1. Цель работы

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

По каналу передается сообщение, состоящее из данных и контрольной суммы. Использование контрольной суммы позволяет определить, по принятому сообщению, возникли ли ошибки при передаче данного сообщения по каналу. На рис. 1 изображена структурная схема рассматриваемой в лабораторной работе системы передачи данных.

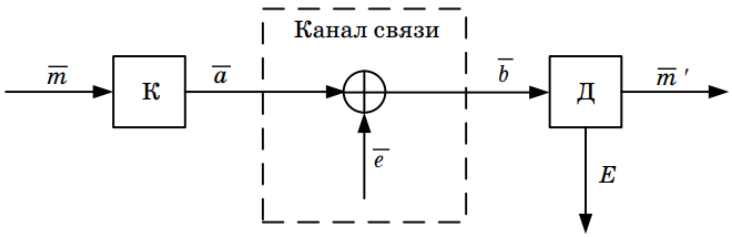


Рис. 1. Структурная схема системы передачи данных:
$\bar{m}$ – информационное сообщение, К – блок кодера,
$\bar{a}$ –закодированное сообщение, $\bar{e}$ –вектор ошибок,
$\bar{b}$ – сообщение на выходе канала, Д – блок декодера,
E –принятое решение, $\bar{m}'$ – сообщение на выходе декодера

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 $\bar{m}$.

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

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

В канале могут произойти ошибки, в результате которых некоторые биты сообщения инвертируются (0 становиться 1 или 1 становиться 0). Вектор ошибок показывает на каких позициях произошла ошибка. Декодер по некоторому алгоритму проверяет контрольную сумму в принятом сообщении и принимает одно из следующих решений:

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

Рассматривается модель двоично-симметричного канала (ДСК) без памяти представленного на рис. 2.

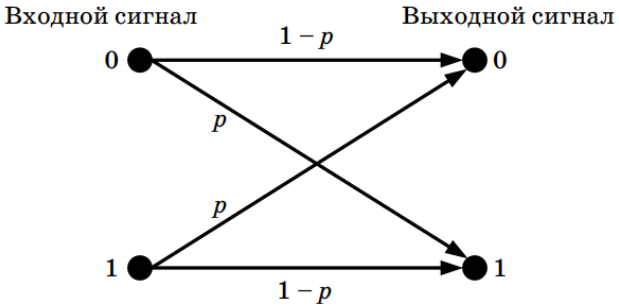


Рис. 2. Модель двоично-симметричного канала

Как видно из рисунка 2 с вероятностью p происходит ошибка. 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.

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).

1) **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 $\{B\}$:

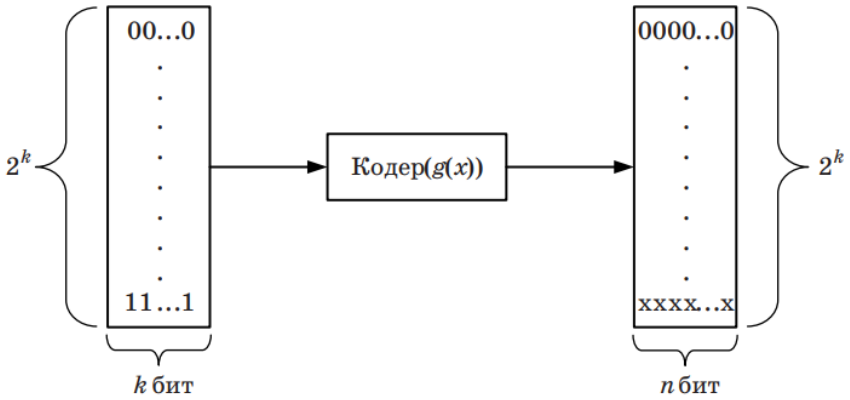
$$\Pr\{B\} = \Pr\{w(\bar{e}) = d \cup w(\bar{e}) = (d+1) \cup \ldots \cup w(\bar{e}) = n\} =$$

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

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

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

2) **Вычисление точного значения ошибки декодирования**:



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:

```
g(x): x^3 + x^1 + 1
messagePolynom: x^3 + x^2 + 1
k = 4
l = 4
```

Множество кодовых слов:

```
p0: 0
p1: x^3 + x^1 + 1
p2: x^4 + x^2 + x^1
p3: x^4 + x^3 + x^2 + 1
p4: x^5 + x^2 + x^1 + 1
p5: x^5 + x^3 + x^2
p6: x^5 + x^4 + 1
p7: x^5 + x^4 + x^3 + x^1
p8: x^6 + x^2 + 1
p9: x^6 + x^3 + x^2 + x^1
p10: x^6 + x^4 + x^1 + 1
p11: x^6 + x^4 + x^3
p12: x^6 + x^5 + x^1
p13: x^6 + x^5 + x^3 + 1
p14: x^6 + x^5 + x^4 + x^2
p15: x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + 1
```

Графики вероятности ошибки:

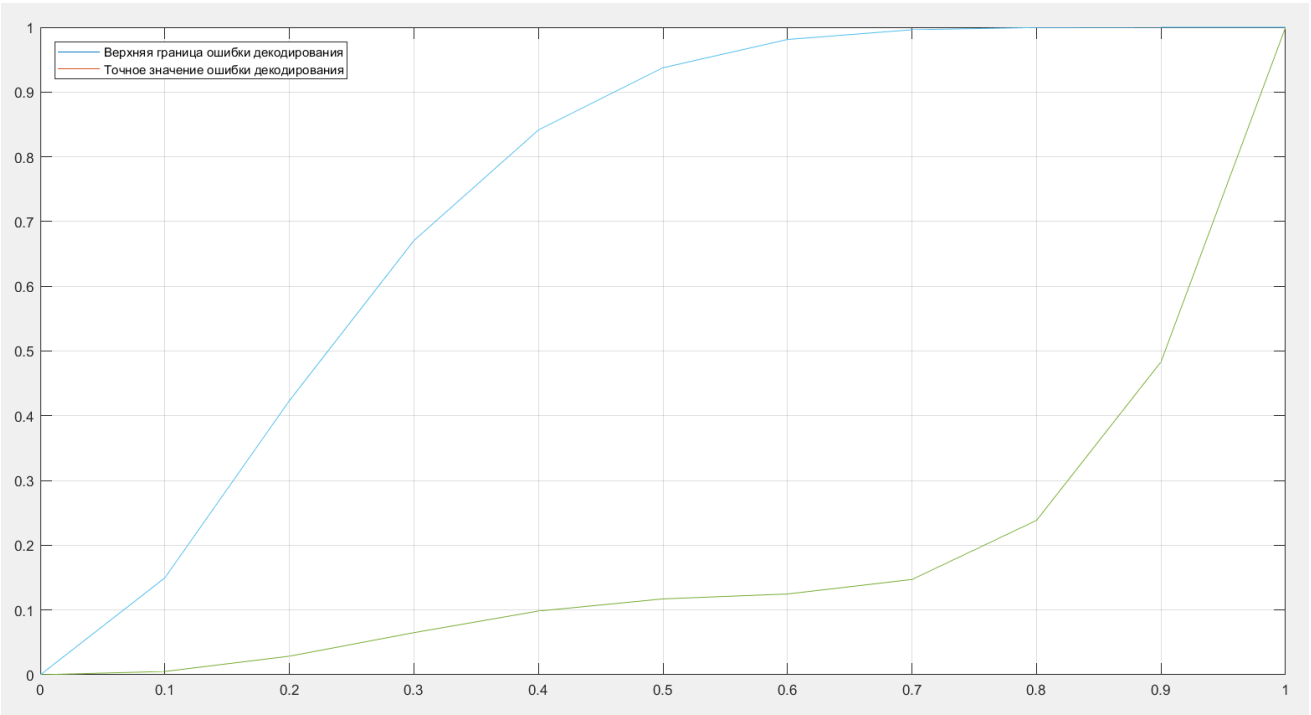


Рисунок 1. Графики вероятности ошибки декодирования.

Исходя из полученного графика можем заметить, что если во множестве кодовых слов присутствует кодовое слово максимальной длины, то есть состоящее из всех единиц, то и верхняя и точная вероятности ошибки декодирования при вероятности ошибки в канале равной единице, также будут равны единице.

Входные данные для работы кодера:

```
g(x): x^3 + x^1 + 1
messagePolynom: x^2 + 1
k = 4
l = 3
```

Множество кодовых слов:

```
p0: 0
p1: x^3 + x^1 + 1
p2: x^4 + x^2 + x^1
p3: x^4 + x^3 + x^2 + 1
p4: x^5 + x^2 + x^1 + 1
p5: x^5 + x^3 + x^2
p6: x^5 + x^4 + 1
p7: x^5 + x^4 + x^3 + x^1
```

Графики вероятности ошибки:

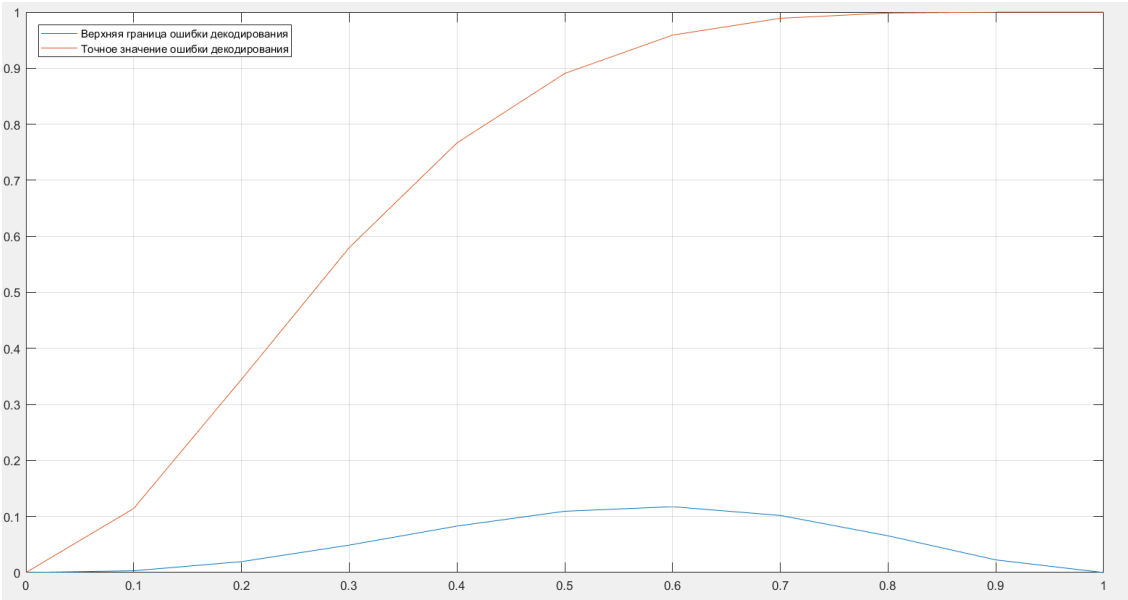


Рисунок 2. Графики вероятности ошибки декодирования.

5. Дополнительное задание:

Построить зависимости верхней оценки вероятности ошибки декодирования $\widehat{Pe}$ и точной вероятности ошибки декодирования Pe при l < k, l > k, l = k. Обосновать полученные зависимости.

Изменение входной последовательности не влияет на оценку вероятности ошибки декодирования. Т.к $s(x) = b(x) mod\ g(x) = (a(x) + e(x))\ mod\ g(x) = a(x) mod\ g(x) + e(x) mod\ g(x)$. $a(x) mod\ g(x) = 0$ по теореме 1. Следовательно, сигнал обнаружения ошибки будет зависеть только от вектора ошибок e. Если он принадлежит множеству кодовых слов, то ошибки не обнаружатся и произойдет ошибка декодирования.

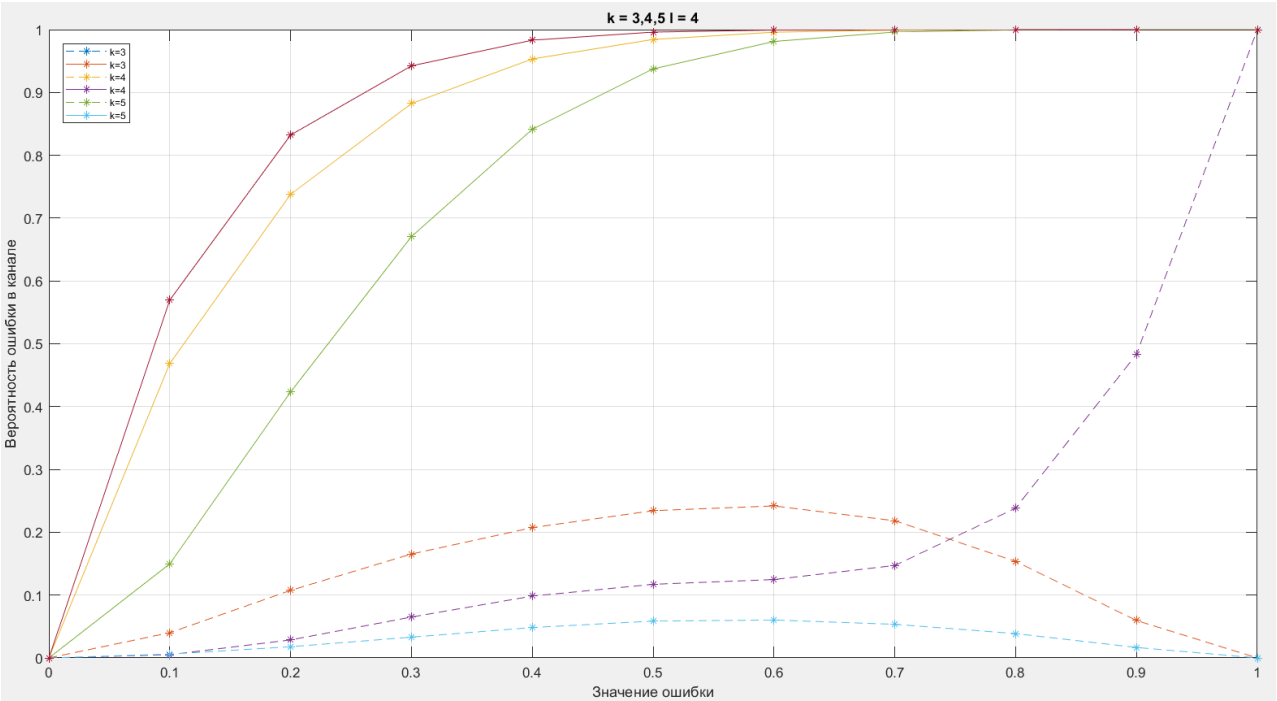


Рисунок 3. Графики вероятности ошибки декодирования.

6. Вывод:

В ходе лабораторной работы была смоделирована система передачи данных. С помощью смоделированной системы произведены исследования работы декодера, в ходе которых получены результаты по зависимости вероятности ошибки декодирования от вероятности ошибки в канале.

Была исследована зависимость вероятности ошибки декодирования от значения вероятности появления ошибки в канале при различных значениях l. Вероятность ошибки декодирования будет зависеть от наличия кодового слова из всех 1. Если оно существует, то вероятность будет стремиться к 1, иначе при приближении p к 1 она будет стремиться к 0. Верхняя граница оценки вероятности ошибки при $k > l$ выше, чем при $k < l$.

7. Листинг программы:

Main.java

```
import java.util.ArrayList;
import java.lang.Math;
public class Main {

    public static void main(String[] args) {
        System.out.println("-------------------------------------
------------------");
        working("11011", "1111",0.1d, 4);
    }


    public static void working(String polynom, String message,
double step, int k) {
        Polynom g = new Polynom(polynom);
        System.out.println("g(x): " + g);
        Polynom messagePolynom = new Polynom(message);
        System.out.println("messagePolynom: " + messagePolynom);
        int l = messagePolynom.getCoefficients().length;
        int r = g.getCoefficients().length - 1;
        int n = l + r;
        System.out.println("k = " + k + "\nl = " + l + "\nr = " +
r + "\nn = " + n);
        Polynom c = new Polynom(messagePolynom.multiply(new
Polynom(r)).mod(g));
        Polynom a = new Polynom(messagePolynom.multiply(new
Polynom(r)).sum(c));
        System.out.println("c(x): " + c);
        System.out.println("a(x): " + a);
        System.out.println("a(x)mod(g(x)) = " + a.mod(g));


        ArrayList<ArrayList<Integer>> codeWords =
Tools.generateCodeWords(l);// ген кодовые слова
        Polynom rPolynom = new Polynom(r);
        ArrayList<ArrayList<Integer>> codeBook = new
ArrayList<>();
        for (int i = 0; i < codeWords.size(); ++i) {
            int[] buffer = new int[codeWords.get(i).size()];
            for (int j = 0; j < buffer.length; ++j) {
                buffer[j] = codeWords.get(i).get(j);
            }
            Polynom tmp = new Polynom(buffer);
            Polynom result =
tmp.multiply(rPolynom).sum(tmp.multiply(rPolynom).mod(g));
            System.out.println("p" + i + ": " + result);
            codeBook.add(Tools.correct(result.getArrayList(), n));
        }//

        int[] weight = Tools.calculateWeight(n, codeBook);
```

```
        for (int i = 0; i < weight.length; ++i) {
            System.out.println("weight " + i + ":  " + weight[i]);
        }

        int d = 10000;
        int t = 1;
        for (int i = 1; i < weight.length; i++) {
            if (weight[i] > 0) {
                t = i;
                d = Math.min(t, d);
            }
        }
        System.out.println("d = " + d);

        Tools.loadToFile("PeUp.txt",Tools.calculatePeUp(step, d,
n) ,step);
        Tools.loadToFile("Pe.txt", Tools.calculatePe(weight, step,
d, n), step);
    }
}
```

Polynom.java

```
import java.util.ArrayList;

public class Polynom {
    private int[] coefficients;

    public Polynom(Polynom other) {
        coefficients = new int[other.getCoefficients().length];
        for (int i = 0; i < coefficients.length; ++i) {
            coefficients[i] = other.getCoefficients()[i];
        }
    }

    public Polynom(int degree) {
        coefficients = new int[degree + 1];
        coefficients[degree] = 1;
    }


    public Polynom(int[] coefficients) {
        this.coefficients = new int[coefficients.length];
        for (int i = 0; i < coefficients.length; ++i) {
            this.coefficients[i] =
coefficients[coefficients.length - i - 1] % 2;
        }
    }

    public Polynom(String coefficients) {
        this.coefficients = new int[coefficients.length()];
        for (int i = 0; i < coefficients.length(); ++i) {
            this.coefficients[i] =
```

```
Integer.parseInt(Character.toString(coefficients.charAt(coefficien
ts.length() - i - 1))) % 2;
        }
    }

    public ArrayList<Integer> getArrayList() {
        ArrayList<Integer> result = new ArrayList<>();
        for (int i = 0; i < coefficients.length; ++i) {
            result.add(coefficients[i]);
        }
        return result;
    }

    public int[] getCoefficients() {
        return coefficients;
    }

    public Polynom sum(Polynom other) {
        int[] result = new int[Math.max(coefficients.length,
other.getCoefficients().length)];
        int[] secondCoefficients = other.getCoefficients();
        for (int i = 0; i < result.length; i++) {
            if (i >= coefficients.length) {
                result[i] = secondCoefficients[i];
            } else if (i >= secondCoefficients.length) {
                result[i] = coefficients[i];
            } else {
                result[i] = (secondCoefficients[i] +
coefficients[i]) % 2;
            }
        }
        return new Polynom(reverse(correct(result)));
    }

    public int[] reverse(int[] array) {
        int[] result = new int[array.length];
        for (int i = 0; i < array.length; ++i) {
            result[i] = array[array.length - i - 1];
        }
        return result;
    }

    public Polynom multiply(Polynom other) {
        int[] result = new int[this.coefficients.length +
other.getCoefficients().length - 1];
        int[] secondCoefficients = other.getCoefficients();
        for (int i = coefficients.length - 1; i >= 0; i--) {
            for (int j = secondCoefficients.length - 1; j >= 0; j-
-) {
                result[i + j] += (coefficients[i] *
secondCoefficients[j]) % 2;
            }
        }
```

```
        return new Polynom(reverse(result));
    }

    public Polynom mod(Polynom other) {
        Polynom tmp = new Polynom(this);

        while (tmp.getCoefficients().length >=
other.getCoefficients().length) {
            int distance = tmp.getCoefficients().length -
other.getCoefficients().length;
            tmp = tmp.sum(other.multiply(new Polynom(distance)));
        }

        return tmp;
    }

    public int[] correct(int[] array) {
        int counter = 0;
        for (int i = 0; i < array.length; ++i) {
            if (array[i] != 0) {
                counter = i;
            }
        }
        int[] result = new int[counter + 1];
        for (int i = 0; i < result.length; ++i) {
            result[i] = array[i];
        }
        return result;
    }

    @Override
    public String toString() {
        if (coefficients.length == 1 && coefficients[0] == 0) {
            return "0";
        }
        StringBuilder stringBuilder = new StringBuilder();
        boolean firstX = false;
        for (int i = coefficients.length - 1; i >= 0; i--) {
            if (coefficients[i] != 0) {
                firstX = true;
                if (i == 0) {
                    stringBuilder.append(coefficients[i]);
                    break;
                }
                stringBuilder.append("x^").append(i);
            }
            if (i - 1 >= 0 && coefficients[i - 1] != 0 && firstX)
{
                stringBuilder.append(" + ");
            }
        }
        return stringBuilder.toString();
```

```
    }
}
```

Tools.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class Tools {
    public static long calculateFactorial(int number) {
        long result = 1;
        for (int i = number; i > 0; i--) {
            result *= i;
        }
        return result;
    }

    public static double calculateCombinations(int m, int n) {
        return (double) calculateFactorial(n) / (double)
(calculateFactorial(m) * calculateFactorial(n - m));
    }

    public static ArrayList<ArrayList<Integer>>
generateCodeWords(int k) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<>();
        ArrayList<Integer> input = new ArrayList<>();
        input.add(0);
        input.add(1);
        generationPlacementRepetitions(k, input, new
ArrayList<>(), result);
        return result;
    }

    private static void generationPlacementRepetitions(int
counter, ArrayList<Integer> inputArray, ArrayList<Integer>
outputArray, ArrayList<ArrayList<Integer>> result) {
        if(counter == 0) {
            result.add(new ArrayList<>(outputArray));
        } else {
            for (int i = 0; i < inputArray.size(); ++i) {
                outputArray.add(inputArray.get(i));
                generationPlacementRepetitions(counter - 1,
inputArray, outputArray, result);
                outputArray.remove(outputArray.size() - 1);
            }

        }
    }

    public static ArrayList<Double> calculatePe(int[] weight,
double step, int d, int n) {
```

```
        ArrayList<Double> result = new ArrayList<>();
        for (double p = 0; p < 1.d; p += step) {
            double buffer = 0;
            for (int i = d; i <= n; ++i) {
                buffer += weight[i] * Math.pow(p, i) * Math.pow(1
- p, n - i);
            }
            result.add(buffer);
        }
        return result;
    }

    public static List<Double> calculatePeUp(double step, int d,
int n){
        ArrayList<Double> result = new ArrayList<>();
        for (double p = 0; p < 1.d; p += step) {
            double buffer = 0;
            for (int i = 0; i < d - 1; ++i) {
                buffer += calculateCombinations(i, n) *
Math.pow(p, i) * Math.pow(1 - p, n - i);
            }
            result.add(1 - buffer);
        }
        return result;
    }

    public static ArrayList<Integer> correct(ArrayList<Integer>
arrayList, int n) {
        for (int i = arrayList.size(); i < n; ++i) {
            arrayList.add(0);
        }
        return arrayList;
    }

    public static int[] calculateWeight(int n,
ArrayList<ArrayList<Integer>> codeBook) {
        int[] result = new int[n + 1];
        for (int i = 0;i < codeBook.size(); ++i) {
            int counter = 0;
            ArrayList<Integer> tmp = codeBook.get(i);
            for (int j = 0; j < tmp.size(); ++j) {
                counter += tmp.get(j);
            }
            result[counter]++;
        }
        return result;
    }
    public static void loadToFile(String filename, List<Double>
array, double step) {
        try {
            FileWriter  fileWriter = new FileWriter(filename);
            for (int i = 0; i < array.size(); ++i) {
                fileWriter.write((i * step) + "   " + array.get(i)
```

```
+ "\n");
                fileWriter.flush();
            }
            fileWriter.close();
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }
}
```