БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики Кафедра
вычислительной математики

Никончик Даниил Викторович
ОТЧЕТ ПО МЕТОДАМ ВЫЧИСЛЕНИЙ
студента 2 курса 12 группы
Лабораторная работа №4

Преподаватель
Бондарь И.В.

Минск 2020

Написать программу, которая по данному набору точек строит интерполяционный параметрический кубический сплайн с соответствующими варианту граничными условиями, а также позволяет вычислять значение этого сплайна в указанной точке t.

Описание класса Spline, в котором происходит подсчёт значения в точке и построение сплайна по точкам и стратегии размещения параметрических узлов (естественные или равномерные):

```
class Spline
{
public:
    enum class NodeMode { EquiDistant, Natural };
private:
    std::vector<double> alphaX, betaX, gammaX, deltaX;
    std::vector<double> alphaY, betaY, gammaY, deltaY;
    std::vector<double> paramNodes;
    double computeX(double t) const;
    double computeY(double t) const;
public:
    Spline(NodeMode mode, const
std::vector<std::pair<double, double>> &points);
    std::pair<double, double> operator()(double t) const;
};
```

Расчёт коэффициентов интерполяционных сплайнов и значения сплайна в точке:

```
Spline::Spline(
    NodeMode mode,
    const std::vector<std::pair<double, double>> &points
)
{
    if (mode == NodeMode::EquiDistant)
        paramNodes =
std::move(equiDistantNodes(points.size()));
    else
        paramNodes = std::move(naturalNodes(points));
    std::vector<double> xs, ys;
    splitPoints(points, xs, ys);
    computeSplineCoeffs(paramNodes, xs, alphaX, betaX,
gammaX, deltaX);
    computeSplineCoeffs(paramNodes, ys, alphaY, betaY,
gammaY, deltaY);
}

double
```

```
Spline::computeX(double t) const
{
    return computeSplineValue(t, paramNodes, alphaX, betaX,
gammaX, deltaX);
}

double
Spline::computeY(double t) const
{
    return computeSplineValue(t, paramNodes, alphaY, betaY,
gammaY, deltaY);
}

std::pair<double, double>
Spline::operator()(double t) const
{
    return { computeX(t), computeY(t) };
}
```

Вспомогательные функции для расчёта коэффициентов сплайна:

```
namespace
{

double
sqr(double x)
{
    return x * x;
}

std::vector<double>
equiDistantNodes(int N)
{
    std::vector<double> result;
    for (int i = 0; i < N; ++i)
        result.push_back((double)i / (N-1));
    return result;
}

std::vector<double>
naturalNodes(const std::vector<std::pair<double, double>>
&points)
{
    std::vector<double> result;
```

```
    double t = 0;
    result.push_back(t);
    for (int i = 1; i < points.size(); ++i) {
        t += std::sqrt(sqr(points[i].first - points[i-
1].first) +
                       sqr(points[i].second - points[i-
1].second));
        result.push_back(t);
    }
    for (int i = 0; i < points.size(); ++i)
        result[i] /= t;
    return result;
}

void
splitPoints(
    const std::vector<std::pair<double, double>> &points,
    std::vector<double> &xs, std::vector<double> &ys
)
{
    for (int i = 0; i < points.size(); ++i) {
        xs.push_back(points[i].first);
        ys.push_back(points[i].second);
    }
}

void
computeC(const std::vector<double> &xs, std::vector<double>
&c)
{
    c.assign(xs.size(), 0);
    for (int i = 2; i < xs.size() - 1; ++i)
        c[i] = ((xs[i] - xs[i-1]) / (xs[i+1] - xs[i-1]));
}

void
computeE(const std::vector<double> &xs, std::vector<double>
&e)
{
    e.assign(xs.size(), 0);
    for (int i = 1; i < xs.size() - 2; ++i)
        e[i] = ((xs[i+1] - xs[i]) / (xs[i+1] - xs[i-1]));
```

```
}

void
computeB(
    const std::vector<double> &xs, const
std::vector<double> &ys,
    std::vector<double> &b)
{
    b.assign(xs.size(), 0);
    for (int i = 1; i < xs.size() - 1; ++i) {
        double firstDiff = (ys[i] - ys[i-1]) / (xs[i] -
xs[i-1]);
        double secondDiff = (ys[i+1] - ys[i]) / (xs[i+1] -
xs[i]);
        b.push_back(6 * (secondDiff - firstDiff) / (xs[i+1]
- xs[i-1]));
    }
}

void
computeGamma(
    const std::vector<double> &xs, const
std::vector<double> &ys,
    std::vector<double> &gamma
)
{
    gamma.assign(xs.size(), 0);
    std::vector<double> c, e, b, d;
    computeC(xs, c);
    computeE(xs, e);
    computeB(xs, ys, b);
    d.assign(xs.size(), 2);
    for (int i = 2; i < xs.size() - 1; ++i) {
        d[i] -= e[i-1] * c[i] / d[i-1];
        b[i] -= b[i-1] * c[i] / d[i-1];
    }
    gamma[xs.size() - 2] = b[xs.size() - 2] / d[xs.size() -
2];
    for (int i = xs.size() - 3; i >= 1; --i)
        gamma[i] = (b[i] - e[i] * gamma[i+1]) / d[i];
}
```

```
void
computeSplineCoeffs(
    const std::vector<double> &xs, const
std::vector<double> &ys,
    std::vector<double> &alpha, std::vector<double> &beta,
    std::vector<double> &gamma, std::vector<double> &delta
)
{
    computeGamma(xs, ys, gamma);
    alpha.push_back(0);
    beta.push_back(0);
    gamma.push_back(0);
    for (int i = 1; i < xs.size(); ++i) {
        alpha.push_back(ys[i]);
        delta.push_back((gamma[i] - gamma[i-1]) / (xs[i] -
xs[i-1]));
        beta.push_back(
            (ys[i] - ys[i-1]) / (xs[i] - xs[i-1]) +
            (2 * gamma[i] + gamma[i-1]) * (xs[i] - xs[i-1])
/ 6
        );
    }
}

double
computeSplineValue(
    double t, const std::vector<double> &nodes,
    const std::vector<double> &alpha, const
std::vector<double> &beta,
    const std::vector<double> &gamma, const
std::vector<double> &delta
)
{
    auto bound = std::upper_bound(nodes.begin(),
nodes.end(), t);
    if (nodes.end() == bound)
        bound = nodes.end() - 1;
    int i = bound - nodes.begin();
    double p = t - nodes[i];
    return alpha[i] + beta[i] * p + gamma[i] / 2 * p * p +
           delta[i] / 6 * p * p * p;
}
```

```
} // namespace
```

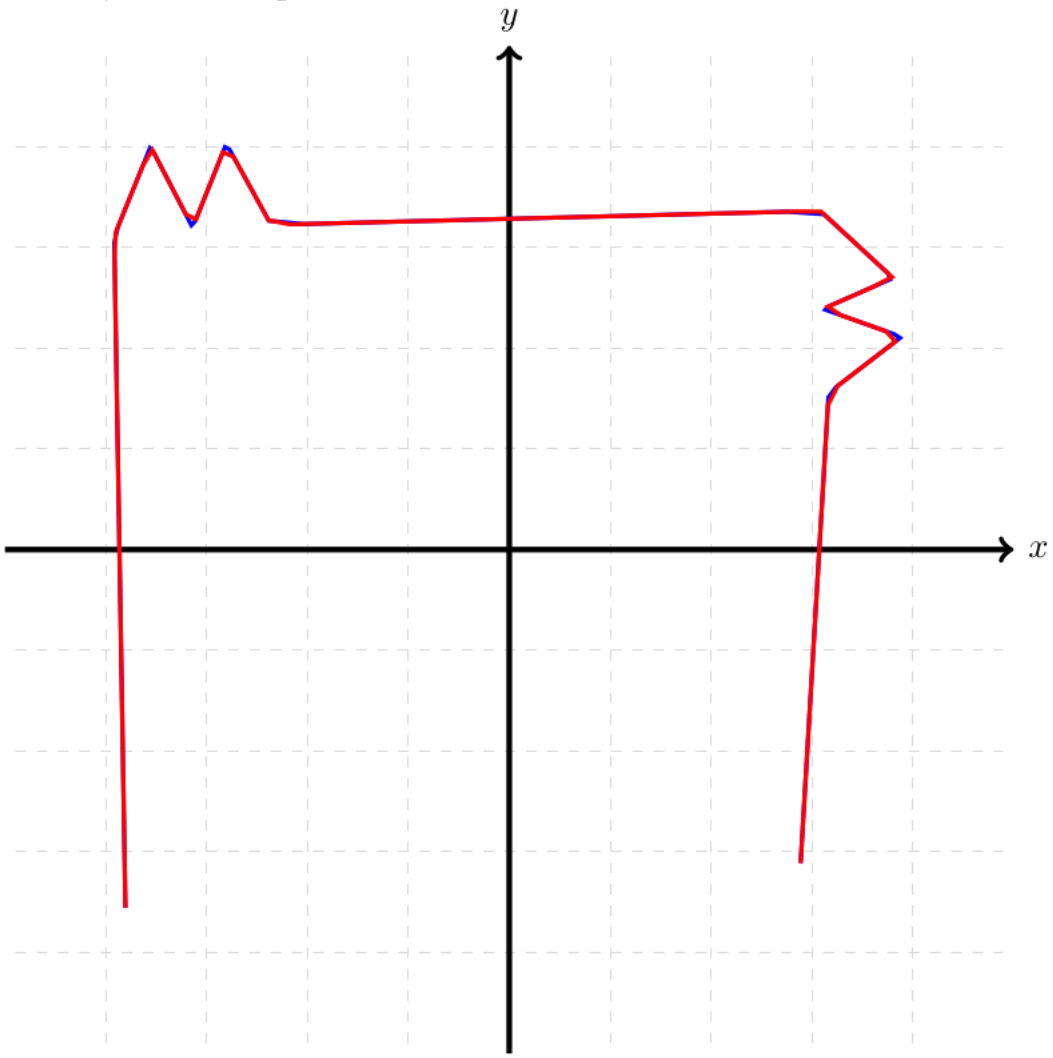
Главный файл, в котором даны исходные данные для построения сплайна и производится вывод его точек:

```
#include <bits/stdc++.h>
#include "spline.h"

std::vector<std::pair<double, double>> points =
{
    { -3.80556, -3.55556 }, { -3.91667,  3.11111 }, { -
3.55556,  4.00000 },
    { -3.13889,  3.19444 }, { -2.80556,  4.02778 }, { -
2.36111,  3.22222 },
    {  3.08333,  3.36111 }, {  3.80556,  2.69444 },
{  3.11111,  2.38889 },
    {  3.88889,  2.11111 }, {  3.16667,  1.55556 },
{  2.88889, -3.11111 }
};

const int MaxPoints = 100;

void
printSpline(const Spline &spline)
{
    for (int i = 0; i <= MaxPoints; ++i) {
        auto point = spline((double)i / MaxPoints);
        std::cout << "(" << point.first << ", " <<
point.second << ")\n";
    }
}

int
main()
{
    Spline firstSpline(Spline::NodeMode::EquiDistant,
points);
    Spline secondSpline(Spline::NodeMode::Natural, points);
    std::cout << "EquiDistant points spline:\n";
    printSpline(firstSpline);
    std::cout << "Natural points spline:\n";
    printSpline(secondSpline);
```

```
}
```

Построить совмещённые графики полученных сплайнов для двух типов параметрических узлов: равномерных и естественных. В чём на практике проявляется разница между равномерными и естественными параметрическими узлами?

График изображён на рис. ниже. Синей линии соответствует сплайн с равномерными узлами, а красной – с естественными.



Если использовать равномерные параметрические узлы, то это выражается в том, что расстояние между точками никак не учитывается при построении сплайна; то есть, по сути, при построении сплайна по каждой координате мы “поставили” узлы интерполяции не на их “ожидаемых” местах, а просто на одном и том же расстоянии; если бы мы сделали такое на обычном сплайне, то получилось бы, что на каких-то отрезках сплайн “сжался”, а на других “растянулся”. Благодаря естественным узлам мы можем добиться того, чтобы таких эффектов (сжатия и растяжения) не было, ведь узлы интерполяции стоят там, где и нужно.