Таблица 1 – Зависимость от *p* при *l < k, l=2*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* | *0* | *0,1* | *0,2* | *0,3* | *0,4* | *0,5* | *0,6* | *0,7* | *0,8* | *0,9* | *1* |
|  | *0* | *0* | *0,0082* | *0,0050* | *0,1046* | *0,1242* | *0,0900* | *0,0982* | *0,1262* | *0,0649* | *0* |

Таблица 2 – Зависимость от *p* при *l > k, l=6*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* | *0* | *0,1* | *0,2* | *0,3* | *0,4* | *0,5* | *0,6* | *0,7* | *0,8* | *0,9* | *1* |
|  | *0* | *0,0132* | *0,0898* | *0,0917* | *0,1165* | *0,1073* | *0,1556* | *0,1172* | *0,1262* | *0,0809* | *0* |

Таблица 3 – Зависимость от *p* при *l = k, l=4*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* | *0* | *0,1* | *0,2* | *0,3* | *0,4* | *0,5* | *0,6* | *0,7* | *0,8* | *0,9* | *1* |
|  | *0* | *0* | *0,0307* | *0,0676* | *0,1069* | *0,1119* | *0,1665* | *0,1412* | *0,1759* | *0,4289* | *0,9865* |

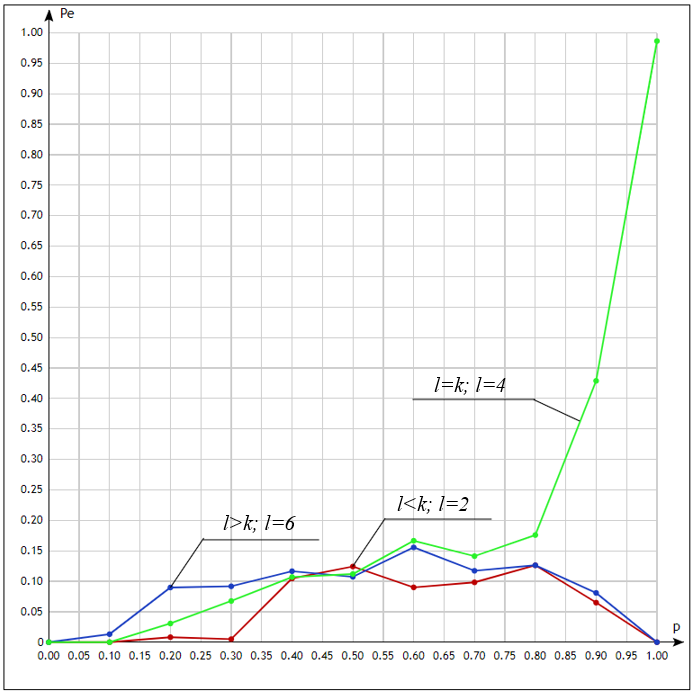


Рисунок 1 – графики зависимости оценки ошибки декодирования от вероятности ошибки в двоично-симметричном канале *p* при *l<k, l>k, l=k*

using System;

/\*

2) g(x)=x^3+x^2+1, d=3, k=4, n=7

2. Разработать программу, с помощью которой путем имитационного моделирования

оценивается вероятность ошибки декодирования при передаче данных по двоично-симметричному каналу.

Исходными данными для работы программы являются: порождающий многочлен g (x),

длина кодируемой последовательности l (может быть как больше, так и меньше k )

и точность ε, с которой программа оценивает вероятность ошибки декодирования.

С помощью программы студент должен исследовать зависимость вероятности ошибки декодирования

от значения вероятности появления ошибки в канале при различных значениях l.

a) Построить график зависимости оценки ошибки декодирования Pe

от вероятности ошибки в двоично-симметричном канале p

при l<k, l>k, l=k. Обосновать полученные зависимости.

\*/

namespace Bespr\_seti\_lab1

{

class Program

{

static int n = 0;

static int gen\_e(int t, double p) //генерация вектора ошибок в(1)

{

double m1 = 0;

int e = 0;

Random rnd = new Random();

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

{

m1 = rnd.NextDouble();

if (m1 >= p) e += 0;

else

{

e += 1;

}

if (i < t - 1) e = e << 1;

}

return e;

}

static int gen\_m(int t) //генерация m

{

double m1 = 0;

int a = 0, m = 0;

Random rnd = new Random();

while (a == 0)

{

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

{

m1 = rnd.NextDouble();

if (m1 >= 0.5) m += 0;

else

{

m += 1;

a++;

}

if (i < t - 1) m = m << 1;

}

}

return m;

}

static int delenie(int m, int g, bool s) //основной алгоритм

{

int prom\_rez = m, d\_m, d\_g, raz;

int m\_mas = 0;

int g1 = g;

d\_m = deg(m);

d\_g = deg(g);

raz = d\_m - d\_g;

if (raz < 0)

{

return m;

}

while (raz >= 0)

{

g1 = g;

g1 = g << raz;

prom\_rez = prom\_rez ^ g1;

d\_m = deg(prom\_rez);

raz = d\_m - d\_g;

}

if (s == false) m\_mas = m ^ prom\_rez;

else m\_mas = prom\_rez;

return m\_mas;

}

static int deg(int m) //определение макс степень

{

int i1 = 0;

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

{

if (m % 2 > 0) i1 = i;

m = m >> 1;

}

return i1;

}

static void Main(string[] args)

{

string otvet = "";

while (otvet != "q")

{

Console.WriteLine("Введите '1' для расчета определенной вероятности и '2' для получения значений для графика");

otvet = Console.ReadLine();

double pi = 0, e = 0;

Console.WriteLine("Введите g(x) в двоичном формате");

string g\_str = Console.ReadLine();

int g1 = Convert.ToInt32(g\_str);

Console.WriteLine("Введите L");

n = Convert.ToInt32(Console.ReadLine());

if (otvet == "1")

{

Console.WriteLine("Введите Pi - вероятность ошибки в бите");

pi = Convert.ToDouble(Console.ReadLine());

}

Console.WriteLine("Введите e - точность");

e = Convert.ToDouble(Console.ReadLine());

int j = 0;

double pd = 0.9;

ulong N = (ulong)(0.25 / ((1 - pd) \* e \* e));

Console.WriteLine($"N = {N}");

int d\_g = g\_str.Length - 1;

int m = 0;

int f = 0;

if (otvet == "2")

{

// Console.WriteLine("Введите m ");

// m = Convert.ToInt32(Console.ReadLine());

f = 10;

pi = 0;

}

else m = gen\_m(n);

n = n + d\_g;

int mx = m << d\_g;

int g = 0;

for (int j1 = 0; j1 < g\_str.Length; j1++)

{

g += (g1 % 10) << j1;

g1 = g1 / 10;

}

for (int k1 = 0; k1 <= f; k1++)

{

for (ulong i = N; i > 0; i--)

{

int deg\_g = d\_g;

int e\_vect = 0;

int b = 0;

int s = 0;

int a = 0;

// закомментить если что

m = gen\_m(n);

mx = m << d\_g;

//

a = delenie(mx, g, false);

e\_vect = gen\_e(n, pi);

b = a ^ e\_vect;

s = delenie(b, g, true);

if (s == 0)

{

if (e\_vect != 0) j++;

}

}

Console.WriteLine("Ошибка не обнаружена " + j + " раз");

Console.WriteLine($"Ошибка обнаружена { N - (ulong)j} раз");

Console.WriteLine($" Вероятность ошибки декодирования Pe при P({pi}) = { (double)((double)j / (double)N)} ");

j = 0;

if (otvet == "2") pi += 0.1;

}

}

}

}

}

Изображение выглядит как текст

Автоматически созданное описание

*l < k, l=2*

Изображение выглядит как текст

Автоматически созданное описание

*l > k, l=6*

Изображение выглядит как текст

Автоматически созданное описание

*l = k, l=4*