МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине: Непрерывные математические модели

на тему: Построение сглаживающих сплайнов с использованием кусочно-полиномиальных эрмитовых базисных функций третьего порядка в двумерной области по произвольно зашумленным наборам данных

Вариант №3

Факультет: ФПМИ

Группа: ПММ-21

Выполнил: Сухих А.С.

Проверили: к.т.н Киселев Д.С., Патрушев И.И.

Дата выполнения: 20.12.22

Отметка о защите:

Новосибирск 2022

Цель работы: Разработать программу построения сглаживающего сплайна с использованием кусочно-полиномиальных эрмитовых базисных функций третьего порядка в двумерной области и опробовать её при решении задач фильтрации для произвольных наборов зашумленных данных.

Ход работы:

1. Построение сплайна по кусочно-линейным базисным функциям

Первоначально была разработана программа для построения кусочно-линейных сплайнов по набору точек. В качестве языка программирования был выбран C++, для решения СЛАУ была использована библиотека Eigen, отрисовка графиков производилась с помощью gnuplot.

Программе на вход подаётся два файла - файл с координатами точек в формате «x y» и файл с узлами сетки конечных элементов. На основе этих файлов генерируются элементы в виде структуры Element, содержащей в себе точки, попадающие в данный элемент.

Для кусочно-линейного сплайна были использованы следующие линейные базисные функции:

$$\check{\psi}_1 = \frac{x_2 - x}{h_k}, \check{\psi}_2 = \frac{x - x_2}{h_k}$$

На основании данных из файлов и базисных функций рассчитываются матрица A и вектор b.

$$A = \sum_{j=1}^{k} \omega_j \psi(x_j) \psi^T(x_j), b = \sum_{j=1}^{k} \omega_j \psi(x_j) f_j$$

В результате решения СЛАУ $Aq = b$ был получен вектор весов q. Чтобы снизить влияние выбросов вычисляется отклонение в каждой точке исходных данных от среднего отклонения. При превышении отклонения в точке в два и более раза вес данной точки уменьшается вдвое. По измененным весам вновь рассчитывается вектор q.

Результат работы программы по набору из 20 точек с 5 узлами элементов представлен на рисунке 1.

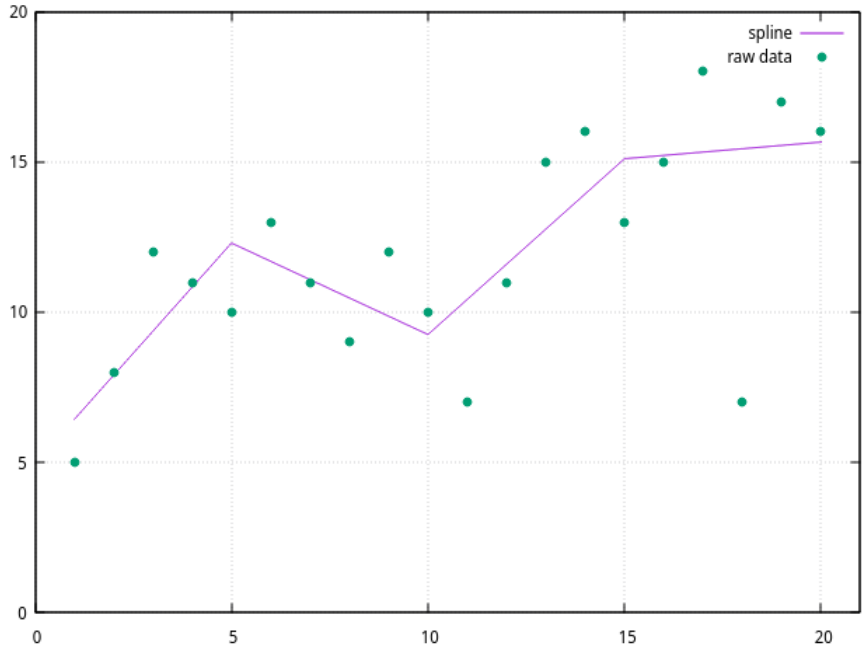

Рисунок 1 — сплайн на основе кусочно-линейных базисных функций по набору 20 точек

2. Построение сплайна по эрмитовым базисным функциям

Аналогично п.1 была разработана программа для построения сплайна по эрмитовым базисным функциям. Построение сплайна реализовано в виде класса Spline, в конструктор которому подаётся вектор точек координат и вектор узлов конечно-элементной сетки.

Были использованы следующие базисные функции:

$$\check{\psi}_1(\xi)=1-3\xi^2+2\xi^2,\check{\psi}_2(\xi)=\xi-2\xi^2+\xi^3,$$
$$\check{\psi}_3(\xi)=3\xi^2-2\xi^3,\check{\psi}_4(\xi)=-\xi^2+\xi^3,$$

где $\xi=\frac{x-x_2}{h_k}$

Для возможности регулирования гладкости сплайна также был добавлен параметр регуляризации α, влияющее на величину первых производных сплайна. Для регуляризации была использована локальная матрица жесткости

$$G = \frac{\alpha}{30 h_k} \begin{pmatrix} 36 & 3h_k & -36 & 3h_k \\ 3h_k & 4h_k^2 & -3h_k & -h_k^2 \\ -36 & -3h_k & 36 & -3h_k \\ 3h_k & -h_k^2 & -3h_k & 4h_k^2 \end{pmatrix}$$

Построенные сплайны на графиках сравниваются сглаживающим сплайном gnuplot с параметрами по умолчанию.

Результаты работы программы по построению сплайна по засоренным данным приведены на рисунках 2-4.

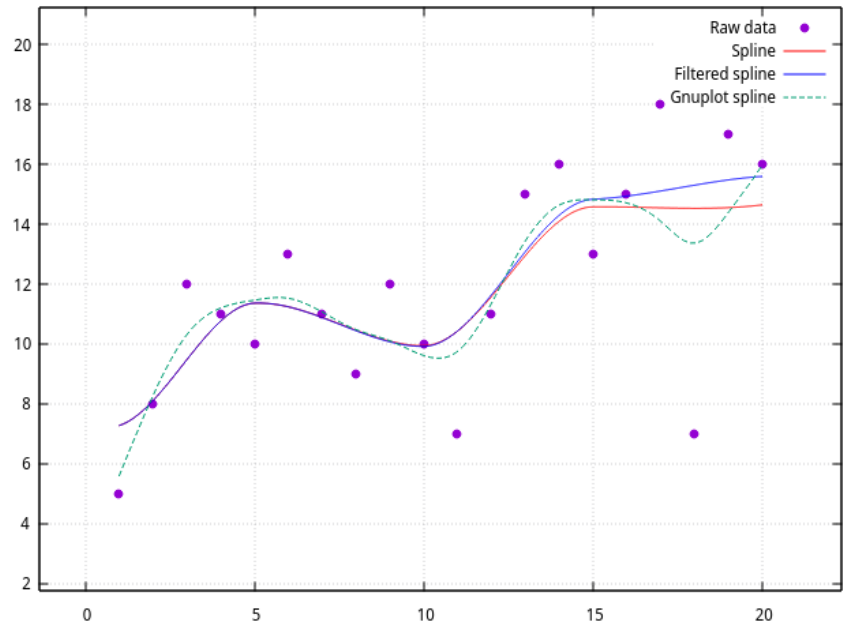

Рисунок 2 — кубический сплайн по набору 20 точек, параметр регуляризации α = 1

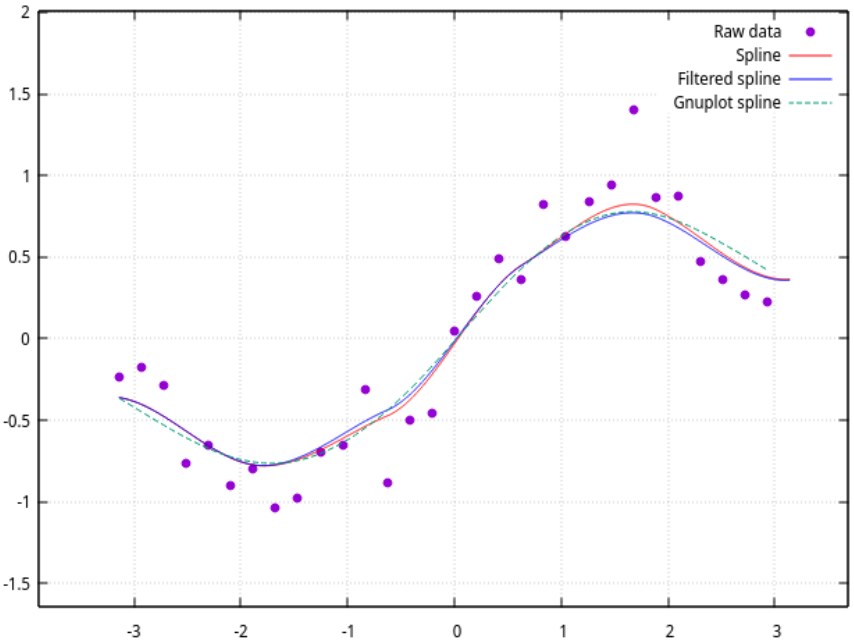

Рисунок 3 — кубический сплайн засоренной функции sin(x), параметр регуляризации α = 1, уровень засорения 0,2

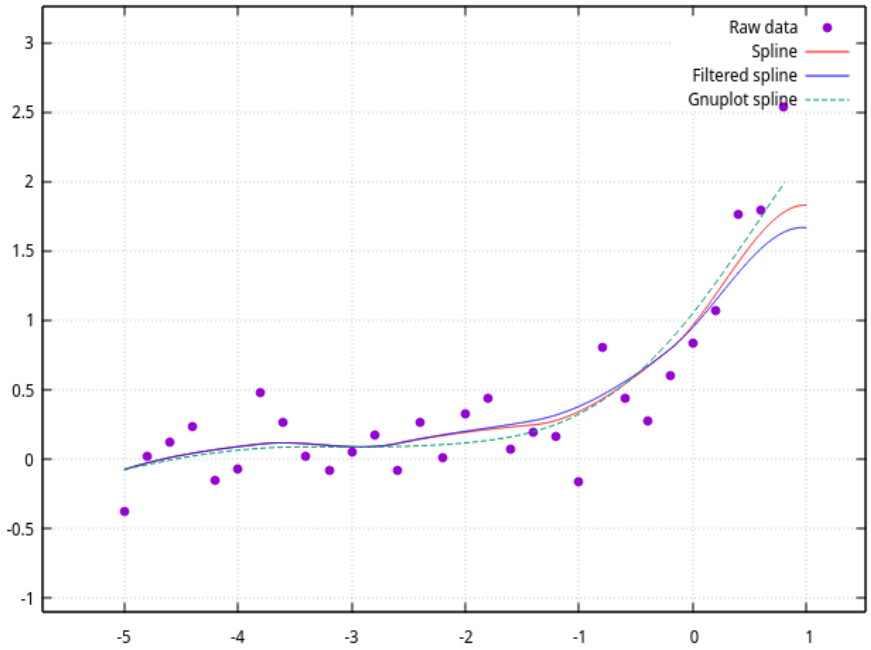

Рисунок 4 — кубический сплайн засоренной функции e^x, параметр регуляризации $\alpha = 1$, уровень засорения 0,2

На рисунках красный график с подписью Spline обозначен построенный сплайн без фильтрации выбросов, а синий подписью Filtered Spline — с фильтрацией выбросов.

На рисунке 5 продемонстрировано изменение сплайна в зависимости от параметра регуляризации на примере функции *sin(x)*:

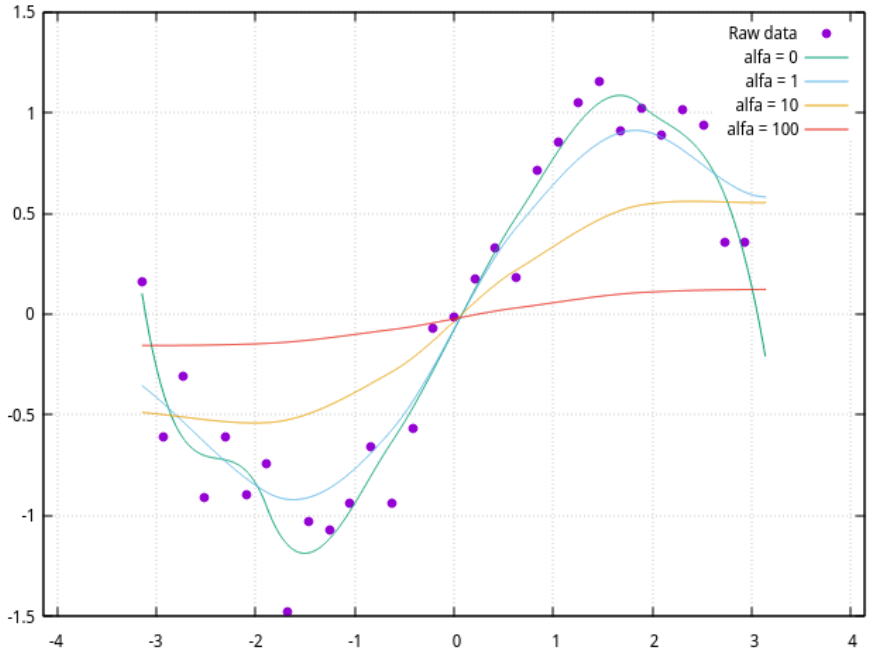

Рисунок 5 — сравнение кубических сплайнов с различными параметрами регуляризации

Вывод: В ходе лабораторной работы была разработана программа для построения кубических сплайнов в двумерной области по засоренным данным.

В разработанной программе присутствует процедура фильтрации выбросов экспериментальных данных. По графикам можно судить об уменьшении влияния выбросов, однако разработанный алгоритм определения выбросов все ещё достаточно неточен, о чем свидетельствует уменьшение производной при анализе экспоненциальной функции, в то время как производная самой функции быстро возрастает.

Также результаты, показанные на графиках сравнения сплайнов с различными параметрами регуляризации, говорят о существенном влиянии параметра регуляризации α на сплайн. Так, при увеличении значения параметра сплайн становился более гладким засчёт уменьшения первых производных сплайна, однако при этом начинал существенно отдаляться от экспериментальных значений, при дальнейшем увеличении параметра превращаясь практически в прямую. Наиболее оптимальное значение параметра можно принять равное 1.

ПРИЛОЖЕНИЯ

Заголовочный файл spline.h

```
#ifndef SPLINE_H
#define SPLINE_H
#include <vector>
#include <Eigen/Dense>
#include <gnuplot-iostream.h>

#define BASIS_FUNCTIONS_COUNT 4
#define MAX_DEVIATION_TIMES 2

typedef std::pair<float, float> Point;

struct SplinePoint {
    float x;
    float y;
    float weight;

    SplinePoint(float x_in, float y_in, float w_in = 1);
};

struct Element {
    float n1;
    float n2;
    std::vector<SplinePoint> values;
};

class Spline
{
    float regularizationAlfa;
    std::vector<Element> elements;
    std::vector<float> splineWeights;
    float basisFunc(float x1, float x2, float x, int func_index);
    Eigen::Matrix4f regularizationAlfaMatrix(float x1, float x2);
    std::vector<float> findSplineWeightsVector();
public:
    Spline(std::vector<Point> sourcePoints, std::vector<float> nodes, float regAlfa = 1);
    std::vector<Point> getSplineData(int splinePointsNumber = 1000);
    void filterSpline();
};

#endif // SPLINE_H
```

Файл с исходным кодом spline.cpp

```
#include "spline.h"

SplinePoint::SplinePoint(float x_in, float y_in, float w_in){
    x = x_in;
    y = y_in;
    weight = w_in;
}

std::ostream& operator<< (std::ostream &os, const Element &elem){
    os << "[" << elem.n1 << ", " << elem.n2 << "]" << ": ";
    for (auto it = elem.values.begin(); it != elem.values.end(); it++)
```

```
        os << "(" << it->x << ", " << it->y << "), ";
    os << std::endl;
    return os;
}

Spline::Spline(std::vector<Point> sourcePoints, std::vector<float> nodes, float regAlfa)
{
    int nodeCount = nodes.size();
    int elemCount = nodeCount - 1;
    elements.resize(elemCount);

    for (int i = 0; i < nodes.size(); i++){
        if (i == 0)
          elements[i].n1 = nodes[i];
        else if (i == elemCount)
          elements[i-1].n2 = nodes[i];
        else {
          elements[i-1].n2 = nodes[i];
          elements[i].n1 = nodes[i];
        }
    }

    for (auto it = sourcePoints.begin(); it != sourcePoints.end(); it++){
        for (int j = 0; j < nodes.size(); j++){
            if (it->first >= elements[j].n1 && it->first <= elements[j].n2){
                if (j != elements.size() - 1 && it->first == elements[j].n2) continue; // координата, попадающая на узел,
должна включиться только в последний узел
                elements[j].values.push_back(SplinePoint(it->first, it->second)); // вставляем в значения элемента
преобразованную к SplinePoint точку
            }
        }
    }

    std::cout << "Filled elements:" << std::endl;
    for(auto it = elements.begin(); it != elements.end(); it++)
        std::cout << *it;

    regularizationAlfa = regAlfa;
    splineWeights = findSplineWeightsVector();
}

std::vector<Point> Spline::getSplineData(int splinePointsNumber)
{
    std::vector<Point> splineData;
    float xmin = elements.begin()->n1;
    float xmax = (elements.end() - 1)->n2;
    std::cout << "Xmin: " << xmin << ", Xmax: " << xmax << std::endl;
    float step = (xmax - xmin) / splinePointsNumber;

    for (float x = xmin; x < xmax; x = x + step){
        float y = 0;
        for (int i = 0; i < elements.size(); i++){
            if (x >= elements[i].n1 && x <= elements[i].n2){
                for (int nu = 0; nu < BASIS_FUNCTIONS_COUNT; nu++)
                    y += splineWeights[2 * i + nu] * basisFunc(elements[i].n1, elements[i].n2, x, nu);
            }
        }
        splineData.push_back(Point(x, y));
    }
```

```
    return splineData;
}

void Spline::filterSpline()
{
    std::vector<std::vector<float>> deltaVector;
    deltaVector.resize(elements.size());
    int countDelta = 0;
    do {
        float deltaSumm = 0;
        splineWeights = findSplineWeightsVector();
        int point_count = 0;
        for (int i = 0; i < elements.size(); i++){
            deltaVector[i].resize(elements[i].values.size());
            int j = 0;
            for (auto dataInElement = elements[i].values.begin(); dataInElement != elements[i].values.end();
dataInElement++){
                point_count++;
                float splineY = 0;
                for (int nu = 0; nu < BASIS_FUNCTIONS_COUNT; nu++)
                    splineY += splineWeights[2 * i + nu] * basisFunc(elements[i].n1, elements[i].n2, dataInElement->x,
nu);
                float delta = abs(splineY - dataInElement->y);

                deltaVector[i][j] = delta;
                deltaSumm += delta;
                j++;
            }
        }

        countDelta = 0;
        float avgDelta = deltaSumm / point_count;
        std::cout << "Average delta: " << avgDelta << std::endl;
        for(int i = 0; i < deltaVector.size(); i++){
            for(int j = 0; j < deltaVector[i].size(); j++){
                if (deltaVector[i][j] >= avgDelta * MAX_DEVIATION_TIMES && elements[i].values[j].weight == 1){
                    countDelta++;
                    elements[i].values[j].weight /= MAX_DEVIATION_TIMES;
                }
            }
        }
        std::cout << "Found " << countDelta << " elements with delta exceeding average delta" << std::endl;
    }
    while(countDelta > 0);
}

// функция расчета базисной функции
// x1 - первая X-координата элемента
// x2 - вторая X-координата
// x - X-координата в промежутке между x1 и x2
// func_index - индекс базисной функции
float Spline::basisFunc(float x1, float x2, float x, int func_index){
    float h = x2 - x1;
    float ksi = (x - x1) / h;
    switch(func_index){
        case 0: return 1 - 3 * pow(ksi, 2) + 2 * pow(ksi, 3);
        case 1: return ksi - 2 * pow(ksi, 2) + pow(ksi, 3);
        case 2: return 3 * pow(ksi, 2) - 2 * pow(ksi, 3);
        case 3: return -pow(ksi, 2) + pow(ksi, 3);
    default: throw ("Incorrect index of Basis function");
```

```
    }
}

Eigen::Matrix4f Spline::regularizationAlfaMatrix(float x1, float x2)
{

    float h = x2 - x1;
    Eigen::Matrix4f matrix;
    matrix << 36, 3*h, -36, 3*h,
              3*h, 4*pow(h,2), -3*h, -pow(h,2),
              -36, -3*h, 36, -3*h,
              3*h, -pow(h,2), -3*h, 4*pow(h,2);
    return 1/(30*h) * matrix;
}

std::vector<float> Spline::findSplineWeightsVector()
{
    int nodeCount = elements.size() + 1;
    Eigen::MatrixXf matrixA(nodeCount*2, nodeCount*2);
    Eigen::VectorXf vectorB(nodeCount*2);
    matrixA.setZero();
    vectorB.setZero();

    // рассчет матрицы A
    for (int i = 0; i < elements.size(); i++){
        Eigen::Matrix4f regMatrix = regularizationAlfaMatrix(elements[i].n1, elements[i].n2);
        for (int nu = 0; nu < BASIS_FUNCTIONS_COUNT; nu++){
            for (int mu = 0; mu < BASIS_FUNCTIONS_COUNT; mu++){
                float matrixCell = 0;
                for (auto dataInElement = elements[i].values.begin(); dataInElement != elements[i].values.end();
dataInElement++){
                    // рассчет ячейки матрицы
                    float psinu = basisFunc(elements[i].n1, elements[i].n2, dataInElement->x, nu);
                    float psimu = basisFunc(elements[i].n1, elements[i].n2, dataInElement->x, mu);
                    matrixCell += dataInElement->weight * psinu * psimu;
                }
                matrixA(2 * i + nu, 2 * i + mu) += matrixCell + regularizationAlfa * regMatrix(nu, mu);

            }
        }
    }
    std::cout << "Global matrix A:" << std::endl;
    std::cout << matrixA << std::endl;
    // рассчет вектора b
    for (int i = 0; i < elements.size(); i++){
        for (int nu = 0; nu < BASIS_FUNCTIONS_COUNT; nu++){
            for (auto dataInElement = elements[i].values.begin(); dataInElement != elements[i].values.end();
dataInElement++){
                // рассчет ячейки вектора
                float psinu = basisFunc(elements[i].n1, elements[i].n2, dataInElement->x, nu);
                vectorB(2 * i + nu) += dataInElement->weight * psinu * dataInElement->y;
            }

        }
    }
    std::cout << "Global vector b:" << std::endl;
    std::cout << vectorB << std::endl;

    Eigen::VectorXf resVector = matrixA.colPivHouseholderQr().solve(vectorB);
    std::cout << "Result Q vector:" << resVector << std::endl;
```

```
    std::vector<float> qVector(resVector.data(), resVector.data() + resVector.size());

    return qVector;
}
```

main.cpp

```
#include <string>
#include <fstream>
#include <cmath>
#include <random>

#include <spline.h>

bool compare(Point p1, Point p2) {
    return p1.second<p2.second;
}

void drawSpline(std::vector<Point> data, Spline spline){
    auto splineData = spline.getSplineData();
    spline.filterSpline();
    std::vector<Point> splineFilteredData = spline.getSplineData();

    float xmin = data[0].first;
    float xmax = data[data.size() - 1].first;
    float ymin = std::min_element(data.begin(), data.end(), compare)->second;
    float ymax = std::max_element(data.begin(), data.end(), compare)->second;
    float xdelta = (xmax - xmin) / 8;
    float ydelta = (ymax - ymin) / 4;
    Gnuplot gp;
//    gp << "set terminal qt title 'Generated Data'\n";
    gp << "set grid\n \
          set xrange" << "[" << xmin-xdelta << ":" << xmax+xdelta << "]\n \
          set yrange" << "[" << ymin-ydelta << ":" << ymax+ydelta << "]\n";
    gp << "plot '-' with points ps 1 pt 7 title 'Raw data', \
               '-' with lines linecolor rgb 'red' lt 3 title 'Spline', \
               '-' with lines linecolor rgb 'blue' lt 1 title 'Filtered spline', \
               '-' with lines linecolor 10 dashtype 2 smooth acsplines title 'Gnuplot spline'\n";
    gp.send1d(data);
    gp.send1d(splineData);
    gp.send1d(splineFilteredData);
    gp.send1d(data);
}

void splineFromFile(std::ifstream& dataFile, std::ifstream& splineFile){
    std::string buf;
    getline(dataFile, buf);
    int size = stoi(buf);
    std::vector<Point> rawPoints;
    for (int i = 0; i < size; i++){
        getline(dataFile, buf);
        int space_pos = buf.find(' ');
        int x = stoi(buf.substr(0, space_pos));
        int val = stoi(buf.substr(space_pos, buf.length() - 1));
        rawPoints.push_back(Point(x, val));
    }

    getline(splineFile, buf);
```

```
    size = stoi(buf);
    std::vector<float> nodes;
    for (int i = 0; i < size; i++){
        getline(splineFile, buf);
        nodes.push_back(stof(buf));
    }

    Spline spline(rawPoints, nodes);
    drawSpline(rawPoints, spline);
}

void splineFromFunction(int pointsCount, int elementsCount, float noiseLevel, float regAlfa = 1){
    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<float> dist(0,1);

    std::vector<Point> data(pointsCount);
    float xmin = -M_PI;
    float xmax = M_PI;
//    float xmin = -5;
//    float xmax = 1;
    float step = (xmax - xmin) / pointsCount;
    float x = xmin;
    for (int i = 0; i < pointsCount; i++){
        data[i] = Point(x, sin(x) + dist(gen)*noiseLevel);
//        data[i] = Point(x, exp(x) + dist(gen)*noiseLevel);
        x += step;
        std::cout << "data[" << i << "] = (" << data[i].first << ", " << data[i].second << ")" << std::endl;
    }

    int nodesCount = (int)(pointsCount / elementsCount);

    step = (xmax - xmin) / (nodesCount - 1);
    x = xmin;
    std::vector<float> nodes(nodesCount);
    for(int i = 0; i < nodesCount; i++){
        nodes[i] = x;
        x += step;
    }

    for (auto it = nodes.begin(); it != nodes.end(); it++)
        std::cout << "node " << *it << std::endl;



    Spline spline(data, nodes, regAlfa);
    drawSpline(data, spline);
}

void regularizationTest(int pointsCount, int elementsCount, float noiseLevel){
    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<float> dist(0,1);

    std::vector<Point> data(pointsCount);
    float xmin = -M_PI;
    float xmax = M_PI;
    float step = (xmax - xmin) / pointsCount;
    float x = xmin;
    for (int i = 0; i < pointsCount; i++){
```

```
        data[i] = Point(x, sin(x) + dist(gen)*noiseLevel);
        x += step;
    }
    int nodesCount = (int)(pointsCount / elementsCount);
    step = (xmax - xmin) / (nodesCount - 1);
    x = xmin;
    std::vector<float> nodes(nodesCount);
    for(int i = 0; i < nodesCount; i++){
        nodes[i] = x;
        x += step;
    }

    Spline spline0(data, nodes, 0);
    Spline spline1(data, nodes, 1);
    Spline spline10(data, nodes, 10);
    Spline spline100(data, nodes, 100);

    Gnuplot gp;
    gp << "set grid\n \
          set xrange" << "[" << xmin-1 << ":" << xmax+1 << "]\n \
          set yrange" << "[" << -1.5 << ":" << 1.5 << "]\n";
    gp << "plot '-' with points ps 1 pt 7 title 'Raw data', \
                '-' with lines linecolor 2 title 'alfa = 0', \
                '-' with lines linecolor 3 title 'alfa = 1', \
                '-' with lines linecolor 4 title 'alfa = 10', \
                '-' with lines linecolor 7 title 'alfa = 100'\n";
    gp.send(data);
    gp.send1d(spline0.getSplineData());
    gp.send1d(spline1.getSplineData());
    gp.send1d(spline10.getSplineData());
    gp.send1d(spline100.getSplineData());
}

int main()
{
    try{
        std::ifstream dataFile("../simplified_lab1/data.txt");
        if(!dataFile.good()) throw std::runtime_error("Data file not found");
        std::ifstream splineFile("../simplified_lab1/spline.txt");
        if(!splineFile.good()) throw std::runtime_error("Spline file not found");

        splineFromFile(dataFile, splineFile);

        splineFromFunction(30, 5, 0.2);
//        regularizationTest(30, 5, 0.2);


    } catch(std::runtime_error& e){
        std::cout << "File not found: " << e.what();
        return 1;
    }

    return 0;
}
```