Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики
Кафедра вычислительной математики

Ярмолкевич Даниил Андреевич

Лабораторная работа №2 по предмету

«Методы численного анализа»

Студента 2 курса 2 группы

Преподаватель
Полищук Максим Александрович

Минск, 2018

Оглавление

Задание 1

Постановка задачи:

Задание 1. Построить квадратичную функцию $P_2(x) = c_2x^2 + c_1x + c_0$, которая дает для $f(x)$ наилучшее приближение по методу наименьших квадратов решением системы линейных уравнений $X^T Xc = X^T y$ с нормальной матрицей $X^T X = \left(\sum_{i=0}^{n} x_i^{k+l-2} \right)$ (normal equations). Функцию $f(x)$ взять из задания лабораторной работы 1. Задать значения y_i функции $f(x)$, определенной на интервале $[a, b]$, в узлах $x_i = a + \frac{b-a}{n} i, i = \overline{0,n}, n = 30$. Код решения системы линейных уравнений методом Гаусса с выбором главного элемента по столбцу (алгоритм GEPP) взять из лабораторной работы 1 курса «Вычислительные методы алгебры». Для вычислений использовать тип *float*.

В отчёте представить значения коэффициентов c_0, c_1, c_2, значение $\hat{\sigma}$ несмещенной оценки дисперсии случайных ошибок для выборок малого объема, $\hat{\sigma} = \sqrt{\frac{RSS}{n+1-k}} = \sqrt{\frac{1}{n+1-k} \sum_{i=0}^{n} \left(y_i - P_2(x_i) \right)^2}$, где k — число степеней свободы, k=3. Построить на одном рисунке графики функций $f(x)$ и $P_2(x)$ на интервале $[a, b]$ для визуализации результата работы программы.

Входные данные:

Вариант 7 (24%17): $10x^3(5\pi + x)^{-1/4}, [0;10]$

Использованные формулы

1) Основная система уравнений для коэффициентов полинома:

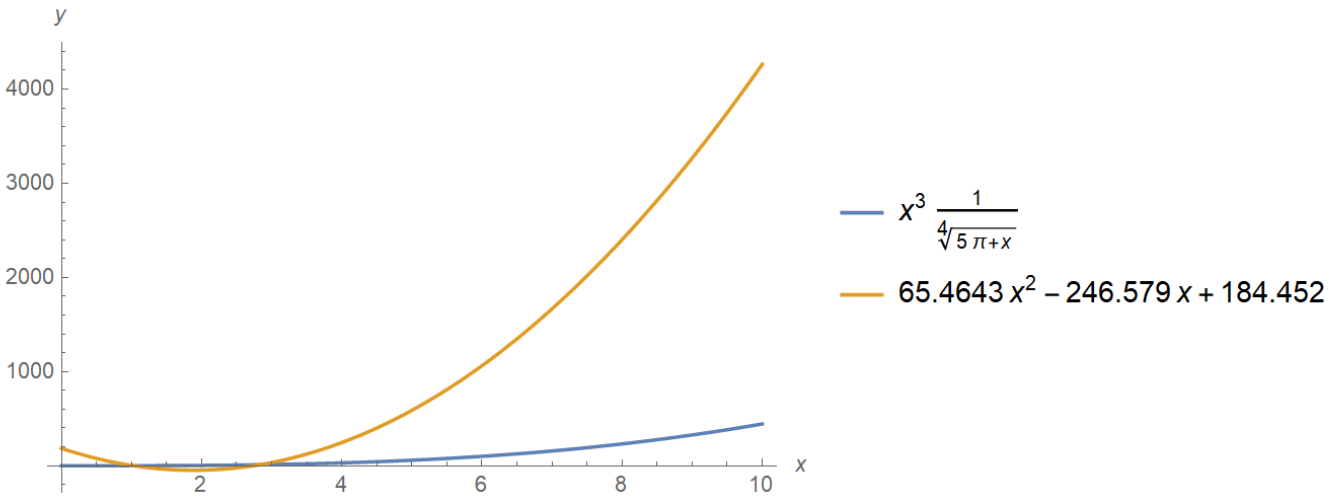
$$\begin{pmatrix} \sum_{i=0}^{30} x_i^0 & \sum_{i=0}^{30} x_i^1 & \sum_{i=0}^{30} x_i^2 \\ \sum_{i=0}^{30} x_i^1 & \sum_{i=0}^{30} x_i^2 & \sum_{i=0}^{30} x_i^3 \\ \sum_{i=0}^{30} x_i^2 & \sum_{i=0}^{30} x_i^3 & \sum_{i=0}^{30} x_i^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{30} y_i \\ \sum_{i=0}^{30} y_i x_i^1 \\ \sum_{i=0}^{30} y_i x_i^2 \end{pmatrix}$$

2) Узлы: $x_i = a + \frac{b-a}{n} i, i = \overline{0,n}, n = 30$

3) Несмещенная оценка дисперсии случайных ошибок для выборок малого объема при k=3 и n=30, $\hat{\sigma} = \sqrt{\frac{RSS}{28}} = \sqrt{\frac{1}{28} \sum_{i=0}^{n} \left(y_i - P_2(x_i) \right)^2}$,

Результат работы программы и анализ полученных результатов

```
Gauss: 184.45169-246.57874x+65.46427x^2
mistake: 86.88848
```



Задание 2

Постановка задачи:

Задание 2. Для условия из задания 1 применить алгоритм QR-разложения матрицы X методом Хаусхолдера: решить систему линейных уравнений $Rc = Q^T y$. Код алгоритма QR-разложения методом Хаусхолдера взять из лабораторной работы 2 курса «Вычислительные методы алгебры». Для вычислений использовать тип *float*.

В отчёте представить значения величин, указанных в задании 1. Сравнить значение величины среднеквадратичного уклонения со значением, полученным в задании 1. В этом задании графики функций строить не требуется.

Входные данные:

Вариант 7 (24%17): $10x^3(5\pi + x)^{-1/4}, [0;10]$

Использованные формулы

1) Основная система уравнений для коэффициентов полинома:

$$\begin{pmatrix} \sum_{i=0}^{30} x_i^0 & \sum_{i=0}^{30} x_i^1 & \sum_{i=0}^{30} x_i^2 \\ \sum_{i=0}^{30} x_i^1 & \sum_{i=0}^{30} x_i^2 & \sum_{i=0}^{30} x_i^3 \\ \sum_{i=0}^{30} x_i^2 & \sum_{i=0}^{30} x_i^3 & \sum_{i=0}^{30} x_i^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{30} y_i \\ \sum_{i=0}^{30} y_i x_i^1 \\ \sum_{i=0}^{30} y_i x_i^2 \end{pmatrix}$$

2) Узлы: $x_i = a + \frac{b-a}{n} i, i = \overline{0,n}, n = 30$

3) Несмещенная оценка дисперсии случайных ошибок для выборок малого объема при k=3 и n=30, $\hat{\sigma} = \sqrt{\frac{RSS}{28}} = \sqrt{\frac{1}{28}\sum_{i=0}^{n}\left(y_i - P_2(x_i)\right)^2}$,

Результат работы программы и анализ полученных результатов

```
Gauss: 184.45169-246.57874x+65.46427x^2
mistake: 86.88848

Householder: 184.50989-246.60231x+65.46623x^2
mistake: 86.88843
```

Как видим, во втором случае ошибка меньше. Это связано с выбором метода решения СЛАУ.

Листинг программы

Файл “util.CommonAnalysis.java”

```
package util;

import java.util.ArrayList;
import java.util.Collections;
import java.util.function.Function;

import static java.lang.Math.PI;
import static java.lang.Math.cos;
import static java.lang.Math.pow;

/**
 * Contains static common functions, connected with numerical analysis
 */
public class CommonAnalysis {

    /**
     * Table of divided differences of function <b><i>f</i></b>
defined on points <b><i>x</i></b>
     * @param x list of points
     * @param f function
     * @return table of divided differences, (i,j)=<i>f(x<sub>(j-
i)</sub> , ... ,x<sub>(j)</sub>)</i>
     */
    public static float[][] table(ArrayList<Float> x,
                                  Function<Float, Float> f) {
        float[][] result = new float[x.size()][x.size()];
        for (int i = 0; i < x.size(); i++) {
            result[0][i] = f.apply(x.get(i));
        }
        for (int i = 1; i < x.size(); i++) {
            for (int j = i; j < x.size(); j++) {
                result[i][j] = (result[i - 1][j] - result[i - 1][j -
1]) / (x.get(j) - x.get(j - i));
            }
        }
        return result;
    }

    /**
     * Mistake of interpolation in chebyshev nodes
     * @param a start of interval
     * @param b end of interval
     * @param n degree of polynom
     * @param nthDerivative nth derivative of function
     * @return numeric value of mistake
     */
    public static float bestMistake(float a,
                                    float b,
                                    int n,
```

```
                                    Function<Float, Float>
nthDerivative) {
        return (float) (maxAbsValue(a, b, nthDerivative, 0.01f) *
pow(b - a, n) * pow(2, 1 - 2 * n) / factorial(n));
    }

    /**
     * Factorial
     * @param n - argument, n>=0
     * @return numeric value of factorial
     */
    public static int factorial(int n) {
        int result = 1;
        int next = 2;
        while (next <= n) {
            result *= next;
            next += 1;
        }
        return result;
    }

    /**
     * Maximum absolute value of function on step with defined step
     * @param a start of step
     * @param b end of step
     * @param f function
     * @param step step of finding values of function
     * @return numeric value of maximum absolute value
     */
    public static float maxAbsValue(float a,
                                    float b,
                                    Function<Float, Float> f,
                                    float step) {
        float result = 0;
        for (float i = a; i <= b; i += step) {
            float newResult = Math.abs(f.apply(i));
            if (newResult > result) {
                result = newResult;
            }
        }
        return result;
    }

    /**
     * Mistakes of interpolating on uniform nodes in the middles of
intervals
     * @param a start of interval
     * @param b end of interval
     * @param m number of points
```

```
     * @param nthDerivative nth derivative of function
     * @return list of numeric values of mistakes
     */
    public static ArrayList<Float> mistake(float a,
                                           float b,
                                           int m,
                                           Function<Float,Float>
nthDerivative) {
        float h = (b - a) / (m - 1);
        ArrayList<Float> mistakes=new ArrayList<>();
        for (int i = 0; i < m - 1; i++) {
            float point = (float) ((b - a) * 0.5 * (2f * i + 1) / (m
- 1f));
            float t = (point - a) / h;
            float mistake = maxAbsValue(a,b,nthDerivative,0.01f);
            for (int j = 0; j < m; j++) {
                mistake *= (t - j);
            }
            mistake = (float) (Math.abs(mistake) * pow(h, m) /
factorial(m));
            mistakes.add(mistake);
        }
        return mistakes;

    }

    /**
     * Chebyshev nodes in interval
     * @param a start of interval
     * @param b end of interval
     * @param m number of points
     * @return list of chebyshev nodes
     */
    public static ArrayList<Float> chebyshevNodes(float a,
                                                  float b,
                                                  int m) {
        ArrayList<Float> list = new ArrayList<>(m);
        for (int i = 0; i < m; i++) {
            float result = (float) ((b + a) / 2 + (b - a) / 2 *
cos(PI * (2 * i + 1) / (2 * m)));
            list.add(result);
        }
        Collections.reverse(list);
        return list;
    }

    /**
     * Uniform nodes in interval
     * @param a start of interval
```

```
     * @param b end of interval
     * @param n number of nodes
     * @return list of nodes
     */
    public static ArrayList<Float> uniformNodes(float a, float b,
int n) {
        ArrayList<Float> list=new ArrayList<>(n);
        for (int i = 0; i <= n; i++) {
            list.add(a+(b-a)*i/n);
        }
        return list;
    }

    /**
     * Function values in nodes
     * @param nodes - nodes
     * @param f - function
     * @return list of values
     */
    public static ArrayList<Float> functionValues(ArrayList<Float>
nodes, Function<Float,Float> f) {
        ArrayList<Float> list=new ArrayList<>(nodes.size());
        nodes.forEach(x -> list.add(f.apply(x)));
        return list;
    }
}
```

Файл “util.Polynom.java”

```
package util;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.function.Function;

/**
 * Class of polynomial function
 */
public class Polynom implements Function<Float, Float> {

    /**
     * Polynomial coefficients
     */
    private ArrayList<Float> a;

    /**
     * Constructs polynomial function with its coefficients
     * @param a list of coefficients, <i>P<sub>n</sub>(x)=a<sub>0</sub>+
     *          a<sub>1</sub>x<sup>1</sup>+...+a<sub>n</sub>x<sup>n</sup></i>
     */
    public Polynom(ArrayList<Float> a) {
        this.a = a;
    }
```

```
    /**
     * Find value of polynom in point
     * @param x point
     * @return value P(x)
     */
    @Override
    public Float apply(Float x) {
        LinkedList<Float> list=new LinkedList<>();
        for (int i = 0; i < a.size(); i++) {
            float start=a.get(i);
            for (int j = 0; j < i; j++) {
                start*=x;
            }
            list.add(start);
        }
        return (float)list.stream().mapToDouble(hey->hey).sum();
    }
}
```

Файл “util.LinearSystemSolver.java”

```
package util;

import com.sun.istack.internal.NotNull;

import java.util.ArrayList;

/**
 * Class, that gives static methods to solve linear systems <b>Ax=b</b>, where
<b>A</b> - <i>square matrix</i>, <b>x</b>, <b>b</b> - <i>columns</i>
 */
public class LinearSystemSolver {

    /**
     * Solves linear system <b>Ax=b</b> using householder method of QR-
decomposition
     * @param squareMatrix matrix A
     * @param column column b
     * @return column x
     */
    public static ArrayList<Float> solveHouseholder(float[][] squareMatrix,
float[] column) {
        int n = squareMatrix.length;
        float[][] matrix = new float[n][n];
        float[] b = new float[n];
        for (int i = 0; i < n; i++) {
            b[i] = column[i];
            System.arraycopy(squareMatrix[i], 0, matrix[i], 0, n);
        }
        float[] diagonalElements = new float[n];
        for (int k = 0; k < n; k++) {
            float norm = columnNorm(matrix, k, k);
            if (norm != 0f) {
                if (matrix[k][k] < 0) {
                    norm = -norm;
                }
                diagonalElements[k] = -norm;
                for (int i = k; i < n; i++) {
                    matrix[i][k] /= norm;
                }
```

```
                matrix[k][k] = matrix[k][k] + 1f;
                for (int j = k + 1; j < n; j++) {
                    float s = 0f;
                    for (int i = k; i < n; i++) {
                        s += matrix[i][k] * matrix[i][j];
                    }
                    s = -s / matrix[k][k];
                    for (int i = k; i < n; i++) {
                        matrix[i][j] += s * matrix[i][k];
                    }
                }
            }
        }
        for (int k = 0; k < n; k++) {
            double s = 0.0;
            for (int i = k; i < n; i++) {
                s += matrix[i][k] * b[i];
            }
            s = -s / matrix[k][k];
            for (int i = k; i < n; i++) {
                b[i] += s * matrix[i][k];
            }

        }
        for (int k = n - 1; k >= 0; k--) {
            b[k] /= diagonalElements[k];
            for (int i = 0; i < k; i++) {
                b[i] -= b[k] * matrix[i][k];
            }
        }
        ArrayList<Float> x = new ArrayList<>(n);
        for (float v : b) {
            x.add(v);
        }
        return x;
    }

    /**
     * Solves linear system <b>Ax=b</b> using gauss method of QR-decomposition
     * @param squareMatrix matrix A
     * @param column column b
     * @return column x
     */
    public static ArrayList<Float> solveGauss(@NotNull float[][] squareMatrix,
@NotNull float[] column) {
        int n = squareMatrix.length;
        float[][] matrix = new float[n][n];
        float[] b = new float[n];
        for (int i = 0; i < n; i++) {
            b[i] = column[i];
            System.arraycopy(squareMatrix[i], 0, matrix[i], 0, n);
        }
        for (int j = 0; j < n; j++) {
            int index = indexMaxInCol(matrix, j);
            float[] temp = matrix[j];
            matrix[j] = matrix[index];
            matrix[index] = temp;
            float tempV = b[j];
            b[j] = b[index];
```

```
            b[index] = tempV;
            for (int i = j + 1; i < matrix.length; i++) {
                matrix[i][j] = matrix[i][j] / matrix[j][j];
            }
            for (int i = j + 1; i < matrix.length; i++) {
                for (int k = j + 1; k < matrix.length; k++) {
                    matrix[i][k] = matrix[i][k] - matrix[i][j] * matrix[j][k];
                }
            }
        }
        float[] y = new float[n];
        ArrayList<Float> x = new ArrayList<>(n);
        for (int i = 0; i < n; i++) {
            x.add(0f);
        }
        for (int i = 0; i < n; ++i) {
            float sum = 0;
            for (int j = 0; j < i; ++j) {
                sum += matrix[i][j] * y[j];
            }
            y[i] = b[i] - sum;
        }
        for (int i = n - 1; i >= 0; --i) {
            float sum = 0;
            for (int j = i + 1; j < n; ++j) {
                sum += matrix[i][j] * x.get(j);
            }
            x.set(i, (y[i] - sum) / matrix[i][i]);
        }
        return x;
    }

    /**
     * Finds index of maximum element in column
     * @param matrix matrix
     * @param column column
     * @return index
     */
    private static int indexMaxInCol(@NotNull float[][] matrix, int column) {
        int result = column;
        for (int i = column + 1; i < matrix.length; ++i) {
            if (Math.abs(matrix[i][column]) > Math.abs(matrix[result][column])) {
                result = i;
            }
        }
        return result;
    }

    /**
     * Finds norm of column starting from kth row
     * @param matrix matrix
     * @param j column index
     * @param k raw index
     * @return numeric value of norm
     */
    private static float columnNorm(@NotNull float[][] matrix, int j, int k) {
        float result = 0;
        for (int i = k; i < matrix.length; i++) {
            result = hypot(result, matrix[i][j]);
```

```
        }
        return result;
    }

    /**
     * Function (a^2+b^2)^(1/2) without under/overflowing
     * @param a first argument
     * @param b second argument
     * @return numeric value of function
     */
    private static float hypot(float a, float b) {
        float result;
        if (Math.abs(a) > Math.abs(b)) {
            result = b / a;
            result = (float) (Math.abs(a) * Math.sqrt(1 + result * result));
        } else if (b != 0) {
            result = a / b;
            result = (float) (Math.abs(b) * Math.sqrt(1 + result * result));
        } else {
            result = 0f;
        }
        return result;
    }

}
```

Файл "task2.OrdinaryListSquaresInterpolator.java"

```
package task2;

import util.LinearSystemSolver;
import util.Polynom;

import static util.CommonAnalysis.*;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.function.Function;

/**
 * Class, that interpolates function with polynom of 2nd degree using Gauss and
Householder methods of LS-solving
 */
public class OrdinaryListSquaresInterpolator {

    public static void main(String[] args) {
        try {
            OrdinaryListSquaresInterpolator interpolator = new
OrdinaryListSquaresInterpolator(
                    0f, 10f, 30, x -> (float) (10 * x * x * x * Math.pow(Math.PI *
5 + x, -0.25)));
            interpolator.createOutput();
        } catch (IOException exc) {
            exc.printStackTrace();
        }
    }

    /**
```

```
     * function to interpolate
     */
    private Function<Float, Float> f;
    /**
     * number of intervals
     */
    private int n;
    /**
     * start of interval
     */
    private float a;
    /**
     * end of interval
     */
    private float b;
    /**
     * coefficients of polynomial, calculated using Gauss method
     */
    private ArrayList<Float> coeffsGauss;
    /**
     * polynomial with coefficients, calculated using Gauss method
     */
    private Polynom polynomGauss;
    /**
     * coefficients of polynomial, calculated using Householder method
     */
    private ArrayList<Float> coeffsHouseholder;
    /**
     * polynomial with coefficients, calculated using Householder method
     */
    private Polynom polynomHouseholder;
    /**
     * mistake of Gauss polynomial
     */
    private float mistakeGauss;
    /**
     * mistake of Householder polynomial
     */
    private float mistakeHouseholder;

    /**
     * Constructs the interpolator class
     * @param a start of interval
     * @param b end of interval
     * @param n number of intervals
     * @param f function
     */
    public OrdinaryListSquaresInterpolator(float a, float b, int n,
Function<Float, Float> f) {
        this.f = f;
        this.n = n;
        this.a=a;
        this.b=b;
        process();
    }

    /**
     * processes input data
     */
```

```
private void process() {
    ArrayList<Float> x1= uniformNodes(a,b,n);
    ArrayList<Float> y=functionValues(x1,f);
    ArrayList<Float> x0=new ArrayList<>(x1.size()),
            x2=new ArrayList<>(x1.size()),
            x3=new ArrayList<>(x1.size()),
            x4=new ArrayList<>(x1.size()),
            yx=new ArrayList<>(x1.size()),
            yx2=new ArrayList<>(x1.size());
    x1.forEach(x -> {
        x0.add(1f);
        x2.add(x*x);
        x3.add(x*x*x);
        x4.add(x*x*x*x);
    });
    for (int i = 0; i < y.size(); i++) {
        yx.add(x1.get(i)*y.get(i));
        yx2.add(x2.get(i)*y.get(i));
    }
    float[][] matrix=new float[3][3];
    matrix[0][0]= (float) x0.stream().mapToDouble(x->x).sum();
    matrix[0][1]= (float) x1.stream().mapToDouble(x->x).sum();
    matrix[0][2]= (float) x2.stream().mapToDouble(x->x).sum();
    matrix[1][0]=matrix[0][1];
    matrix[1][1]=matrix[0][2];
    matrix[1][2]=(float) x3.stream().mapToDouble(x->x).sum();
    matrix[2][0]=matrix[1][1];
    matrix[2][1]=matrix[1][2];
    matrix[2][2]=(float) x4.stream().mapToDouble(x->x).sum();
    float[] column=new float[3];
    column[0]=(float) y.stream().mapToDouble(x->x).sum();
    column[1]=(float) yx.stream().mapToDouble(x->x).sum();
    column[2]=(float) yx2.stream().mapToDouble(x->x).sum();
    coeffsGauss =LinearSystemSolver.solveGauss(matrix,column);
    polynomGauss =new Polynom(coeffsGauss);
    coeffsHouseholder =LinearSystemSolver.solveHouseholder(matrix,column);
    polynomHouseholder =new Polynom(coeffsHouseholder);
    float sum=0;
    for (int i = 0; i < x1.size(); i++) {
        float temp=y.get(i)- polynomGauss.apply(x1.get(i));
        temp*=temp;
        sum+=temp;
    }
    mistakeGauss = (float) Math.sqrt(1./28.*sum);
    sum=0;
    for (int i = 0; i < x1.size(); i++) {
        float temp=y.get(i)- polynomHouseholder.apply(x1.get(i));
        temp*=temp;
        sum+=temp;
    }
    mistakeHouseholder = (float) Math.sqrt(1./28.*sum);
}

/**
 * creates output of task2
 * @throws IOException
 */
public void createOutput() throws IOException {
    PrintWriter writer=new PrintWriter("task2out/output.txt");
```

```
            writer.print("Gauss: ");
            writer.print(coeffsGauss.get(0));
            if (coeffsGauss.get(1)>0) {
                writer.print("+"+ coeffsGauss.get(1)+"x");
            } else if (coeffsGauss.get(1)<0) {
                writer.print(coeffsGauss.get(1)+"x");
            }
            if (coeffsGauss.get(2)>0) {
                writer.print("+"+ coeffsGauss.get(2)+"x^2");
            } else if (coeffsGauss.get(2)<0) {
                writer.print(coeffsGauss.get(2)+"x^2");
            }
            writer.println();
            writer.println("mistake: "+ mistakeGauss);
            writer.println();
            writer.print("Householder: ");
            writer.print(coeffsHouseholder.get(0));
            if (coeffsHouseholder.get(1)>0) {
                writer.print("+"+ coeffsHouseholder.get(1)+"x");
            } else if (coeffsHouseholder.get(1)<0) {
                writer.print(coeffsHouseholder.get(1)+"x");
            }
            if (coeffsHouseholder.get(2)>0) {
                writer.print("+"+ coeffsHouseholder.get(2)+"x^2");
            } else if (coeffsHouseholder.get(2)<0) {
                writer.print(coeffsHouseholder.get(2)+"x^2");
            }
            writer.println();
            writer.println("mistake: "+ mistakeHouseholder);
            writer.close();
        }
    }
```