Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа № 4

по дисциплине «Вычислительная математика»

Интерполяция и аппроксимация

Выполнил:

Гурьянов Кирилл Алексеевич

Группа: P32302

Преподаватель:

Перл Ольга Вячеславовна

Санкт-Петербург

2023

1. Описание методов, расчетные формулы. Метод наименьших квадратов.

Метод наименьших квадратов — математический метод, используемый для решения различных задач, связанных с минимизацией суммы квадратов отклонений функций от экспериментальных входных данных.

Метод наименьших квадратов представляет собой процесс нахождения функциональной зависимости, которая наилучшим образом соответствует данным. Для того, чтобы применить метод наименьших квадратов необходимо минимизировать сумму:

$$\sum_{i=1}^{N} (y_i - f(x_i))^2 \to min$$

Пусть в качестве функциональной зависимости многослен:

$$P_m(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_m x^m$$

Условие минимума – нулевые производные по всем переменным c_0, c_1, c_2, ..., c_m

$$-\sum_{i=0}^{n} 2(y - c_0 - c_1 x_i - \ldots - c_m x_i) x_i^k = 0$$

$$c_0 \sum_{i=0}^{n} x_i^k + c_1 \sum_{i=0}^{n} x_i^{k+1} + c_2 \sum_{i=0}^{n} x_i^{k+2} + \ldots + c_m \sum_{i=0}^{n} x_i^{k+m} = \sum_{i=0}^{n} x_i^k y_i$$

где $k = 0, 1, 2, \ldots, m$

Если обозначить:

$a_k = \sum_{i=0}^{n} x_i^k$, $b_k = \sum_{i=0}^{n} x_i^k y_i$, то получим систему:

$$\begin{cases} a_0 c_0 + a_1 c_1 + a_2 c_2 + \ldots + a_m c_m = b_0 \\ a_1 c_0 + a_2 c_1 + a_3 c_2 + \ldots + a_{m+1} c_m = b_1 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_m c_0 + a_{m+1} c_1 + a_{m+2} c_2 + \ldots + a_{2m} c_m = b_m \end{cases}$$

Решая данную систему линейных уравнений получим коэффициенты $c_0, c_1, c_2, \ldots, c_m$, которые являются искомыми параметрами формулы.

Существуют различные виды аппроксимации: линейная, квадратичная, логарифмическая, тригонометрическая и т.п. Общий подход для применения всех видов аппроксимации методом наименьших квадратов схожий.

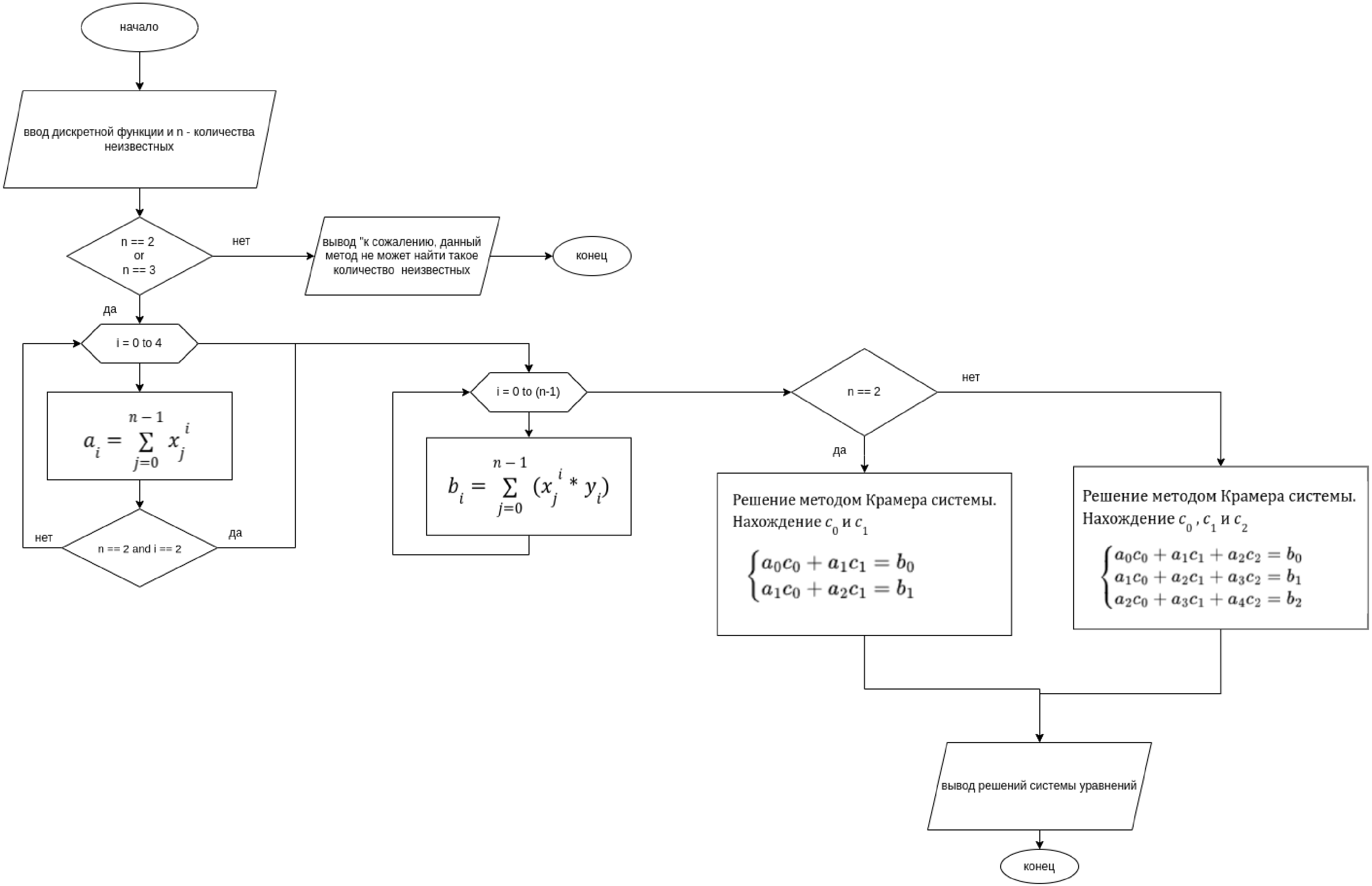
Если в качестве аппроксимирующей функции взять логарифм, то метод наименьших квадратов примет вид:

$$\sum_{i=1}^{N} (y_i - (a * ln\ x_i + b))^2 \rightarrow min$$

Формально в системе нужно заменить x на $\ln x$, или $\sin x$.

Метод наименьших квадратов может быть использован для различных задач, таких как предсказание будущих значений переменной, оценки эффекта независимой переменной на зависимую переменную, а также для создания моделей прогнозирования.

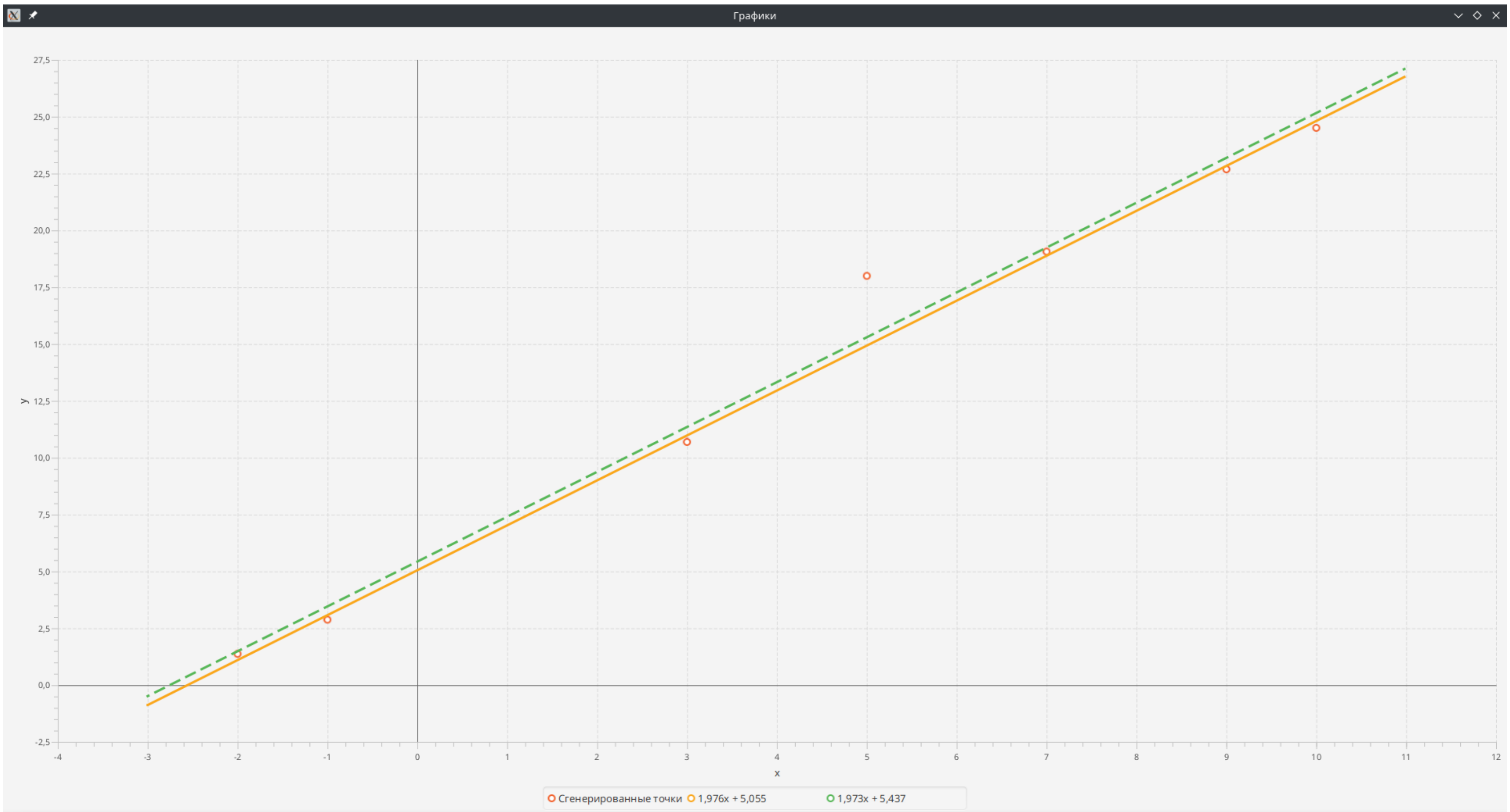
2. Блок-схемы методов

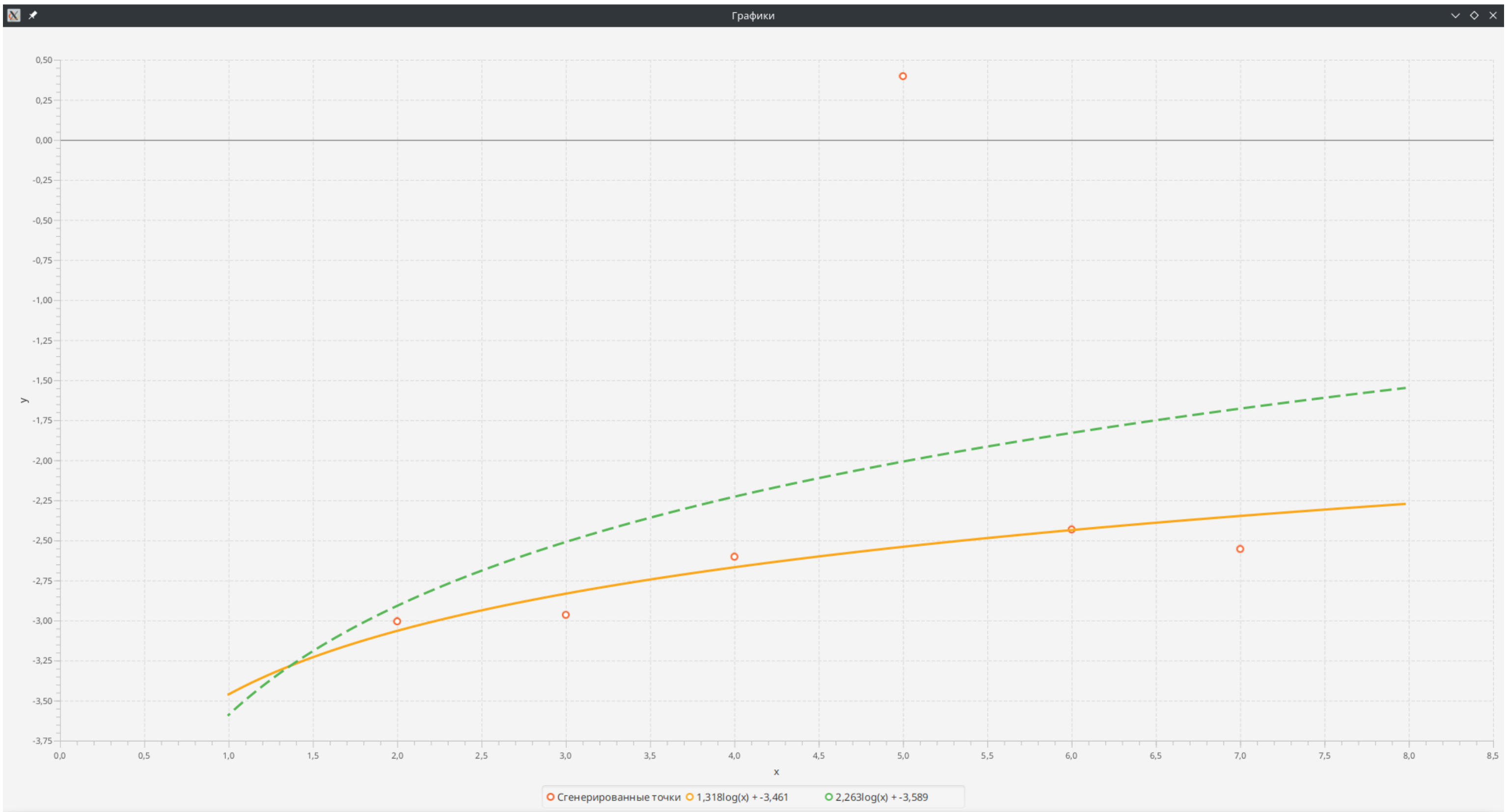

начало
ввод дискретной функции и n - количества неизвестных
n == 2 or n == 3
нет
вывод "к сожалению, данный метод не может найти такое количество неизвестных
конец
да
i = 0 to 4
$a_i = \sum_{j=0}^{n-1} x_j^i$
n == 2 and i == 2
нет
да
i = 0 to (n-1)
$b_i = \sum_{j=0}^{n-1} (x_j^i * y_i)$
n == 2
нет
да
Решение методом Крамера системы. Нахождение c_0 и c_1
$\begin{cases} a_0c_0 + a_1c_1 = b_0 \\ a_1c_0 + a_2c_1 = b_1 \end{cases}$
Решение методом Крамера системы. Нахождение c_0, c_1 и c_2
$\begin{cases} a_0c_0 + a_1c_1 + a_2c_2 = b_0 \\ a_1c_0 + a_2c_1 + a_3c_2 = b_1 \\ a_2c_0 + a_3c_1 + a_4c_2 = b_2 \end{cases}$
вывод решений системы уравнений
конец

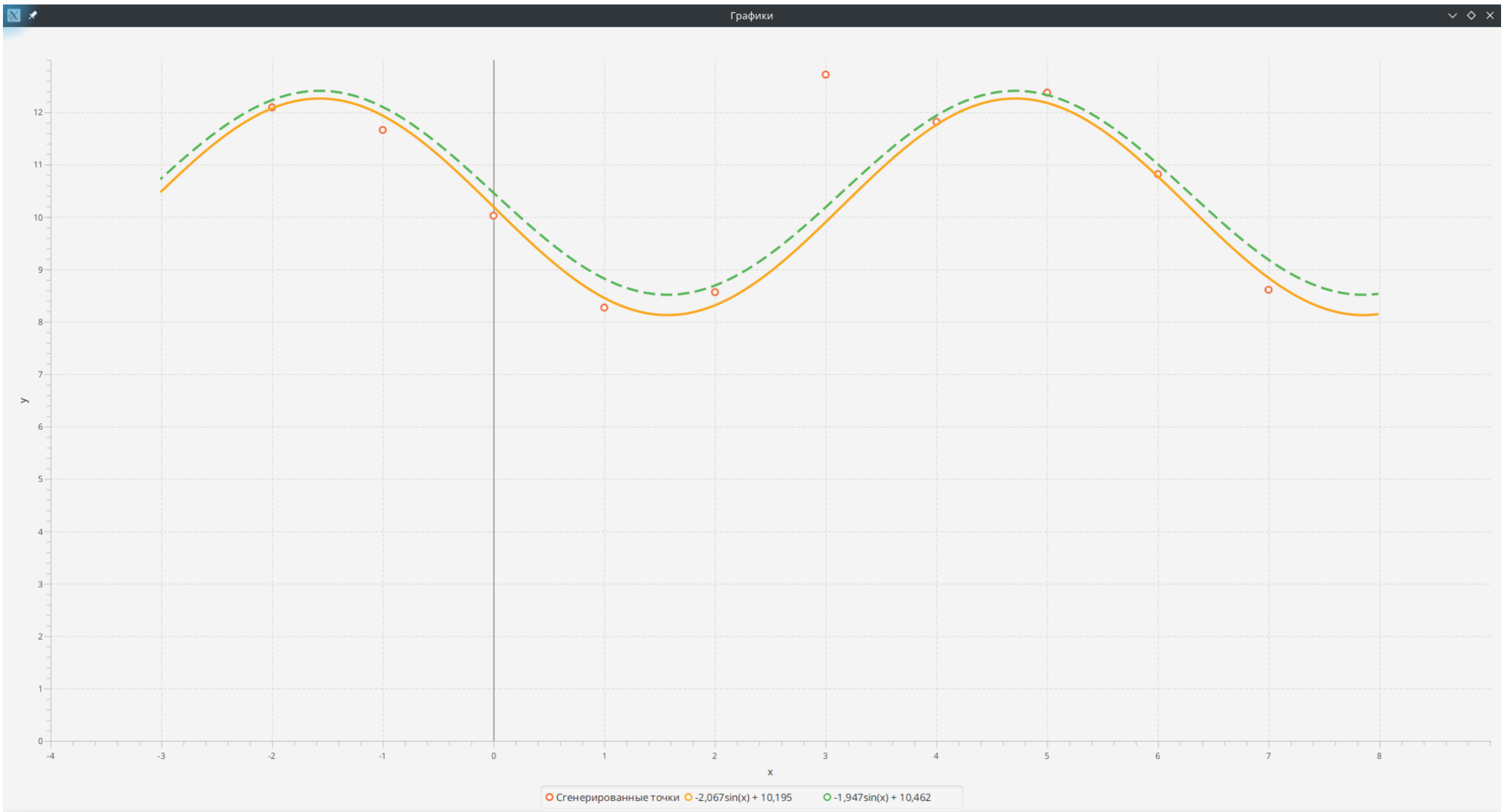
3. Листинг реализованных методов программы

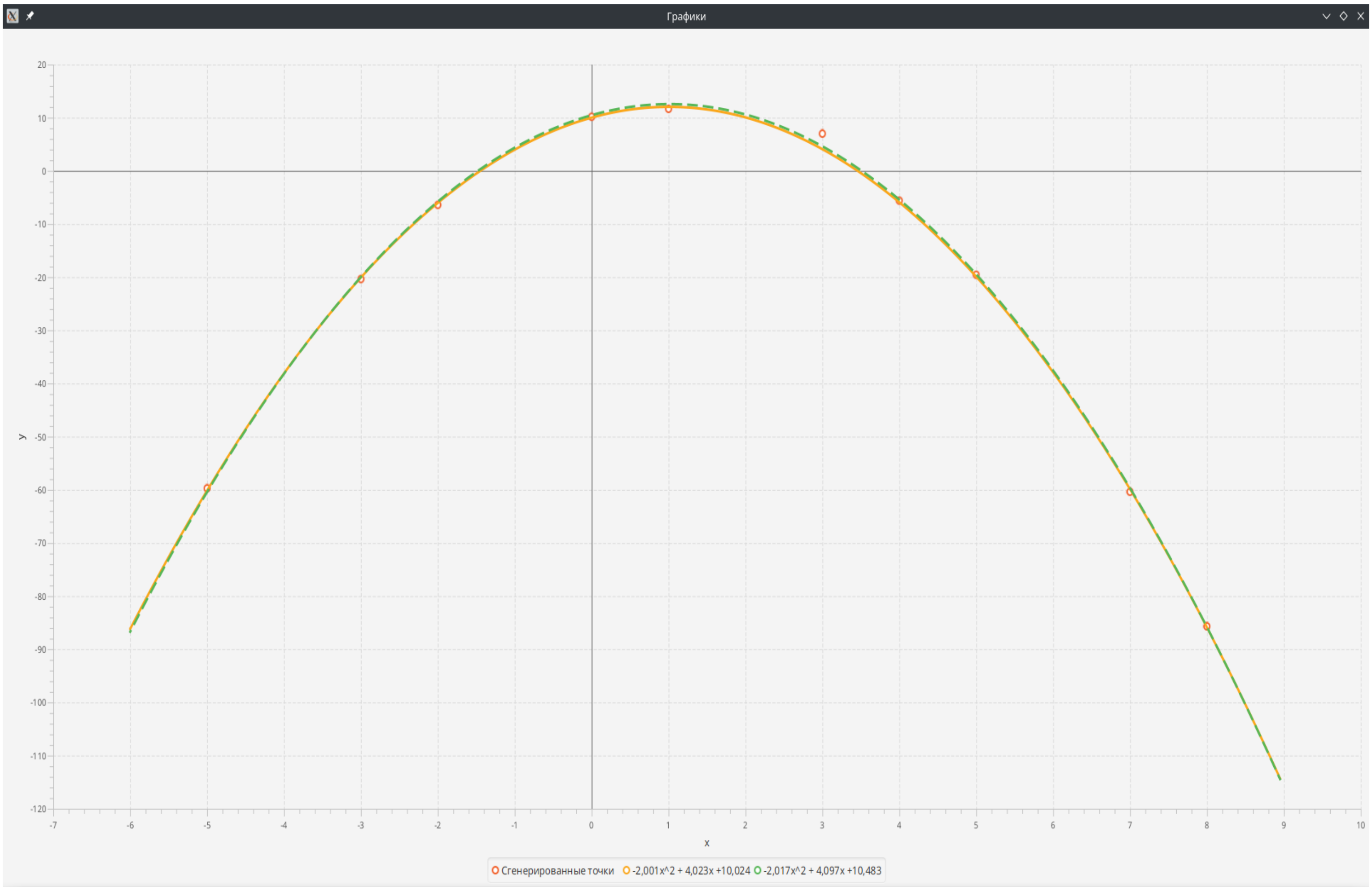
```java
package lab4.algorithms;

import lab4.data.AnswerFromKramerMethod;
import lab4.data.DiscreteFunction;
import lab4.data.ApproximationTypeFunc;
import lab4.data.Matrix;
import lab4.exceptions.WrongNumberOfCoefficients;


import java.util.ArrayList;

3 usages
public class LeastSquaresMethod {

    3 usages
    private final ApproximationTypeFunc approximationTypeFunc;
    1 usage
    public LeastSquaresMethod(ApproximationTypeFunc function) { approximationTypeFunc = function; }
    2 usages
    public double[] findCoefficients(DiscreteFunction function) throws WrongNumberOfCoefficients{
        int numberOfCoefficients = function.getNumberOfCoefficients();
        if (numberOfCoefficients < 2 || numberOfCoefficients > 3)
            throw new WrongNumberOfCoefficients("К сожалению, пока что данный метод не умеет решать уравнения с таким количеством неизвестных");
        Double[] sumsOfX = new Double[5];
        Double[] sumsOfFreeTerms = new Double[numberOfCoefficients];
        Matrix matrix = new Matrix(numberOfCoefficients);
        calculateSumOfX(function, sumsOfX);
        calculateSumOfFreeTerms(function, sumsOfFreeTerms);

        if (numberOfCoefficients == 2) {
            matrix.setRow( i: 0, new ArrayList<>(){{add(sumsOfX[0]); add(sumsOfX[1]); add(sumsOfFreeTerms[0]);}});
            matrix.setRow( i: 1, new ArrayList<>(){{add(sumsOfX[1]); add(sumsOfX[2]); add(sumsOfFreeTerms[1]);}});
        } else if (numberOfCoefficients == 3) {
            matrix.setRow( i: 0, new ArrayList<>(){{add(sumsOfX[0]); add(sumsOfX[1]); add(sumsOfX[2]); add(sumsOfFreeTerms[0]);}});
            matrix.setRow( i: 1, new ArrayList<>(){{add(sumsOfX[1]); add(sumsOfX[2]); add(sumsOfX[3]); add(sumsOfFreeTerms[1]);}});
            matrix.setRow( i: 0, new ArrayList<>(){{add(sumsOfX[2]); add(sumsOfX[3]); add(sumsOfX[4]); add(sumsOfFreeTerms[2]);}});
        }

        KramerMethod kramerMethod = new KramerMethod();
        AnswerFromKramerMethod answer = kramerMethod.solveTheSystem(matrix);
        return answer.getApproximations();
    }
    1 usage
    private void calculateSumOfX(DiscreteFunction function, Double[] sumsOfX) {
        int numberOfCoefficients = function.getNumberOfCoefficients();
            for (int i = 0; i < 3; i++) {
                sumsOfX[i] = 0.0;
                for (int j = 0; j < function.getNumberOfPoints(); j++) {
                    sumsOfX[i] += Math.pow(approximationTypeFunc.calculate(function.getXArray()[j]), i);
                }
                if (numberOfCoefficients == 2 && i == 2) break;
            }
    }
    1 usage
    private void calculateSumOfFreeTerms(DiscreteFunction function, Double[] sumsOfFreeTerms) {
        int numberOfCoefficients = function.getNumberOfCoefficients();
        for (int i = 0; i < numberOfCoefficients; i++) {
            sumsOfFreeTerms[i] = 0.0;
            for (int j = 0; j < function.getNumberOfPoints(); j++) {
                sumsOfFreeTerms[i] += Math.pow(approximationTypeFunc.calculate(function.getXArray()[j]), i) * function.getYArray()[j];
            }
        }
    }
}
```

4. Примеры и результаты работы программы на разных данных


Графики
y
x
Сгенерированные точки
1,976x + 5,055
1,973x + 5,437


Графики
y
x
Сгенерированные точки
1,318log(x) + -3,461
2,263log(x) + -3,589


Графики
y
x
Сгенерированные точки
-2,067sin(x) + 10,195
-1,947sin(x) + 10,462


Графики
y
x
Сгенерированные точки
-2,001x^2 + 4,023x +10,024
-2,017x^2 + 4,097x +10,483

5. Вывод

Аппроксимация — это процесс нахождения функции, которая наилучшим образом приближает реальную функцию, основываясь на некотором наборе данных или ограниченном количестве информации. Это может быть полезным в различных областях, таких как математика, физика, инженерия, экономика и т.д. Например, если у нас есть некоторые экспериментальные данные, мы можем использовать аппроксимацию, чтобы найти математическую функцию, которая наилучшим образом описывает эти данные. Аппроксимация может быть выполнена с использованием различных методов. В данной лабораторной работе был рассмотрен метод наименьших квадратов.

Интерполяция — это метод нахождения значений функции между известными точками. Она используется для построения новых данных на основе имеющихся значений. Это может быть полезно, когда мы имеем набор данных, но хотим узнать значения в промежуточных точках.

Простой пример интерполяции - это нахождение значения функции на промежутке между двумя известными точками. Если у нас есть значения функции $f(x)$ для x_1 и x_2, мы можем использовать интерполяцию, чтобы найти значение функции для некоторого x_n между x_1 и x_2.

Стоит отметить, что интерполяционная функция всегда проходит через заданные точки, в то время как аппроксимирующая проходит между ними.

Существует большое количество методов интерполяции. Рассмотрим для начала метод кубических сплайнов — это метод локальной интерполяции, т.е. на каждом отдельном интервале мы строим отдельные уравнения сплайнов. В общем виде уравнение кубического сплайна выглядит так:

$$S = a + b(x - x_0) + c(x - x_0)^2 + d(x - x_0)^3$$

При применении метода кубических сплайнов необходимо, чтобы выполнялся ряд условий, чтобы была достигнута плавность и гладкость картинки:

- Сплайны должны проходить через узловые точки.
- В стыках сплайнов должна обеспечиваться гладкость, т.е. не должно быть изломов и изменения кривизны. Гладкость обеспечивается за счет первых и вторых производных. То, под каким углом заходит сплайн в заданную точку определяется его первой производной. Кривизна сплайна определяется его второй производной. В каждой точке, за исключением крайних, должны быть равны первые и вторые производные соседних сплайнов.
- На конце интервалов наиболее просто и естественно задать условие, чтобы вторая производная сплайнов была равна 0. Таким образом крайние сплайны не будут иметь ни выпуклости, ни вогнутости.

Метод интерполяции кубическими сплайнами также имеет несколько недостатков:

- Сложность реализации. Метод кубических сплайнов требует решения системы линейных уравнений, что может быть трудоемкой операцией, особенно при большом количестве узлов интерполяции.
- Неустойчивость при наличии больших погрешностей в данных. Если значения функции в некоторых узлах интерполяции содержат большие ошибки, то метод кубических сплайнов может давать неверные результаты.

- Сложность использования результатов в различных задачах. В отличие от многочлена Лагранжа или Ньютона, метод сплайнов дает отдельные многочлены на каждом участке, что может усложнять их использование в дополнительных задачах, таких как дифференцирование и интегрирование.

Следующий метод интерполяции — интерполяция полиномом Ньютона. Этот метод позволяет найти многочлен степени $n - 1$, проходящий через n точек. Этот метод основан на том, что любой многочлен степени $n - 1$ однозначно определяется n точками, через которые он проходит.

В общем виде уравнение многочлена Ньютона выглядит так:

$$N = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + ... + a_n(x - x_0)(x - x_1) * ... * * (x - x_{n-1})$$

В отличие от метода сплайнов, метод интерполяции полиномом Ньютона дает нам одно уравнение на всем промежутке. Таким образом метод Ньютона — это метод глобальной интерполяции. Это довольно удобно, т.к. нам не нужно искать отдельно уравнения на каждом интервале.

Метод интерполяции полиномом Лагранжа — это метод, который используется для интерполяции функции полиномом более низкой степени. Этот метод основан на интерполяции многочленом Лагранжа, который представляет собой полином степени $n - 1$, где n — это количество узлов интерполяции. Данный метод более удобен, если меняются значения в точках, т.к. его не требуется пересчитывать. Однако при изменении количества точек полином полностью приходится пересчитывать, в отличие от метода Ньютона.

Полином Лагранжа представляется в виде:

$$L(x) = \sum_{i=0}^{n} y_i l_i(x), \text{ где } l_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$$

Методы интерполяции многочленами Лагранжа и Ньютона имеет несколько недостатков:

- Высокая степень многочлена может привести к тому, что значения интерполяционного многочлена на краях интерполяционного отрезка сильно отличаются от реальных значений функции.
- Метод неустойчивы при наличии сильных погрешностей в данных.
- Методы не подходит для интерполяции функций, которые имеют особенности, такие как разрывы или особые точки.

Таким образом, выбор между методом наименьших квадратов и интерполяцией зависит от целей аппроксимации и характеристик данных. Если требуется аппроксимация функции на всем интервале значений, использование МНК может быть более подходящим, а если требуется точное восстановление функции по ее значениям в конечном наборе точек, то следует использовать интерполяционный подход.