МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 52

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Бурков А.А.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

ИСПОЛЬЗОВАНИЕ ЦИКЛИЧЕСКИХ КОДОВ ДЛЯ ОБНАРУЖЕНИЯ ОШИБОК В СЕТЯХ ПЕРЕДАЧИ ДАННЫХ

по курсу: Сети и системы передачи информации

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5911		В.С. Кулешов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

1. Цель

Исследование типового алгоритма формирования контрольной суммы с использованием циклических кодов, использование численного расчета и имитационного моделирования для оценки вероятности того, что декодер не обнаружит ошибки.

2. Описание моделируемой системы

На вход кодера поступает некоторое информационное сообщение m, состоящее из нулей и единиц. Кодер по некоторому алгоритму вычисляет контрольную сумму, дописывает ее к передаваемому сообщению и таким образом формирует закодированное сообщение a так же состоящее из 0 и 1. В канале могут произойти ошибки, в результате которых некоторые биты сообщения инвертируются. Декодер по некоторому алгоритму проверяет контрольную сумму в принятом сообщении и принимает одно из следующих решений:

$$E = \begin{cases} 1, \text{если были ошибки} \\ 0, \text{если не было ошибок} \end{cases}.$$

Рис.1. Структурная схема системы передачи данных.

$\bar{m}$ – *информационное сообщение,* К – *блок кодера,*
$\bar{a}$ –*закодированное сообщение,* $\bar{e}$ –*вектор ошибок,*
$\bar{b}$ – *сообщение на выходе канала,* Д – *блок декодера,*
E –*принятое решение,* $\bar{m}'$ – *сообщение на выходе декодера*

Рассматривается модель двоично-симметричного канала (ДСК) без памяти

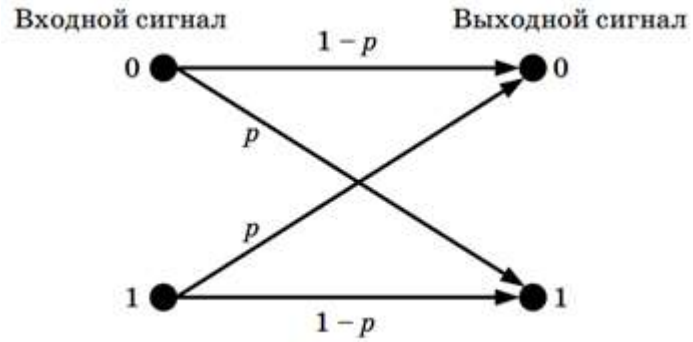

Рис.2. Модель двоично-симметричного канала.

Как видно, ошибка происходит с вероятностью p. Канал является двоичным, поэтому возможны только два значения битов на входе и выходе канала: {0,1}. Канал называется симметричным ввиду того, что вероятность ошибки для обоих значений битов одинакова.

Канал без памяти характеризуется тем, что случайные события, связанные с ошибками в канале независимы для разных моментов времени.

Кодер хранит порождающий многочлен $g(x)$. Степень многочлена обозначается как $deg(g(x)) = r$ и определяет количество бит контрольной суммы в кодовом слове. k – число информационных символов передаваемого сообщения $\overline{m}$.

Передаваемое сообщение рассматривается как вектор длины k. Для каждого сообщения $(\overline{m})$ кодер выполняет следующие действия:

- На основе вектора $\overline{m}$ формируется многочлен $m(x)$. Степень многочлена $m(x)$ при этом меньше или равна $k-1$;
- Вычисляется многочлен $c(x) = m(x)x^r mod(g(x))$. Степень многочлена $c(x)$ при этом меньше или равна $r-1$;
- Вычисляется многочлен $a(x) = m(x)x^r + c(x)$;
- На основе многочлена $a(x)$ формируется вектор $\bar{a}$, длина которого n бит, где $n = k + r$.

Декодер хранит $g(x)$ и n (длина кодового слова). Декодер выполняет следующие действия:

1. Принятое сообщение $\bar{b} = \bar{a} + \bar{e}$ переводится в многочлен $b(x)$;
2. Вычисляется синдром: $s(x) = b(x)mod(g(x))$;
3. Если $s(x) \neq 0$, то декодер выносит решение, что произошли ошибки $(E = 1)$, иначе декодер выносит решение, что ошибки не произошли $(E = 0)$

3. Описание проводимого исследования

Разработать программу вычисления верхней оценки для вероятности ошибки декодирования сверху и вычисления точного значения вероятности ошибки декодирования. Исходными данными для работы программы являются: порождающий многочлен g(x) и длина кодируемой последовательности *l* (может быть, как больше, так и меньше k).

a. **Вычисление верхней оценки ошибки декодирования**:

Рассмотрим два множества векторов ошибок:

$$A = \{\bar{e} \neq 0, E = 0\}$$
$$B = \{\bar{e}: w(\bar{e}) \geq d\}$$

Мощность множества B больше, чем мощность множества A. Предположим, что все вектора ошибок с весом $w(\bar{e}) \geq d$ приводят к ошибке декодирования. Используя данный подход, мы можем получить верхнюю границу для вероятности ошибки декодирования, для этого нужно найти $\Pr\{B\}$:

$$\Pr\{B\} = \Pr\{w(\bar{e}) = d \cup w(\bar{e}) = (d+1) \cup \ldots \cup w(\bar{e}) = n\} =$$

$$= \sum_{i=d}^{n} \Pr\{w(\bar{e}) = i\} = \sum_{i=d}^{n} C_n^i p^i (1-p)^{n-i}$$

Тогда верхнюю границу ошибки декодирования можно определить, как вероятность того, что вектор ошибки принадлежит множеству B:

$$P_e^+ = \Pr\{B\} = \sum_{i=d}^{n} C_n^i p^i (1-p)^{n-i}$$

b. **Вычисление точного значения ошибки декодирования**:

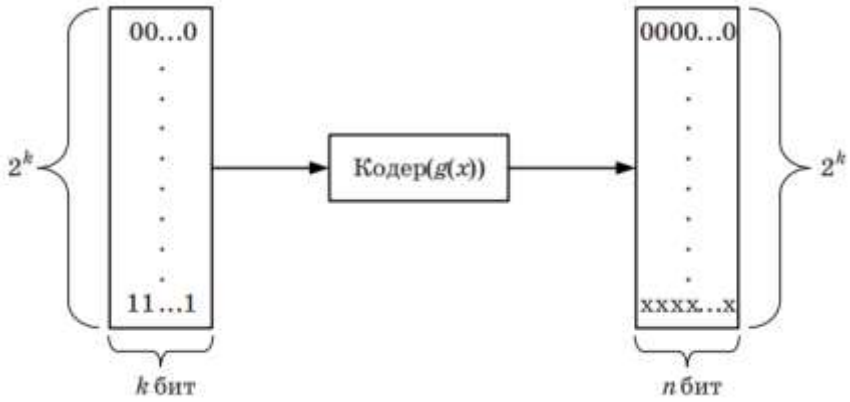

Рис.3. Схематичное представление формирования кодовых слов.

Пусть A – множество кодовых слов, $|A| = 2^k$; B –множество векторов ошибок, $|B| = 2^n$. Обозначим через A_i число кодовых слов веса i, где i – индекс от 0 до n. Для точного определения значения вероятности ошибки декодирования следует посчитать вероятность попадания вектора ошибок в множество A.

Теперь мы можем записать следующее выражение для вычисления точного значения вероятности ошибки декодирования:

$$P_e = \sum_{i=d}^{n} A_i p^i (1-p)^{(n-i)}$$

4. Результаты исследований

Входные данные для работы кодера:

$$k = 4,$$
$$g(x) = x^3 + x + 1$$

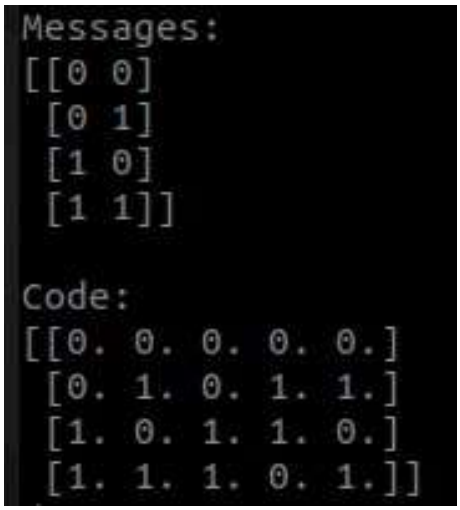

Рис.4. Сообщения и кодовые слова при l = 2.

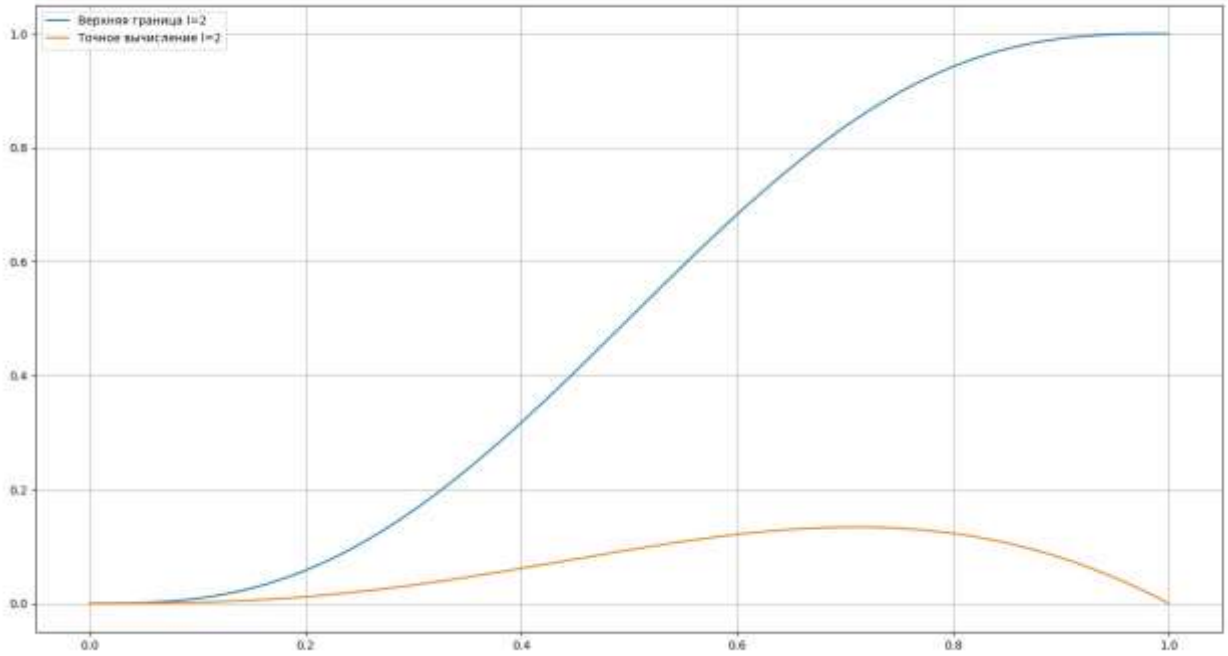

Рис.5. Верхняя граница и точное вычисление вероятности ошибки декодирования при l = 2.

```
Messages:
[[0 0 0]
 [0 0 1]
 [0 1 0]
 [0 1 1]
 [1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]]

Code:
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 1. 1.]
 [0. 1. 0. 1. 1. 0.]
 [0. 1. 1. 1. 0. 1.]
 [1. 0. 0. 1. 1. 1.]
 [1. 0. 1. 1. 0. 0.]
 [1. 1. 0. 0. 0. 1.]
 [1. 1. 1. 0. 1. 0.]]
```

Рис.6. Сообщения и кодовые слова при $l = 3$.

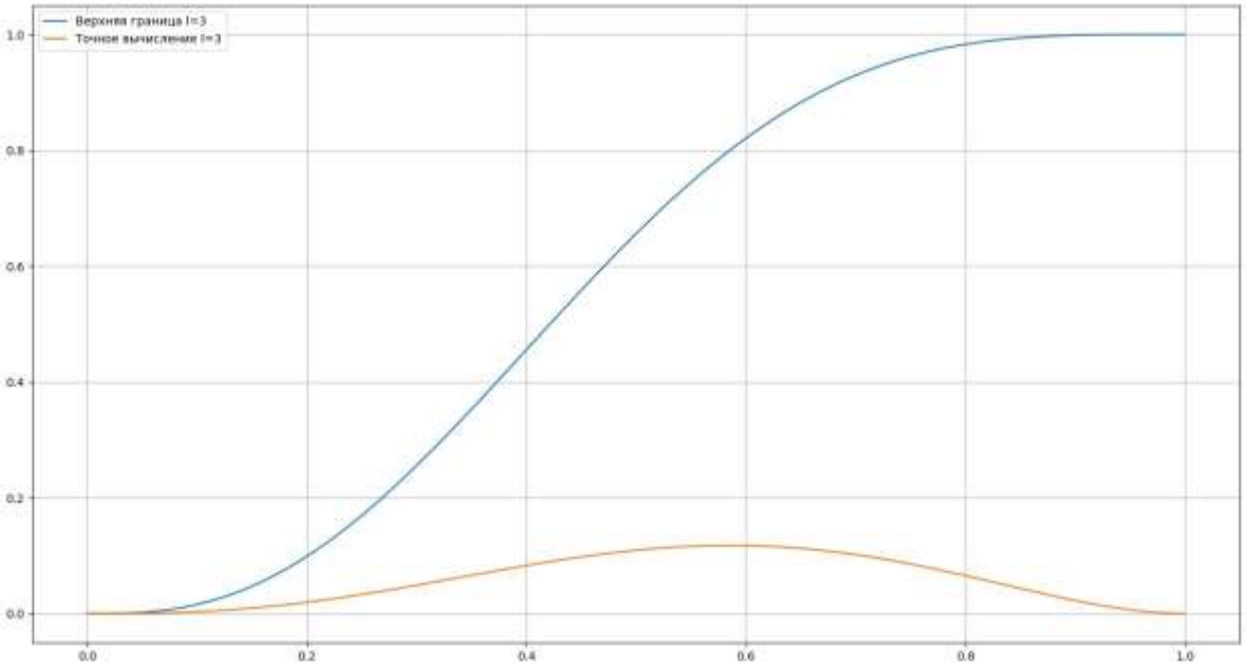

Рис.7. Верхняя граница и точное вычисление вероятности ошибки декодирования при $l = 3$.

```
Messages:
[[0 0 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [0 0 1 1]
 [0 1 0 0]
 [0 1 0 1]
 [0 1 1 0]
 [0 1 1 1]
 [1 0 0 0]
 [1 0 0 1]
 [1 0 1 0]
 [1 0 1 1]
 [1 1 0 0]
 [1 1 0 1]
 [1 1 1 0]
 [1 1 1 1]]

Code:
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 1. 1.]
 [0. 0. 1. 0. 1. 1. 0.]
 [0. 0. 1. 1. 1. 0. 1.]
 [0. 1. 0. 0. 1. 1. 1.]
 [0. 1. 0. 1. 1. 0. 0.]
 [0. 1. 1. 0. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 0.]
 [1. 0. 0. 0. 1. 0. 1.]
 [1. 0. 0. 1. 1. 1. 0.]
 [1. 0. 1. 0. 0. 1. 1.]
 [1. 0. 1. 1. 0. 0. 0.]
 [1. 1. 0. 0. 0. 1. 0.]
 [1. 1. 0. 1. 0. 0. 1.]
 [1. 1. 1. 0. 1. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1.]]
```

Рис.8. Сообщения и кодовые слова при $l = 4$.

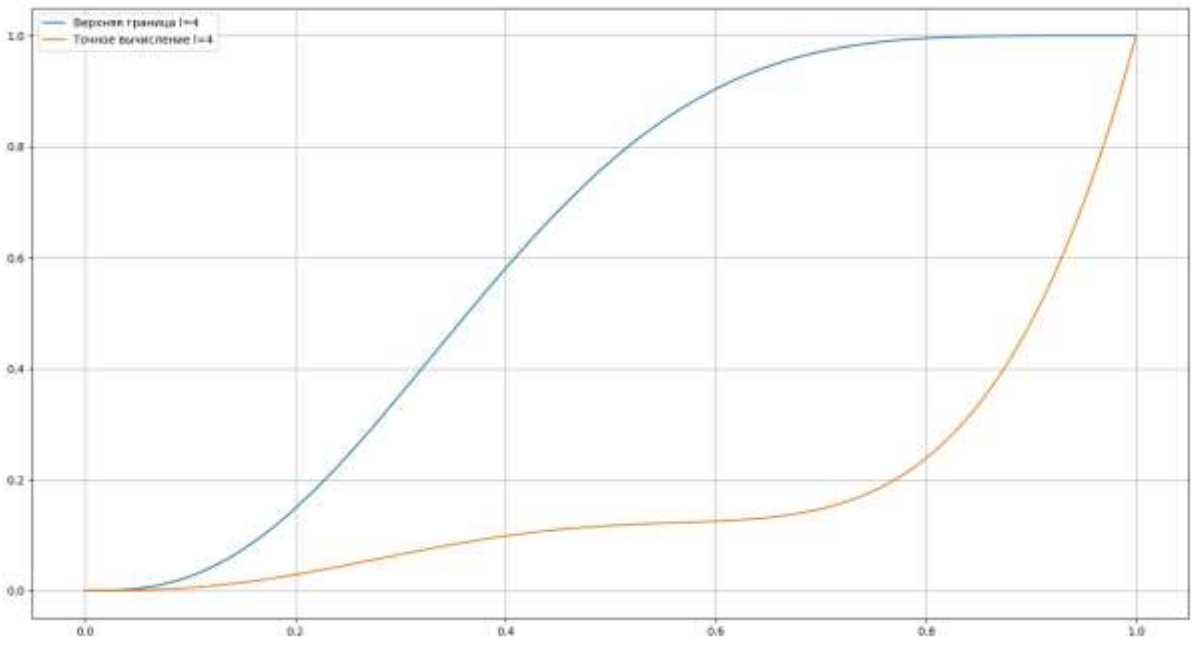

Рис.9. Верхняя граница и точное вычисление вероятности ошибки декодирования при $l = 4$.

```
Messages:
[[0 0 0 0 0]
 [0 0 0 0 1]
 [0 0 0 1 0]
 [0 0 0 1 1]
 [0 0 1 0 0]
 [0 0 1 0 1]
 [0 0 1 1 0]
 [0 0 1 1 1]
 [0 1 0 0 0]
 [0 1 0 0 1]
 [0 1 0 1 0]
 [0 1 0 1 1]
 [0 1 1 0 0]
 [0 1 1 0 1]
 [0 1 1 1 0]
 [0 1 1 1 1]
 [1 0 0 0 0]
 [1 0 0 0 1]
 [1 0 0 1 0]
 [1 0 0 1 1]
 [1 0 1 0 0]
 [1 0 1 0 1]
 [1 0 1 1 0]
 [1 0 1 1 1]
 [1 1 0 0 0]
 [1 1 0 0 1]
 [1 1 0 1 0]
 [1 1 0 1 1]
 [1 1 1 0 0]
 [1 1 1 0 1]
 [1 1 1 1 0]
 [1 1 1 1 1]]
```

```
Code:
[[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 1. 1.]
 [0. 0. 0. 1. 0. 1. 1. 0.]
 [0. 0. 0. 1. 1. 1. 0. 1.]
 [0. 0. 1. 0. 0. 1. 1. 1.]
 [0. 0. 1. 0. 1. 1. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 1.]
 [0. 0. 1. 1. 1. 0. 1. 0.]
 [0. 1. 0. 0. 0. 1. 0. 1.]
 [0. 1. 0. 0. 1. 1. 1. 0.]
 [0. 1. 0. 1. 0. 0. 1. 1.]
 [0. 1. 0. 1. 1. 0. 0. 0.]
 [0. 1. 1. 0. 0. 0. 1. 0.]
 [0. 1. 1. 0. 1. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 0. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 1. 0.]
 [1. 0. 0. 1. 0. 1. 1. 1.]
 [1. 0. 0. 1. 1. 1. 0. 0.]
 [1. 0. 1. 0. 0. 1. 1. 0.]
 [1. 0. 1. 0. 1. 1. 0. 1.]
 [1. 0. 1. 1. 0. 0. 0. 0.]
 [1. 0. 1. 1. 1. 0. 1. 1.]
 [1. 1. 0. 0. 0. 1. 0. 0.]
 [1. 1. 0. 0. 1. 1. 1. 1.]
 [1. 1. 0. 1. 0. 0. 1. 0.]
 [1. 1. 0. 1. 1. 0. 0. 1.]
 [1. 1. 1. 0. 0. 0. 1. 1.]
 [1. 1. 1. 0. 1. 0. 0. 0.]
 [1. 1. 1. 1. 0. 1. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 0.]]
```

Рис.10. Сообщения и кодовые слова при $l = 5$.

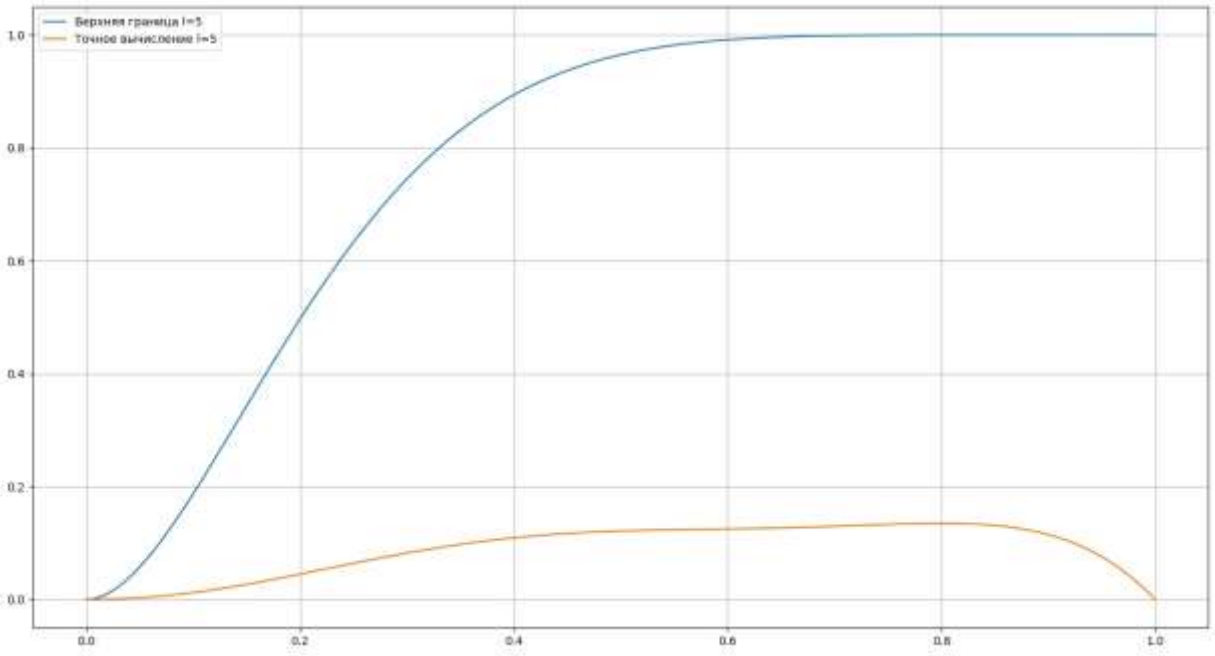

Рис.11. Верхняя граница и точное вычисление вероятности ошибки декодирования при $l = 5$.

Исходя из полученных графиков, можно заметить, что если среди кодовых слов есть вектор, состоящий из всех единиц и вероятность ошибки p = 1, то вероятность ошибки декодирования Pe = 1, т.к. вектор ошибок всегда будет кодовым словом $e(x)mod\big(g(x)\big) = 0$.

Дополнительное задание:
Построить зависимости верхней оценки вероятности ошибки декодирования $\widehat{Pe}$ и точной вероятности ошибки декодирования Pe при различных порождающих многочленах $g(x)$. Исследовать, как влияет изменение порождающего многочлена на изменение вероятности ошибки декодирования.

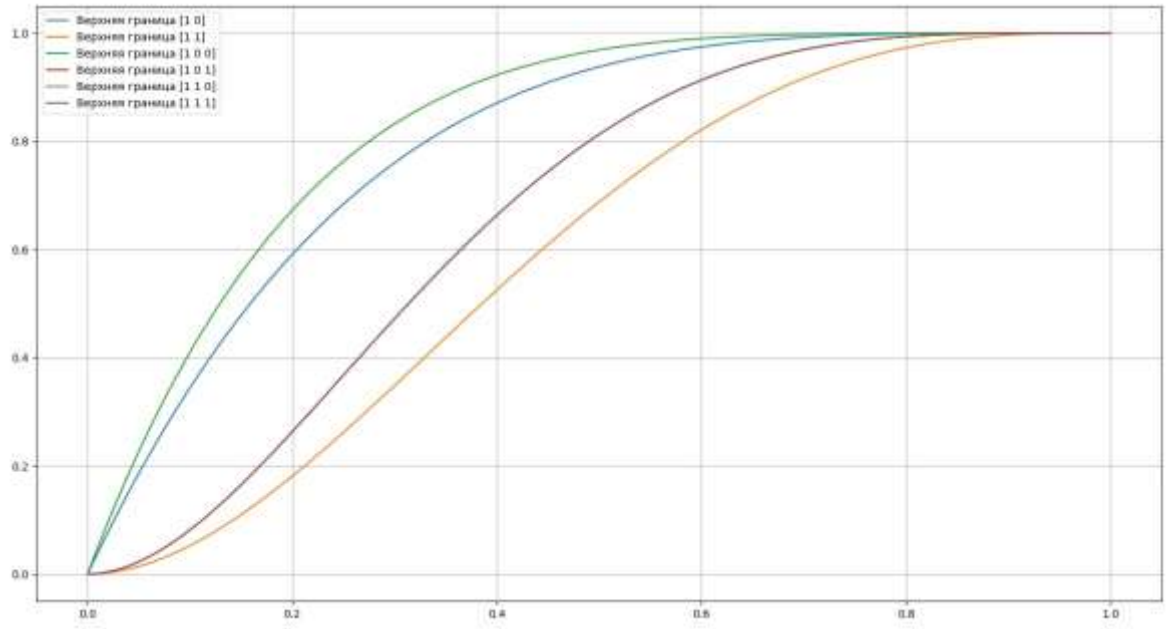

Рис.12. Верхняя граница вероятности ошибки декодирования для многочленов максимальной степени 3 и сообщений длины 3.

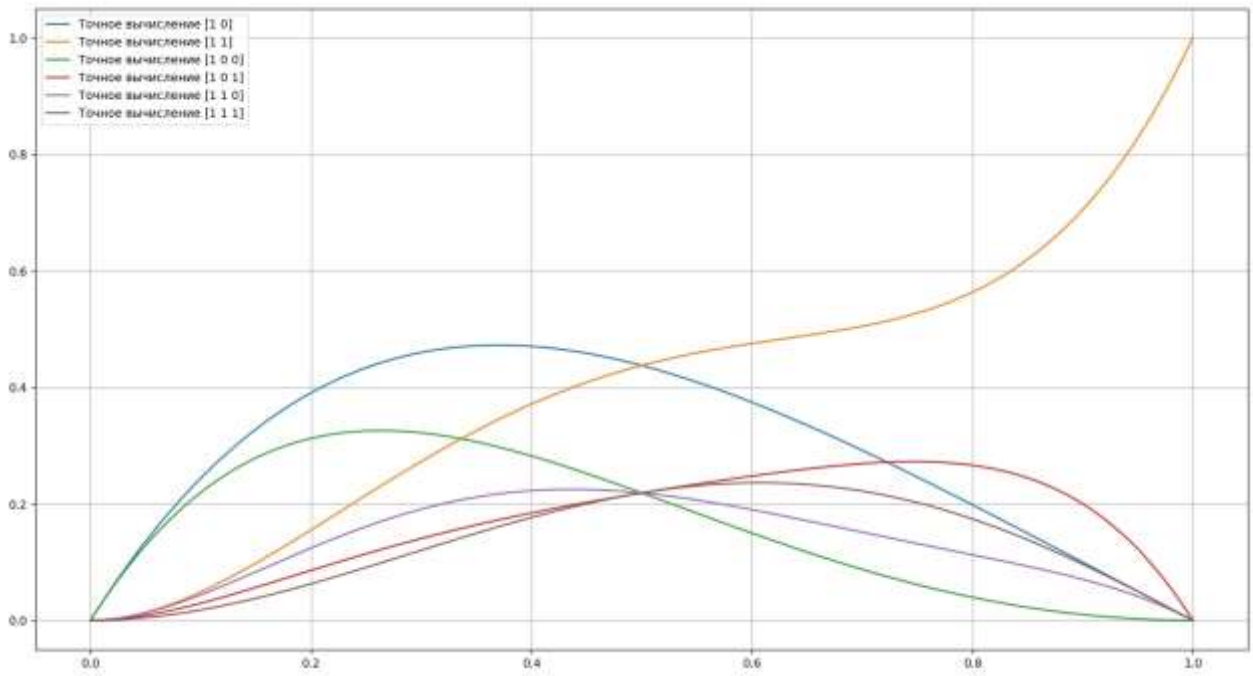

Рис.13. Точное вычисление вероятности ошибки декодирования для многочленов максимальной степени 3 и сообщений длины 3.

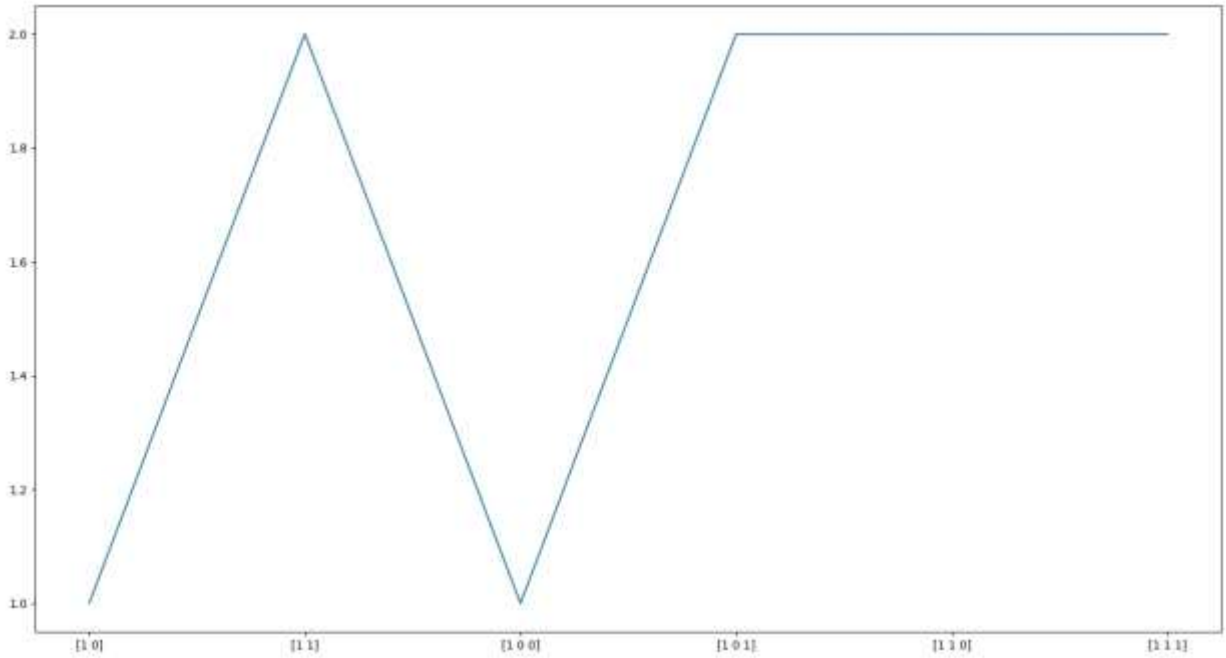

Рис.14. Минимальный вес кодовых слов для многочленов максимальной степени 3 и сообщений длины 3.

Исходя из графиков вычисления верхней границы вероятности ошибки декодирования, можно сделать вывод, что она тем ниже, чем больше минимальный вес и меньше длина кодовых слов. Из графиков вычисления точного значения видно, что вероятность ошибки декодирования становится ниже при увеличении веса кодовых слов в среднем.

5. Выводы

В результате лабораторной работы был изучен и исследован алгоритм формирования контрольной суммы с использованием циклических кодов. Был разработан программный модуль для вычисления верхней границы и точного значения вероятности ошибки декодирования.

По результатам программы можно сделать вывод о том, что если среди кодовых слов есть вектор, состоящий из всех единиц и вероятность ошибки равна 1, то и вероятность ошибки декодирования Pe тоже будет равна 1.

Была исследована зависимость изменения верхней границы и точного вычисления вероятности ошибки декодирования. Наилучшие результаты верхней оценки получаются при наименьшей длине и наибольшем минимальном весе кодовых слов. А наилучшие результаты точного вычисления вероятность ошибки декодирования получаются при максимальной средней длине кодовых слов.

6. Листинг программы

main.py

```
import numpy as np
import numpy.random as random
import itertools as it
import matplotlib.pyplot as plt

def toMath(polynom):
      if 1 not in polynom:
            return "0"
      string = ""
      for i in range(len(polynom)):
            if polynom[i] != 0:
                  if i != len(polynom)-1:
                        string += "x^" + str(len(polynom)-1-i) + " + "
                  else:
                        string += "1 + "
      return string[:len(string)-3]

def clearBegin(g):
      indexes = []
      for i in range(len(g)-1):
            if g[i] == 0:
                  indexes.append(i)
            else:
                  break
      g = np.delete(g, indexes)
      return g

def deg():
      index = 0
      for i in range(len(gx)):
            if gx[i] == 1:
                  break
            index = i
      return len(gx)-index-1

def getXr():
      index = deg()
      xr = []
      for i in range(index):
            xr.append(0)
      return np.array(xr)

def mul_xr(m):
      return np.append(m, getXr())

def bin_mod(m):
      for i in range(len(m)):
            if np.isnan(m[i]):
                  m[i] = 0
            m[i] = (int)(m[i]%2)
      return m

def code(msg):
      mxr = mul_xr(msg)
      _,c = np.polydiv(mxr, gx)
      c = bin_mod(c)
      return np.polyadd(mxr, c)
```

```
def plot(pes, p, legends):
      for pe in pes:
            plt.plot(p, pe)
      plt.legend(legends)
      plt.grid(True)
      plt.show()

def C(n, k):
      return len(list(it.combinations(range(0, n), k)))

def getMsgsOld(l):
      msgs = np.zeros((2**l, l))
      for r in range(2**l):
            msg = np.unpackbits(np.uint8(r))
            msgs[r] = np.array(msg[len(msg)-l:])
      return msgs

def getbit(num, index):
      if (num & (1 << index)) == 0:
            return 0
      else:
            return 1

def getMsgs(l):
      msgs = []
      for i in range(2**l):
            msg = []
            for x in range(l):
                  msg.insert(0,getbit(i, x))
            msgs.append(msg)
      return np.array(msgs)

def getWords(l):
      words = np.zeros((2**l, l+deg()))
      msgs = getMsgs(l)
      for i in range(len(msgs)):
            words[i] = code(msgs[i])
      return words

def isNull(word):
      for i in range(len(word)):
            if word[i] == 1:
                  return False
      return True

def getAllD(words):
      d = []
      for i in range(len(words)):
            if False == isNull(words[i]):
                  d.append(getWeight(words[i]))
      # for i in range(len(words)):
      #     for j in range(len(words)):
      #           if i != j:
      #                 d.append(getD(words[i],words[j]))
      return np.array(d)

def calculateD(l):
      words = getWords(l)
      d = getAllD(words)
      return np.min(d)

def Pe_ceil(d, n, p):
      summ = 0
      for i in range(0, d):
```

```
            summ += C(n, i)*(p**i)*((1-p)**(n-i))
      return 1-summ

def getP_Pe_ceil(p, l):
      Pe_c = np.zeros(len(p))
      d = calculateD(l)
      print("gx:", gx)
      print("d:", d)
      print("n:", l+deg())
      for i in range(len(p)):
            Pe_c[i] = Pe_ceil(d, l+deg(), p[i])
            # print(Pe_c[i])
      return Pe_c


def getWeight(word):
      d = 0
      for i in range(len(word)):
            if (word[i] == 1):
                  d += 1
      return d

def Ai(words, i):
      ai = 0
      for x in range(len(words)):
            if getWeight(words[x]) == i:
                  ai += 1
      return ai

def Pe(d, n, p, words):
      summ = 0
      for i in range(d, n+1):
            summ += Ai(words, i)*(p**i)*((1-p)**(n-i))
      return summ

def getP_Pe(p, l):
      Pe_ = np.zeros(len(p))
      d = calculateD(l)
      # print("d:", d)
      words = getWords(l)
      for i in range(len(p)):
            Pe_[i] = Pe(d, len(words[0]), p[i], words)
            # print(Pe_[i])
      return Pe_

def dopusk5task():
      words = getWords(5)
      for word in words:
            if(getWeight(word) < 3):
                  print(word)


def mainCode(g, l, p):
      global gx
      gx = clearBegin(np.array(g))
      pe_ceil = getP_Pe_ceil(p, l) # Если w(e(x)) > d -> (w(e(x)) - вес вектора
ошибок) ; (d - минимальное расстояние кода)
      pe = getP_Pe(p, l) # Если e(x) == a(x) -> если вектор ошибок является
кодовым словом
      plot([pe_ceil, pe], p, ["Верхняя граница l=" +str(l), "Точное вычисление
l=" + str(l)])

def main(gx, p):
      for i in range(2,7):
```

```
            print("Messages:")
            print(getMsgs(i))
            print()
            print("Code:")
            print(getWords(i))
            mainCode(gx, i, p)

p = []
for i in range(0,1001, 1):
      p.append(i/1000)
p = np.array(p)
# p = np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])

def dop(p,l):
      global gx
      pe_ceils = []
      pe_ceils_labels = []
      pes = []
      pes_labels = []
      Dmins = []
      Dmins_labels = []
      Dmins_x = []
      gxs = getMsgs(l)
      for i in range(2,len(gxs)):
            gx = clearBegin(np.array(gxs[i]))

            pe_ceils.append(getP_Pe_ceil(p, l))
            pe_ceils_labels.append("Верхняя граница " + str(gx))

            pes.append(getP_Pe(p, l))
            pes_labels.append("Точное вычисление " + str(gx))

            Dmins.append(int(calculateD(l)))
            Dmins_x.append(str(gx))
            Dmins_labels.append("Dmin " + str(gx))

      plot(pe_ceils, p, pe_ceils_labels)
      plot(pes, p, pes_labels)
      plt.plot(Dmins_x, Dmins)
      plt.show()


gx = clearBegin(np.array([0,0,1,1,1]))
l = 4

main(gx, p)
dop(p, 3)
# dopusk5task()
```