

## Оглавление

2 Описание моделируемой системы.....	3
2.1 Использование циклических кодов для обнаружения ошибок.....	3
2.2 Описание алгоритма передачи.....	4
3 Список допущений рассматриваемой модели.....	5
4 Описание моделирующей программы в виде псевдокода .....	6
4.1 Класс CyclicCodes .....	6
4.2 Класс BacktrackingAlgorithmP0.....	6
5 Теоретические расчеты.....	7
5.1 Вероятность возникновения $t$ ошибок в канале.....	7
5.2 Вероятность ошибки декодирования .....	7
5.3 Зависимость вероятности ошибки в канале от ошибки на бит .....	9
5.4 Вычисление зависимости коэффициента использования канала от ошибки на бит .....	10
6 Сравнение практических и теоретических значений коэффициента использования канала для заданных параметров .....	10
7 Выводы.....	16
<i>Листинг кода</i> .....	17

**Цель работы:** моделирование и анализ работы системы передачи данных с использованием контрольной суммы на основе циклических избыточных кодов. Нахождение зависимости ошибки в канале от ошибки на бит и зависимости коэффициента использования канала от ошибки на бит.

## 1 Описание моделируемой системы

### 1.1 Использование циклических кодов для обнаружения ошибок

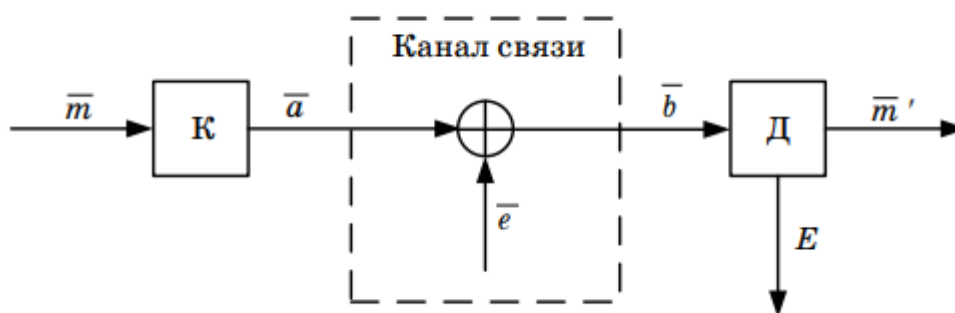


Рисунок 1 - Структурная схема передачи данных

где  $\bar{m}$  – информационное сообщение,  $K$  – кодер,  $\bar{a}$  – закодированное сообщение,  $\bar{e}$  – вектор ошибки,  $\bar{b}$  – сообщение на выходе канала,  $D$  – декодер,  $E$  – принятое решение,  $\bar{m}'$  – сообщение на выходе декодера.

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 а так же состоящее из 0 и 1.

В канале могут произойти ошибки, в результате которых некоторые биты сообщения инвертируются (0 становится 1 или 1 становится 0) с вероятностью ошибки на бит  $p_0$ . Вектор ошибок показывает на каких позициях произошла ошибка, при этом канал может быть описан как операция XOR передаваемого сообщения и вектора ошибок.

Декодер по некоторому алгоритму проверяет контрольную сумму в принятом сообщении и принимает одно из следующих решений:

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

Декодер может допустить ошибку в принятии решения с вероятностью ошибки декодирования  $P_e$ .

## 1.2 Описание алгоритма передачи

В качестве алгоритма передачи используется алгоритм с возвратом.

Алгоритм с возвратом работает следующим образом. Источник непрерывно передает сообщения, не дожидаясь квитанции на отправленное ранее сообщение. При задержке получения квитанции равной  $\tau$  единицам времен, после передачи сообщения, абонент успевает передать еще  $\tau$  сообщений, до того, как получит квитанцию на него. При получении отрицательной квитанции на сообщение источник повторяет передачу этого сообщения и всех следующих за ним сообщений, которые он успел передать за это время. Приемник после отправки отрицательной квитанции удаляет  $\tau$  пришедших за ним сообщений, даже если они были приняты без ошибок.

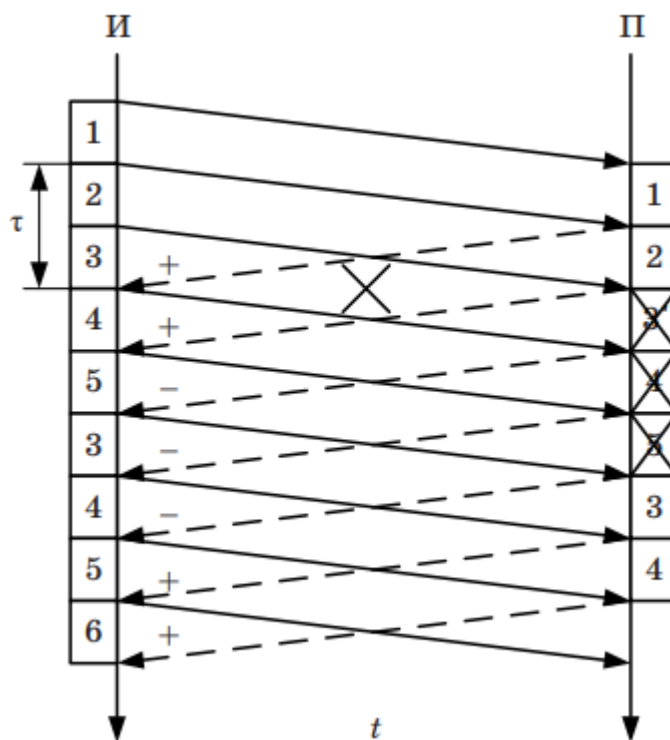


Рисунок 2 - Пример работы алгоритма с возвратом при  $\tau = 2$

## 2 Список допущений рассматриваемой модели

Предполагается, что сообщения, передаваемые по прямому каналу, состоят из данных и контрольной суммы. Использование контрольной суммы позволяет на приемной стороне определить наличие ошибок. Канал передачи от источника к получателю называется прямым каналом. Канал передачи от приемника к источнику называется обратным каналом.

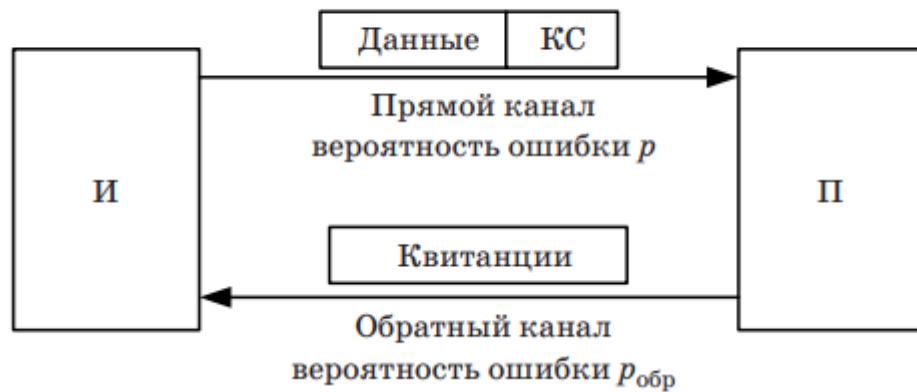


Рисунок 3 - Схема передачи данных по каналу с обратной связью

где И — источник сообщений, П — приемник, КС — контрольная сумма для передаваемых данных.

Введем следующие допущения:

1) Приемник проверяет полученное сообщение на наличие ошибок. Если ошибок нет, он отправляет положительную квитанцию по обратному каналу, а данные передает на дальнейшую обработку. В противном случае получатель отправляет отрицательную квитанцию, а данные стирает.

2) При передаче квитанции по обратному каналу ошибок нет.

3) В прямом канале может произойти ошибка с вероятностью  $P$ .

4) Все сообщения, которые передает источник, имеют одинаковую длину. Время передачи сообщения принято за единицу времени, а время передачи квитанции считается равным нулю. Источник получает квитанцию о результате передачи через  $\tau$  единиц времени после окончания передачи сообщения, где  $\tau$  — целое число.

### 3 Описание моделирующей программы в виде псевдокода

#### 3.1 Класс *CyclicCodes*

Данный класс служит моделирования отправки сообщения, для генерации, кодирования и декодирования циклических кодов. Он включает в себя следующие методы:

- *CyclicCodes(String g, int k)*

Конструктор класса, который создает объект и принимает в качестве параметров порождающий многочлен в бинарной форме и длину информационной последовательности.

- *generateMessage()*

Данный метода генерирует случайную информационную последовательность длины  $k$ .

- *encoding()*

Данный методы кодируется информационную последовательность в кодовое слово.

- *generateE()*

Данный метод генерирует вектор ошибки, где каждый бит последовательности генерируется с учетом вероятности ошибки на бит  $p_0$ .

- *boolean decoding()*

Данный метод декодирует вектор, полученный сложением по модулю 2 кодового слова и сгенерированного вектора ошибки. Принимается решение о наличии ошибок.

- *boolean messageSendingModeling()*

Данный метод моделирует отправки сообщения и включает в себя последовательный вызов следующих методов: *encoding()*, *generateE()* и *decoding()*.

#### 3.2 Класс *BacktrackingAlgorithmP0*

Данный класс моделирует работу алгоритма с возвратом. Он включает в себя следующие методы:

- *sending()*

Данный метода моделирует отправку сообщения и обращается к методу *CyclicCodes.messageSendingModeling()*.

○ *markUnset()*

Данный метод помечает сообщение, как неотправленное. Вызывается в том случае, когда при передаче сообщения с меньшим порядковым номером возникает ошибка в канале.

○ *modeling()*

Данный метод моделирует работу алгоритма с возвратом.

## 4 Теоретические расчеты

### 4.1 Вероятность возникновения $t$ ошибок в канале

Пусть необходимо найти вероятность возникновения  $t$  ошибок, независимо от их позиций в векторе ошибки  $\bar{e}$  длины  $n$ .

$$\Pr\{w(\bar{e}) = 0\} = (1 - p_0)^n$$

$$\Pr\{w(\bar{e}) = 1\} = C_n^1 p_0 (1 - p_0)^{n-1}$$

$$\Pr\{w(\bar{e}) = 2\} = C_n^2 p_0^2 (1 - p_0)^{n-2}$$

...

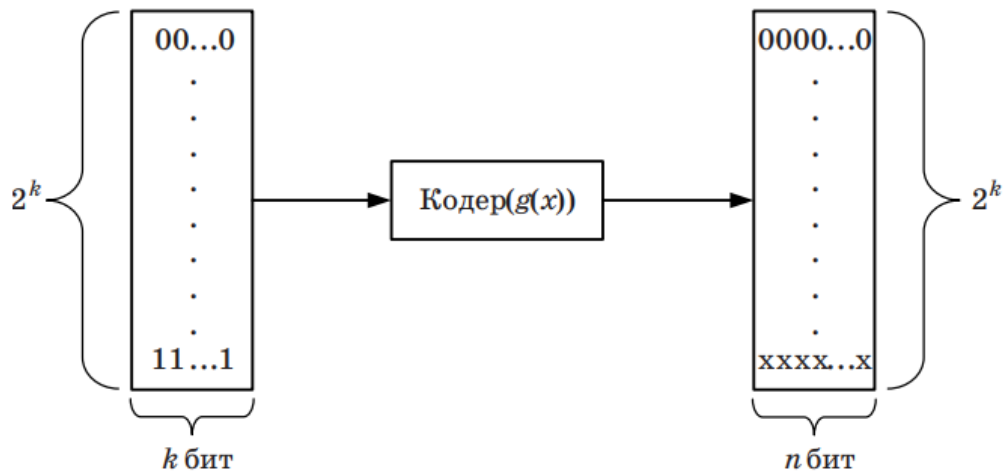
$$\Pr\{w(\bar{e}) = t\} = C_n^t p_0^t (1 - p_0)^{n-t}$$

где  $t$  — число ошибок,  $\bar{e}$  — вектор ошибки,  $n$  — длина кодового слова,  $p_0$  — вероятность ошибки на бит.

### 4.2 Вероятность ошибки декодирования

Пусть заданы: порождающий многочлен  $g(x)$ , длина кодируемой последовательности  $k$ , минимальное расстояние кода  $d$ , вероятность ошибки на бит  $p_0$ .

Пусть необходимо найти точное значение вероятности ошибки декодирования  $P_e$ . Для решения этой задачи рассмотрим каким образом с помощью кодера множество сообщений может быть отображено на множество кодовых слов.



Множество  $A$  — множество кодовых слов. Мощность множества кодовых слов  $|A| = 2^k$ . Множество  $B$  — множество кодовых слов. Мощность множества кодовых слов  $|B| = 2^n$ .

Обозначим через  $A_i$  число кодовых слов веса  $i$ , где  $i$  — индекс от 0 до  $n$ . Тогда:

$$A_0 = 1$$

$$A_1 = 0$$

...

$$A_{d-1} = 0$$

$$A_d \neq 0$$

$$A_{d+1} = ?$$

....

$$A_n = ?$$

Как видно, количество кодовых слов веса меньше  $d$ , кроме 0-вектора, будет равно 0, так как  $d$  — это минимальное расстояние. Количество кодовых слов с весом  $d$  гарантировано не равно 0, так как существует как минимум одно слово весом  $d$ . Количество кодовых слов с весом больше  $d$  неизвестно.

В таком случае ошибка декодирования происходит в том случае, когда ненулевой вектор ошибки принадлежит множеству кодовых слов, то есть  $P_e = \Pr\{\bar{e} \in A, \bar{e} \neq 0\}$ . Такому условию удовлетворяют кодовые слова, у которых  $A_i \neq 0$  или  $A_i = ?$ , то есть

$A_d \dots A_n$ . Тогда можно записать выражение для вычисления значения вероятности ошибки декодирования:

$$P_e = \sum_{i=d}^n A_i p_0^i (1 - p_0)^{n-i}$$

#### 4.3 Зависимость вероятности ошибки в канале от ошибки на бит

Для возникновения ошибки в канале  $P$  необходимо, чтобы возникла ошибка с весом от 1 до  $t$  и не произошло ошибки декодирования, то есть

$$P = \left( \sum_{i=1}^n C_n^i p_0^i (1 - p_0)^{n-i} \right) (1 - P_e)$$

Можно подставить ранее подсчитанное значение вероятности ошибки декодирования  $P_e$ , тогда:

$$P = \left( \sum_{i=1}^n C_n^i p_0^i (1 - p_0)^{n-i} \right) \left( 1 - \left( \sum_{i=d}^n A_i p_0^i (1 - p_0)^{n-i} \right) \right)$$

$$C_n^i = \frac{n!}{i! (n - i)!}$$

Будет рассмотрен первый множитель. Пусть  $a = p_0$ ,  $b = (1 - p_0)$ , тогда в результате замены будет получено:

$$\left( \sum_{i=1}^n C_n^i a^i b^{n-1} \right) = \left( \sum_{i=0}^n C_n^i a^i b^{n-i} \right) - C_n^0 a^0 b^n = \left( \sum_{i=0}^n C_n^i a^i b^{n-i} \right) - b^n$$

Согласно формуле бинома Ньютона,

$$\left( \sum_{i=1}^n C_n^i a^i b^{n-1} \right) = (a + b)^n - b^n$$

$$\left( \sum_{i=1}^n C_n^i a^i b^{n-1} \right) = (p_0 + 1 - p_0)^n - (1 - p_0)^n = 1 - (1 - p_0)^n$$

Тогда, подставив значение множителя, будет получено:



$$P = (1 - (1 - p_0)^n) (1 - \sum_{i=d}^n A_i p^i (1 - p)^{n-i})$$

#### 4.4 Вычисление зависимости коэффициента использования канала от ошибки на бит

Объединяя вышеописанные рассуждения, можно получить:

$$P = (1 - (1 - p_0)^n) (1 - \sum_{i=d}^n A_i p^i (1 - p)^{n-i})$$

$$\eta = \frac{1 - ((1 - (1 - p_0)^n) (1 - \sum_{i=d}^n A_i p^i (1 - p)^{n-i}))}{1 + \tau((1 - (1 - p_0)^n) (1 - \sum_{i=d}^n A_i p^i (1 - p)^{n-i}))}$$

Как видно есть зависимость от множества кодовых слов и заранее определить вероятность ошибки и коэффициент использования канала невозможно. Рассмотрим вычисление значения коэффициента использования канала с заданными параметрами.

#### 5 Сравнение практических и теоретических значений коэффициента использования канала для заданных параметров

Пусть  $g(x) = x^3 + x + 1$  или  $\bar{g} = 1011$ , тогда  $r = 3$ . Пусть  $k = 3$ . Тогда кодовые слова имеют длину  $n = r + k = 6$ . С помощью моделирующей программы было получено множество кодовых слов:

$$\overline{m_0} = 000 \Rightarrow \overline{a_0} = 000000 \Rightarrow w(a_0) = 0$$

$$\overline{m_1} = 001 \Rightarrow \overline{a_1} = 001011 \Rightarrow w(a_0) = 3$$

$$\overline{m_2} = 010 \Rightarrow \overline{a_2} = 010110 \Rightarrow w(a_0) = 3$$

$$\overline{m_3} = 011 \Rightarrow \overline{a_3} = 011101 \Rightarrow w(a_0) = 4$$

$$\overline{m_4} = 100 \Rightarrow \overline{a_4} = 100111 \Rightarrow w(a_0) = 4$$

$$\overline{m_5} = 101 \Rightarrow \overline{a_5} = 101100 \Rightarrow w(a_0) = 3$$

$$\overline{m_6} = 110 \Rightarrow \overline{a_6} = 110001 \Rightarrow w(a_0) = 3$$

$$\overline{m_7} = 111 \Rightarrow \overline{a_7} = 111010 \Rightarrow w(a_0) = 4$$

Тогда  $A_3 = 4$ ,  $A_4 = 3$ .

Таким образом:

$$P = (1 - (1 - p_0)^n) \left(1 - \sum_{i=d}^n A_i p^i (1 - p)^{n-i}\right)$$

$$P = (1 - (1 - p_0)^6) (1 - (4 p_0^3 (1 - p_0)^3 + 3 p_0^4 (1 - p_0)^2))$$

Новое теоретическое значение коэффициента использования канала при  $\tau = 2$ :

$$\eta = \frac{1 - P}{1 + P\tau} = \frac{1 - ((1 - (1 - p_0)^6) (1 - (4 p^3 (1 - p)^3 + 3 p^4 (1 - p)^2)))}{1 + 2((1 - (1 - p_0)^6) (1 - (4 p^3 (1 - p)^3 + 3 p^4 (1 - p)^2)))}$$

$p_0$	$P$	$\eta_{\text{теор}}$	$\eta_{\text{эксп}}$
0	0,000	1,000	1
0,1	0,467	0,276	0,271
0,2	0,724	0,113	0,112
0,3	0,839	0,060	0,061
0,4	0,874	0,046	0,045
0,5	0,877	0,045	0,46
0,6	0,879	0,044	0,043
0,7	0,897	0,037	0,034
0,8	0,934	0,023	0,023
0,9	0,977	0,008	0,0076
1	1,000	0,000	0

```

Type g as a binary vector: 1011
Type k: 3
n(0.0,2) = 1.0
n(0.1,2) = 0.27122321670735017
n(0.2,2) = 0.11232168931820735
n(0.30000000000000004,2) = 0.06105006105006105
n(0.4,2) = 0.04519161243673174
n(0.5,2) = 0.04644897580008361
n(0.6,2) = 0.04322268326417704
n(0.7,2) = 0.034908887802834604
n(0.7999999999999999,2) = 0.023300246982618015
n(0.8999999999999999,2) = 0.007619512046448546

```

Рисунок 4 - Значение коэффициента использования канала при  $g(x) = x^3 + x + 1$  и  $k = 3$

Как видно из рисунка, значение при  $p_0 = 1$  получить не удалось. Это объясняется тем, что программа вошла в бесконечный цикл, который возникает из-за того, что ни одно сообщение не будет удачно передано и канал программа будет бесконечно пытаться передать сообщения.

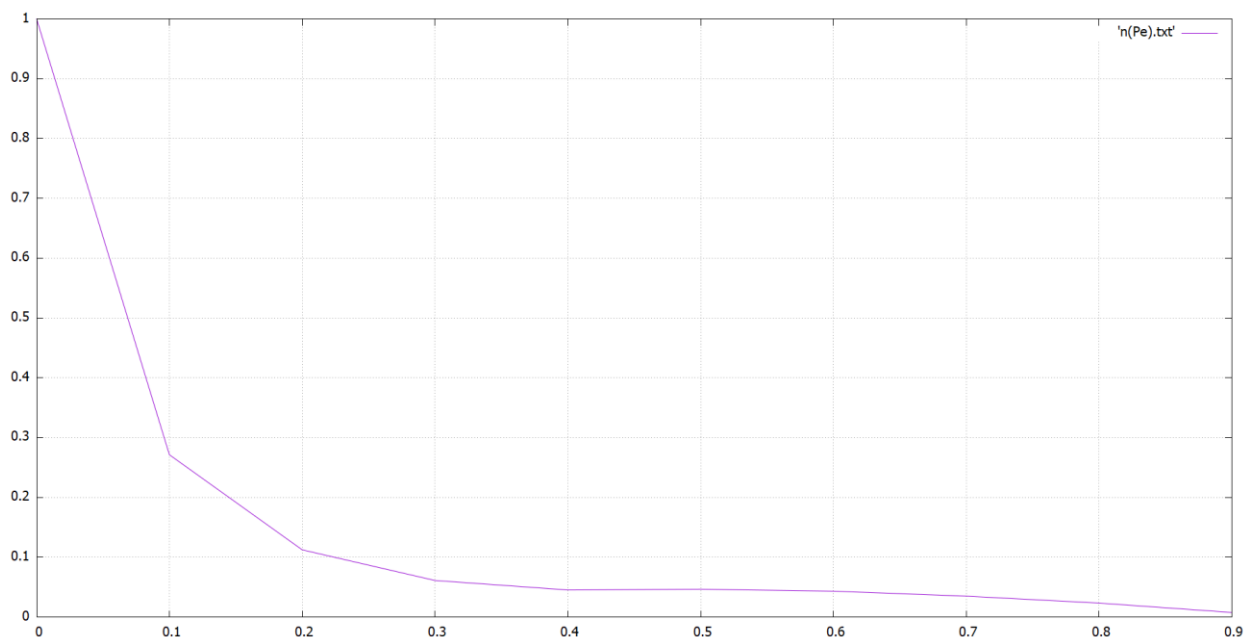


Рисунок 5 - Зависимость коэффициента использования канала от ошибки на бит при  $g(x) = x^3 + x + 1$  и  $k = 3$

Пусть  $g(x) = x^3 + x + 1$  или  $\bar{g} = 1011$ , тогда  $r = 3$ . Пусть  $k = 4$ . Тогда кодовые слова имеют длину  $n = r + k = 7$ . С помощью моделирующей программы было получено множество кодовых слов:

$$\overline{m}_0 = 0000 \Rightarrow \overline{a}_0 = 0000000 \Rightarrow w(a_0) = 0$$

$$\overline{m}_1 = 0001 \Rightarrow \overline{a}_1 = 0001011 \Rightarrow w(a_0) = 3$$

$$\overline{m}_2 = 0010 \Rightarrow \overline{a}_2 = 0010110 \Rightarrow w(a_0) = 3$$

$$\overline{m}_3 = 0011 \Rightarrow \overline{a}_3 = 0011101 \Rightarrow w(a_0) = 4$$

$$\overline{m}_4 = 0100 \Rightarrow \overline{a}_4 = 0100111 \Rightarrow w(a_0) = 4$$

$$\overline{m}_5 = 0101 \Rightarrow \overline{a}_5 = 0101100 \Rightarrow w(a_0) = 3$$

$$\overline{m}_6 = 0110 \Rightarrow \overline{a}_6 = 1110100 \Rightarrow w(a_0) = 4$$

$$\overline{m}_7 = 0111 \Rightarrow \overline{a}_7 = 0111010 \Rightarrow w(a_0) = 4$$

$$\overline{m}_8 = 1000 \Rightarrow \overline{a}_8 = 1000101 \Rightarrow w(a_0) = 3$$

$$\overline{m}_9 = 1001 \Rightarrow \overline{a}_9 = 1001110 \Rightarrow w(a_0) = 4$$

$$\overline{m}_{10} = 1010 \Rightarrow \overline{a}_{10} = 1010011 \Rightarrow w(a_0) = 4$$

$$\overline{m}_{11} = 1011 \Rightarrow \overline{a}_{11} = 1011000 \Rightarrow w(a_0) = 3$$

$$\overline{m}_{12} = 1100 \Rightarrow \overline{a}_{12} = 1100010 \Rightarrow w(a_0) = 3$$

$$\overline{m}_{13} = 1101 \Rightarrow \overline{a}_{13} = 1101001 \Rightarrow w(a_0) = 4$$

$$\overline{m}_{14} = 1110 \Rightarrow \overline{a}_{14} = 1110100 \Rightarrow w(a_0) = 4$$

$$\overline{m}_{15} = 1111 \Rightarrow \overline{a}_{15} = 1111111 \Rightarrow w(a_0) = 7$$

Тогда  $A_3 = 6, A_4 = 8, A_7 = 1$ .

Таким образом:

$$P = (1 - (1 - p_0)^n) \left(1 - \sum_{i=d}^n A_i p^i (1 - p)^{n-i}\right)$$

$$P = (1 - (1 - p_0)^7) (1 - (6 p_0^3(1 - p_0)^4 + 8 p_0^4(1 - p_0)^3 + p_0^7))$$

Новое теоретическое значение коэффициента использования канала при  $\tau = 2$ :

$$\eta = \frac{1 - P}{1 + P\tau} = \frac{1 - ((1 - (1 - p_0)^7) (1 - (6 p^3(1 - p)^4 + 8 p^4(1 - p)^3 + p^7)))}{1 + 2((1 - (1 - p_0)^7) (1 - (6 p^3(1 - p)^4 + 8 p^4(1 - p)^3 + p^7)))}$$

$p_0$	$P$	$\eta_{\text{теор}}$	$\eta_{\text{эксп}}$
0	0,000	1,000	1
0,1	0,519	0,236	0,236
0,2	0,770	0,091	0,095
0,3	0,861	0,051	0,052
0,4	0,879	0,044	0,046
0,5	0,876	0,045	0,045
0,6	0,871	0,047	0,047
0,7	0,849	0,056	0,056
0,8	0,759	0,096	0,093
0,9	0,516	0,238	0,245
1	0,000	1,000	1

```

Type g as a binary vector: 1011
Type k: 4
n(0.0,2) = 1.0
n(0.1,2) = 0.2363507445048452
n(0.2,2) = 0.09520182787509521
n(0.30000000000000004,2) = 0.05278437582475587
n(0.4,2) = 0.04693293283897311
n(0.5,2) = 0.04545247943275306
n(0.6,2) = 0.047442831388177245
n(0.7,2) = 0.056274620146314014
n(0.7999999999999999,2) = 0.09366804046459348
n(0.8999999999999999,2) = 0.2455795677799607
n(0.9999999999999999,2) = 1.0

```

Рисунок 6 - Значение коэффициента использования канала при  $g(x) = x^3 + x + 1$  и  $k = 4$

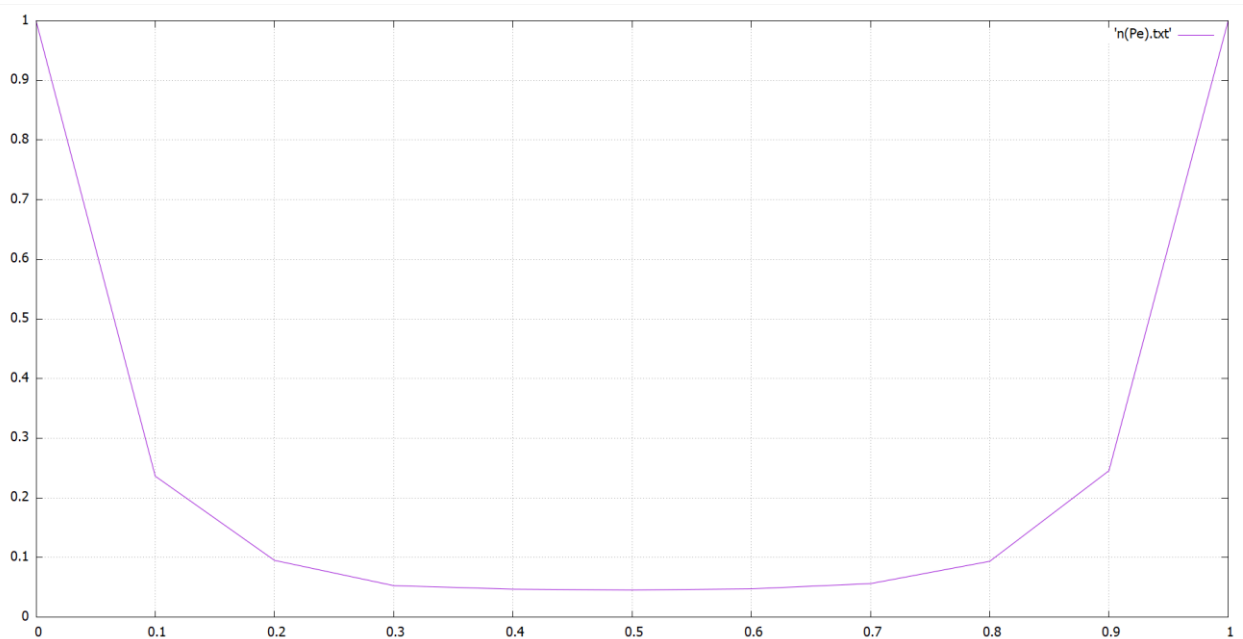


Рисунок 7 - Зависимость коэффициента использования канала от ошибки на бит при  $g(x) = x^3 + x + 1$  и  $k = 4$

## 6 Выводы

В ходе выполнения курсовой работы была смоделирована и проанализирована работа системы передачи данных с использованием контрольной суммы на основе циклических избыточных кодов.

Была получена зависимость вероятности возникновения ошибки в канале  $P$  от вероятности ошибки на бит  $p_0$ . Было установлено, что ошибка в канале зависит в том числе от ошибки декодирования  $P_e$ .

Была получена зависимость коэффициента использования канала  $\eta$  от вероятности ошибки на бит  $p_0$ . Полученные теоретические значения совпали с экспериментальными, что свидетельствует о корректной работе моделирующей программы.

Было установлено, что при некоторых параметрах, например,  $k = 4$  и  $g(x) = x^3 + x + 1$ , график  $\eta(p_0)$  при ошибке на бит  $p_0 > 0,5$  начинает возрастать, что объясняется возрастанием ошибки декодирования и декодер принимает неверное решение, а значит, канал начинает пропускать ошибку.

***CyclicCodes.java***

```
package com.suai;

import java.util.Scanner;

public class CyclicCodes {

    private int g;
    private int r;
    private int k;
    private int m;
    private int a;
    private int e;
    private int b;
    private double p_e; // вероятность единицы
    private boolean E;

    private class Pair {

        private int first;
        private int second;

        public Pair(int a, int b) {
            first = a;
            second = b;
        }
    }

    public CyclicCodes(String in, int k) {
        Pair pair = binaryStringToInt(in);
        g = pair.first;
        r = pair.second;
        this.k = k;
    }

    public CyclicCodes() {
        try {
            Scanner in = new Scanner(System.in);
            System.out.print("Type g as a binary vector: ");
            if (in.hasNextInt()) {
                Pair pair = binaryStringToInt(in.nextLine());
                g = pair.first;
                r = pair.second;
            } else {
                throw new Exception("Incorrect g");
            }
            System.out.println("r = " + (r - 1));
            print("g", g, r);
        }
    }
}
```



```

        System.out.print("Type k: ");
        in = new Scanner(System.in);
        if (in.hasNextInt()) {
            k = in.nextInt();
            if (k <= 0) {
                throw new Exception("Incorrect k");
            }
        }
    }
    catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

public double getP_e() {
    return p_e;
}

public void setP_e(double p) {
    p_e = p;
}

public int getM() {
    return m;
}

public void setM(int newM) {
    m = newM;
}

public void generateMessage() {
    m = 0;
    for (int i = 0; i < k; i++) {
        double tmp = Math.random();
        if (tmp <= 0.5) {
            m += Math.pow(2, i);
        }
    }
    //print("m", m, k);
}

private void print(String v, int val, int size) {
    System.out.print(v + "(x) = ");
    binaryVectorToPolynomial(val, size);
    System.out.println(v + " = " + Integer.toBinaryString(val));
    System.out.println();
}

```

```

private Pair binaryStringToInt(String str) {
    int size = str.length();
    int result = 0;
    for (int i = 0; i < str.length(); i++) {
        char h = str.charAt(i);
        if ((str.charAt(i) - '0') == 0 && result == 0) { // убираем 0
слева
            size--;
        } else {
            result += Math.pow(2, str.length() - 1 - i) * (str.charAt(i) -
'0');
        }
    }
    return new Pair(result, size);
}

private void binaryVectorToPolynomial(int num, int size) { // size -
фактический размер
    if (num - Math.pow(2, size - 1) >= 0) {
        System.out.print("x^" + (size - 1));
        num -= Math.pow(2, size - 1);
    }
    for (int i = size - 2; i > 0; i--) {
        if (num - Math.pow(2, i) >= 0) {
            System.out.print(" + x^" + i);
            num -= Math.pow(2, i);
        }
    }
    if (num == 1) {
        System.out.print(" + 1");
    }
    System.out.println();
}

private int getSize(int num, int upperBound) {
    int i = upperBound;
    for (; i >= 0; i--) {
        if (num - Math.pow(2, i) >= 0) {
            break;
        }
    }
    return (i + 1);
}

private int division(int num, int size) { // num - делимое, size -
фактический размер
    int tmp = g;
    while (size - r >= 0) {
        int pow = size - r;
        tmp = g << pow;
    }
}

```

```

        num = num ^ tmp;
        size = getSize(num, size);
    }
    return num;
}

public void encoding() { //
    int c = m * (int) Math.pow(2, r - 1);
    c = division(c, k + r);
    a = c + m * (int) Math.pow(2, r - 1);
    //print("a", a, getSize(a, k + r));
}

private void generateE() {
    e = 0;
    for (int i = 0; i < (k + r - 1); i++) {
        double tmp = Math.random();
        if (tmp <= p_e) {
            e += Math.pow(2, i);
        }
    }
    //print("e", e, (k + r - 1));
}

public void decoding() throws Exception {
    b = a ^ e;
    // print("b", b, getSize(b, k + r));
    int s = division(b, k + r);
    if (s == 0) {
        //System.out.println("s = 0");
        E = false;
        //System.out.println("E = " + E); // E = 0
    } else {
        //print("s", s, getSize(s, r));
        E = true;
        //System.out.println("E = " + E); // E = 1
    }
}

public boolean messageSendingModeling() {
    try {
        encoding();
        generateE();
        decoding();
        return E;
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

```

```

    }
    return false;
}
}

```

### ***BacktrackingAlgorithmP\_0.java***

```

package com.suai;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class BacktrackingAlgorithmP_0 {

    private int tau = 2;
    private int T;
    private double n;
    private int N = 1000; // 1000
    private ArrayList<Message> messages = new ArrayList();
    private double Pe;
    private String g;
    private int k;

    class Message {

        private int index;
        private boolean isSent;
        private CyclicCodes cyclicCodes;

        public Message(int i) {
            index = i;
            cyclicCodes = new CyclicCodes(g,k);
            cyclicCodes.setP_e(Pe);
            cyclicCodes.generateMessage();
        }

        public boolean isSent() {
            return isSent;
        }
    }

    public BacktrackingAlgorithmP_0() {
        try {
            Scanner in = new Scanner(System.in);
            System.out.print("Type g as a binary vector: ");
            if(in.hasNextInt()) {
                g = in.nextLine();
            }
        }
    }
}

```

```

    }
    else {
        throw new Exception("Incorrect g");
    }

    System.out.print("Type k: ");
    in = new Scanner(System.in);
    if (in.hasNextInt()) {
        k = in.nextInt();
        if (k <= 0) {
            throw new Exception("Incorrect k");
        }
    }
}
catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
}

public void writeToFile(String name, double res) {
    try {
        FileWriter file = new FileWriter(name, true);
        file.write(Pe + " " + res + "\n");
        file.flush();
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

public void clear() {
    try {
        FileWriter file = new FileWriter("n(Pe).txt");
        file.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

public void sending(int index) {
    Message m = new Message(index);
    m.isSent = !(m.cyclicCodes.messageSendingModeling());
    messages.add(m);
    setSameMessage(index, messages.size() - 1);
}

public void markUnsent(int index) { // пометка как неотравленного
    for (int i = 0; i < messages.size(); i++) {

```

```

        if (messages.get(i).index > index) {
            messages.get(i).isSent = false;
        }
    }
}

public void setSameMessage(int index, int messIndex) { // нужный
номер сообщения и индекс в массиве сообщений
    for (int i = 0; i < messages.size(); i++) {
        if (messages.get(i).index == index) {

messages.get(messIndex).cyclicCodes.setM(messages.get(i).cyclicCodes.g
etM());

            break;
        }
    }
}

public void modeling(double p) {
    Pe = p;
    T = 0;
    messages.clear();
    for (int i = 0; i < tau + 1; i++) { // отправка первых tau + 1
сообщений
        sending(i);
        T++;
    }

    int count = N + tau + 1; // порядковый номер сообщения
    int message = tau + 1; // число сообщений на текущий момент
    for (int i = tau + 1; i < count; i++) { // порядковый номер
сообщения
        if (messages.get(i - tau - 1).isSent() == false) { // если
сообщение из tau + 1 сета не отправлено
            sending(messages.get(i - tau - 1).index);
            markUnsent(messages.get(i - tau - 1).index);
            count += tau + 1;
            T++;
        } else {
            if (message < N) {
                sending(message);
                message++;
                T++;
            } else { // сообщения для передачи закончились и канал стоит
                Message m = new Message(-1);
                m.isSent = true;
                messages.add(m);
            }
        }
    }
}

```

```

        n = (double) N / T;
        System.out.println("n(" + p + "," + tau + ") = " + n);
        writeToFile("n(Pe).txt", n);
    }

    public static void main(String[] args) {
        BacktrackingAlgorithmP_0 dop5 = new BacktrackingAlgorithmP_0();
        dop5.clear();
        for (double p = 0; p <= 1; p += 0.1) {
            dop5.modeling(p);
        }
    }
}

```