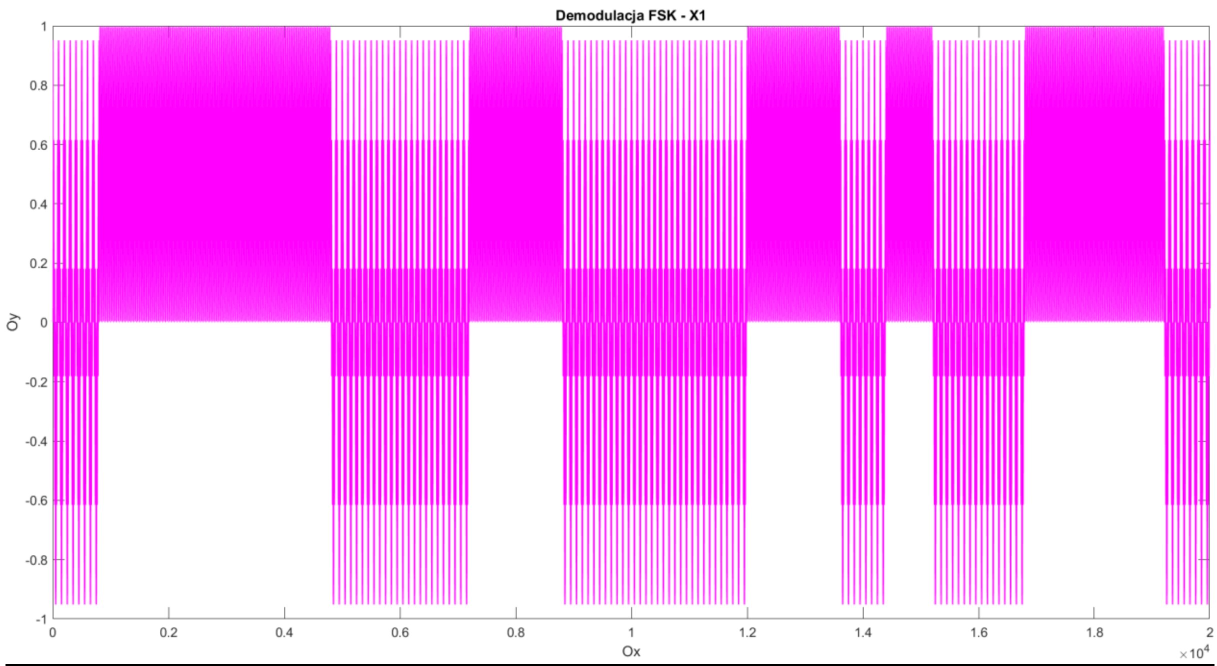
Demodulacja ASK - P
Oy
Ox
3500
3000
2500
2000
1500
1000
500
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4

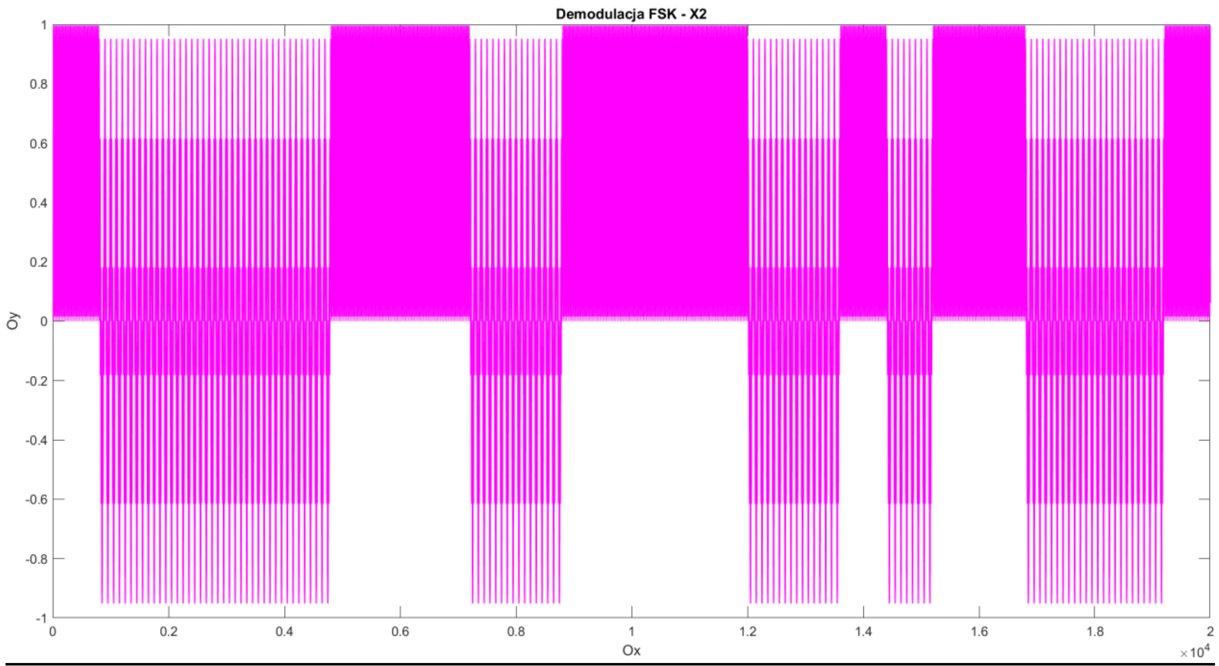


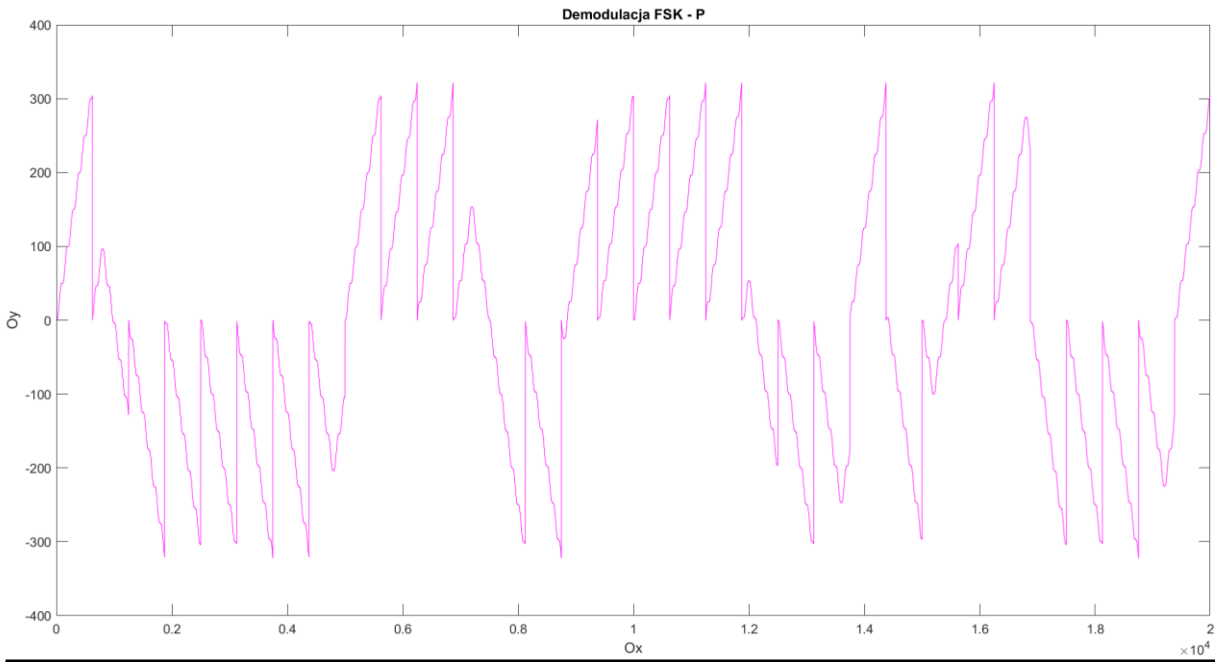
Demodulacja ASK - Mprim
Oy
Ox
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4

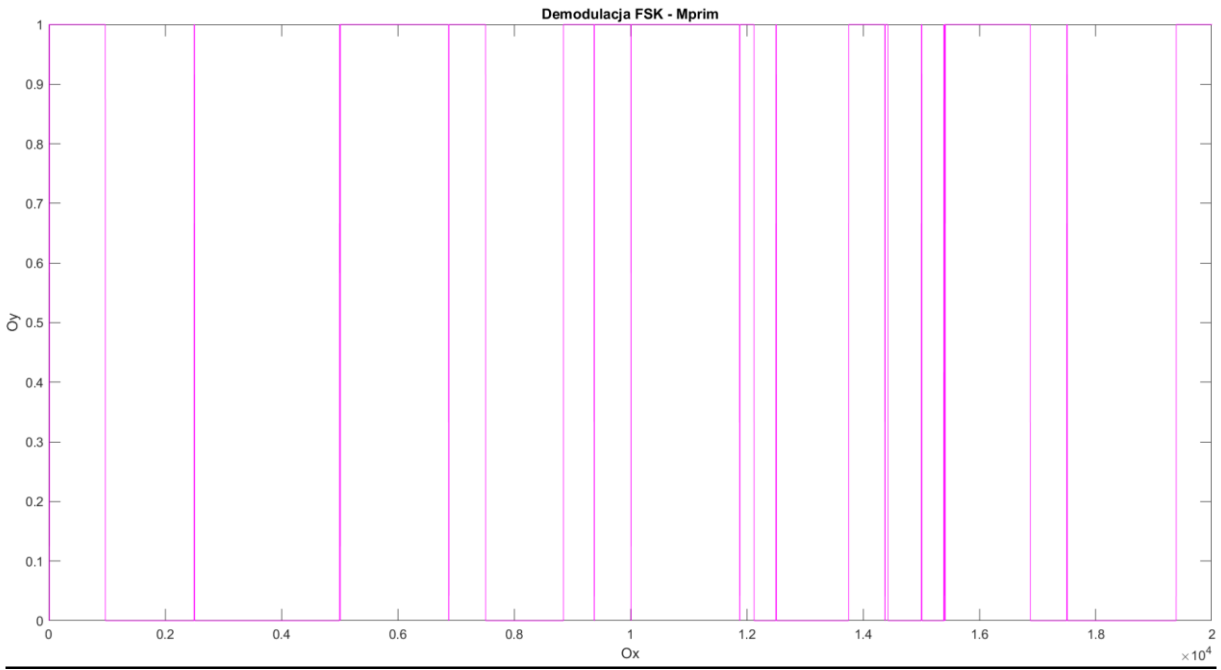
FSK:






Demodulacja FSK - X2
Oy
Ox
×10^4


Demodulacja FSK - P
Oy
Ox
×10^4


Demodulacja FSK - Mprim
Oy
Ox
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4