|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к домашнему заданию**

по дисциплине:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Технология мультимедиа\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

на тему: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_Разработка интерактивной программы вычисления и построения графика классического\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_эрмитового сплайна на наборе из 10 контрольных точек\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_РТ5-61Б\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Забурунов Л. В.\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Афанасьев Г. И.\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2021 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Руководитель секции \_\_\_\_ИУ5\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение домашнего задания**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Технология мультимедиа\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по теме \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_Разработка интерактивной программы вычисления и построения графика классического\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_эрмитового сплайна на наборе из 10 контрольных точек\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_РТ5-61Б\_\_\_Забурунов Леонид Вячеславович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

График выполнения домашнего задания: 25% к \_\_\_ нед., 50% к \_\_\_ нед., 75% к \_\_ нед., 100% к \_\_\_ нед.

Техническое задание Создать графическую программу, в которой реализовано\_\_\_\_\_\_\_\_\_\_\_

\_\_интерактивное взаимодействие со сплайновой кривой из 10 точек\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_(с возможностью изменять значение производной в каждой точке)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_\_\_\_\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « 12 » марта 2021 г.

**Руководитель домашнего задания**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**Афанасьев Г. И.\_\_**\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Забурунов Л. В.\_\_\_**

(Подпись, дата) (И.О.Фамилия)

**Оглавление**

[Цель выполнения работы 4](#_Toc73397302)

[Ход выполнения работы 5](#_Toc73397303)

[1. Общее описание программы 5](#_Toc73397304)

[2. Класс MainWindow 6](#_Toc73397305)

[3. Класс HermiteBuilder 9](#_Toc73397306)

[4. Класс DerivativeEditor 16](#_Toc73397307)

[Результаты работы 19](#_Toc73397308)

# Цель выполнения работы

Целью выполнения данной работы является приобретение навыков в построении интерактивных мультимедиа систем. Кубические сплайны являются распространённым инструментом работы в специализированном программном обеспечении (проектирование, анимации и др.).

# Ход выполнения работы

# Общее описание программы

Для решения задачи будем использовать следующие инструменты:

1. *C++ Qt* версии 5.15 – для создания приложения с графическим интерфейсом;
2. *QCustomPlot* – специализированная библиотека для работы с графиками (является надстройкой над *Qt*);
3. *ALGLIB* – *C++*-библиотека для решения множества вычислительных задач, в том числе сплайновой интерполяции.

С точки зрения структуры программы выделим следующие основные элементы:

1. Базовый класс основного окна *MainWindow*;
2. Обработчик интерактивной работы на рабочей области графиков и построения сплайна *HermiteBuilder*;
3. Обработчик действий с редактором производных *DerivativeEditor*.

# Класс MainWindow

В рамках данного класса мы создаём, настраиваем рабочую область и инициализируем все компоненты; для обработки событий мыши генерируем сигналы.

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "menu.h"

#include "qcustomplot.h"

#include "hermitebuilder.h"

#include "derivativeeditor.h"

#include <QMainWindow>

#include <QKeyEvent>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **HermiteBuilder**;

*class* **DerivativeEditor**;

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

QCustomPlot\* **GetPlotWidget**();

*//HermiteBuilder\** *GetHermiteBuilder();*

DerivativeEditor\* **GetDerEditor**();

*private*:

Ui::MainWindow \*ui;

Menu \*menu;

QCustomPlot \*graph;

HermiteBuilder \*hermite;

DerivativeEditor \*derivativeEditor;

void **SetupHermite**();

*protected*:

void ***keyPressEvent***(QKeyEvent \*event);

signals:

void **AddPointKeyPressed**();

void **RemovePointKeyPressed**();

};

#endif *//* *MAINWINDOW\_H*

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "menu.h"

#include <QBoxLayout>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

*this*->menu = *new* Menu(*this*);

*this*->setMenuBar(*this->menu->menuBar*);

*this*->graph = ui->plot;

*this*->derivativeEditor = *new* DerivativeEditor(*ui->derivativeSlider*, *ui->sliderValueText*, *ui->spinBox*);

SetupHermite();

}

*//* *Настройка* *рабочей* *области* *отрисовки* *+* *инициализация* *обработчика* *для* *работы* *со* *сплайном*

void MainWindow::**SetupHermite**() {

*/\** *В* *QCustomPlot* *есть* *возможность* *добавлять* *на* *виджет* *несколько* *графиков* *и* *взаимодействовать* *с* *каждым* *по* *отдельности.*

*В* *связи* *с* *этим* *будем* *использовать* *два* *графика:*

*1.* *Верхний* *слой,* *где* *будут* *располагаться* *контрольные* *точки,* *с* *которыми* *можно* *взаимодействовать;*

*2.* *Нижний* *слой,* *где* *будет* *отрисовываться* *сплайн* *на* *основе* *заданных* *контр.* *точек.*

*Соответственно,* *graph(0)* *отвечает* *за* *отрисовку* *точек,* *graph(1)* *-* *за* *кривую.\*/*

*this*->graph->addGraph(*graph*->*xAxis*, *graph*->*yAxis*);

*this*->graph->addGraph(*graph*->*xAxis*, *graph*->*yAxis*);

*//* *Параметры* *отрисовки* *узловых* *точек*

*this*->graph->graph(1)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ScatterShape::*ssCircle*,

QColor(255, 170, 0),

QColor(255, 170, 0),

14));

*//* *Помимо* *самих* *точек,* *будем* *показывать* *штрихпунктирной* *линией* *кусочно-линейную* *функцию* *на* *основе* *узловых* *точек*

QPen pen0;

pen0.setStyle(Qt::PenStyle::*DashDotLine*);

pen0.setColor(QColor(0, 36, 61));

pen0.setWidth(2);

*this*->graph->graph(1)->setPen(pen0);

*//* *Параметры* *отрисовки* *выбранной* *узловой* *точки* *(к* *оранжевой* *точке* *добавится* *красная* *обводка)*

QPen decPen;

decPen.setWidth(14);

decPen.setBrush(Qt::GlobalColor::*red*);

QCPSelectionDecorator\* decorator = *new* QCPSelectionDecorator();

decorator->setPen(decPen);

decorator->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ScatterShape::*ssCircle*,

QPen(QBrush(Qt::GlobalColor::*red*), 3),

QBrush(QColor(255, 170, 0)),

18), QCPScatterStyle::ScatterProperty::*spAll*);

decorator->setBrush(Qt::GlobalColor::*red*);

*this*->graph->graph(1)->setSelectionDecorator(*decorator*);

*//* *Параметры* *выбора* *элементов* *на* *верхнем* *слое*

*this*->graph->setSelectionTolerance(50);

*this*->graph->setInteractions(QCP::*iSelectPlottables* | QCP::*iRangeZoom*);

*this*->graph->graph(1)->setSelectable(QCP::SelectionType::*stSingleData*);

*//* *То* *же* *самое* *для* *графика,* *где* *будет* *рисоваться* *сплайн*

*this*->graph->graph(0)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ScatterShape::*ssNone*)); *//* *нужны* *только* *линии*

*this*->graph->graph(0)->setSelectable(QCP::SelectionType::*stNone*);

QPen pen1;

pen1.setStyle(Qt::PenStyle::*SolidLine*);

pen1.setCapStyle(Qt::PenCapStyle::*RoundCap*);

pen1.setWidth(4);

pen1.setBrush(QColor(0, 61, 20));

*this*->graph->graph(0)->setPen(pen1);

*//* *Инициализация* *обработчика* *построения* *сплайна*

*this*->hermite = *new* HermiteBuilder(*this*);

}

QCustomPlot\* MainWindow::**GetPlotWidget**() {

*return* *this*->graph;

}

DerivativeEditor\* MainWindow::**GetDerEditor**() {

*return* *this*->derivativeEditor;

}

void MainWindow::***keyPressEvent***(QKeyEvent \*event) {

*if* (event->type() == QKeyEvent::*KeyPress* && event->key() == Qt::*Key\_Plus*) {

*emit* AddPointKeyPressed();

} *else* *if* (event->type() == QKeyEvent::*KeyPress* && event->key() == Qt::*Key\_Minus*) {

*emit* RemovePointKeyPressed();

}

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

# Класс HermiteBuilder

Здесь происходит обработки всех действий с рабочей областью: взаимодействие с контрольными точками (в том числе изменение их производных) и отрисовка графиков как реакция на изменения.

**hermitebuilder.h**

#ifndef HERMITEBUILDER\_H

#define HERMITEBUILDER\_H

#include <QObject>

#include <QWidget>

#include <QList>

#include <QPointF>

#include <QMouseEvent>

*class* **MainWindow**;

*class* **QCustomPlot**;

*class* **QCPGraph**;

*//* *Структура* *данных* *для* *хранения* *всех* *исходных* *данных* *сплайна*

*class* **HermitePoint** {

*public*:

QPointF point;

double derivative;

**HermitePoint**(QPointF p, double d) {

point = p;

derivative = d;

}

*static* int **IndexOfPoint**(QList<HermitePoint> \*list, QPointF point) {

*for* (int i = 0; i < list->size(); i = i + 1) {

*if* (list->at(i).point == point) *return* i;

}

*return* -1;

}

};

*class* **HermiteBuilder** : *public* QWidget

{

Q\_OBJECT

*public*:

**HermiteBuilder**();

**HermiteBuilder**(MainWindow \*mw);

void **AddPoint**(QPointF \*newPoint);

void **RemovePoint**(int index);

~***HermiteBuilder***();

*private*:

*const* int MAX\_LENGTH = 10;

*//const* *int* *SPLINE\_STEPS* *=* *40;*

QList<HermitePoint> \*values;

*//* *Ссылки* *на* *оба* *слоя* *рабочей* *области*

QCPGraph\* basePointsGraph;

QCPGraph\* splineGraph;

*//* *Индекс* *выбранной* *точки* *(чтобы* *знать,* *что* *в* *массиве* *менять)*

int selectedPointIndex;

*//* *Смещение* *от* *мыши* *до* *курсора* *на* *начало* *перемещения* *(для* *сохранения)*

QPointF \*offsetBeforeDragging;

*//* *Флаг* *активности* *процесса* *перемещения*

bool isDragging;

*//* *Критерии* *сортировки*

*static* bool **ComparePoints\_X**(HermitePoint point1, HermitePoint point2);

*static* bool **ComparePoints\_Y**(HermitePoint point1, HermitePoint point2);

*//* *Перерисовка* *графиков* *после* *любых* *событий,* *изменяющих* *значения* *узловых* *точек*

void **Redraw**(bool newRangeApplied);

*//* *Построение* *сплайна* *на* *основе* *заданных* *значений*

void **DrawSplines**();

signals:

void **PointsSizeChanged**(int newSize);

void **SelectedPointChanged**(int newIndex);

*private* slots:

void **OnMouseClick**(QMouseEvent \*event);

void **OnMouseMove**(QMouseEvent \*event);

void **OnMouseRelease**(QMouseEvent \*event);

void **OnAddingNewPoint**();

void **OnRemovingPoint**();

void **OnChangingSelection**();

*public* slots:

void **OnDerivativeChanged**(int index, double value);

};

#endif *//* *HERMITEBUILDER\_H*

**hermitebuilder.cpp**

#include "hermitebuilder.h"

#include <QDebug>

#include <QSharedPointer>

#include "mainwindow.h"

#include "alglib/interpolation.h"

HermiteBuilder::**HermiteBuilder**()

{

*this*->values = *new* QList<HermitePoint>();

*this*->selectedPointIndex = -1;

*this*->offsetBeforeDragging = *new* QPointF(0.f, 0.f);

*this*->isDragging = *false*;

}

HermiteBuilder::**HermiteBuilder**(MainWindow \*mw) : HermiteBuilder() {

*auto* qcp = mw->GetPlotWidget();

*this*->basePointsGraph = qcp->graph(1);

*this*->splineGraph = qcp->graph(0);

*//* *Подвязываем* *события* *мыши* *для* *интерактивного* *вз-я*

connect(qcp, &QCustomPlot::mousePress, *this*, &HermiteBuilder::OnMouseClick);

connect(qcp, &QCustomPlot::mouseMove, *this*, &HermiteBuilder::OnMouseMove);

connect(qcp, &QCustomPlot::mouseRelease, *this*, &HermiteBuilder::OnMouseRelease);

*//* *Подвязываем* *события* *клавиатуры* *для* *регулирования* *числа* *точек*

connect(mw, &MainWindow::AddPointKeyPressed, *this*, &HermiteBuilder::OnAddingNewPoint);

connect(mw, &MainWindow::RemovePointKeyPressed, *this*, &HermiteBuilder::OnRemovingPoint);

*//* *Привязываем* *изменение* *выбранной* *точки* *графика*

connect(qcp, &QCustomPlot::selectionChangedByUser, *this*, &HermiteBuilder::OnChangingSelection);

*//* *Подвязываем* *редактор* *производных* *для* *корректной* *работы*

connect(mw->GetDerEditor(), &DerivativeEditor::sliverDoubleValueChanged, *this*, &HermiteBuilder::OnDerivativeChanged);

connect(*this*, &HermiteBuilder::SelectedPointChanged, mw->GetDerEditor(), &DerivativeEditor::OnSelectedPointChange);

connect(*this*, &HermiteBuilder::PointsSizeChanged, mw->GetDerEditor(), &DerivativeEditor::OnPointsResize);

*//* *Указываем,* *куда* *смотреть* *для* *работы* *со* *значениями* *производных*

mw->GetDerEditor()->SetValuesListPointer(*this->values*);

}

void HermiteBuilder::**AddPoint**(QPointF \*newPoint = *nullptr*) {

*if* (*this*->values->size() >= MAX\_LENGTH) {

qDebug() << "Нельзя добавить точку, достигнут максимальный размер!";

*return*;

}

*if* (newPoint == *nullptr*) {

*if* (values->size() > 0) {

*if* (*this*->selectedPointIndex == -1) {

newPoint = *new* QPointF(values->last().point + QPointF(1.f, 1.f));

} *else* { *//* *есть* *выбранная* *точка*

*if* (*this*->selectedPointIndex == values->size() - 1) { *//* *Добавляем* *точку* *после* *крайней*

*//* *Делаем* *последнюю* *точку* *серединой* *между* *предпоследней* *и* *новой*

newPoint = *new* QPointF(values->last().point + (values->last().point - values->at(values->size() - 2).point));

} *else* *if* (*this*->selectedPointIndex > -1) { *//* *Добавляем* *точку* *между* *выбранной* *и* *следующей*

newPoint = *new* QPointF(0.5 \* (values->at(selectedPointIndex).point + values->at(selectedPointIndex + 1).point));

}

}

} *else* { *//* *первая* *точка* *на* *добавление*

newPoint = *new* QPointF(0.f, 0.f);

}

}

values->append(HermitePoint(\*newPoint, 0));

std::sort(values->begin(), values->end(), ComparePoints\_X);

Redraw(*true*);

emit PointsSizeChanged(values->size());

}

void HermiteBuilder::**RemovePoint**(int index) {

*if* (values->size() == 0) {

qDebug() << "Не осталось точек для удаления!";

*return*;

} *else* *if* (values->size() <= index || index < 0) {

qDebug() << "Неверный индекс для удаления!";

*return*;

}

values->removeAt(index);

*//* *Если* *не* *обнулять* *выбор,* *то* *пойдут* *ошибки* *IndexOutOfRangeException* *(см.* *OnMouseMove())*

*this*->selectedPointIndex = -1;

basePointsGraph->parentPlot()->deselectAll();

Redraw(*true*);

emit PointsSizeChanged(values->size());

}

bool HermiteBuilder::**ComparePoints\_X**(HermitePoint point1, HermitePoint point2) {

*return* point1.point.x() < point2.point.x();

}

bool HermiteBuilder::**ComparePoints\_Y**(HermitePoint point1, HermitePoint point2) {

*return* point1.point.y() < point2.point.y();

}

void HermiteBuilder::**Redraw**(bool newRangeApplied) {

*if* (values->isEmpty() == *true*) *return*;

*//* *Обновление* *данных*

QCPGraphDataContainer\* container = *new* QCPGraphDataContainer();

*for* (*auto* value : \*values) {

container->add(QCPGraphData(value.point.x(), value.point.y()));

}

*this*->basePointsGraph->setData(QSharedPointer<QCPGraphDataContainer>(*container*));

*//* *Настройка* *границ*

*if* (newRangeApplied == *true*) {

*//* *Смотрим* *на* *текущую* *область,* *занимаемую* *графиком*

double xMin = std::min\_element(values->begin(), values->end(), ComparePoints\_X)->point.x(),

xMax = std::max\_element(values->begin(), values->end(), ComparePoints\_X)->point.x(),

yMin = std::min\_element(values->begin(), values->end(), ComparePoints\_Y)->point.y(),

yMax = std::max\_element(values->begin(), values->end(), ComparePoints\_Y)->point.y();

*//qDebug()* *<<* *xMin* *<<* *xMax* *<<* *";* *"* *<<* *yMin* *<<* *yMax;*

*//* *Смотрим* *на* *то,* *какие* *на* *виджете* *границы* *сейчас*

QCPRange xRange = basePointsGraph->parentPlot()->xAxis->range(),

yRange = basePointsGraph->parentPlot()->yAxis->range();

double xThreshold = 0.05 \* (xRange.upper - xRange.lower),

yThreshold = 0.05 \* (yRange.upper - yRange.lower);

*//qDebug()* *<<* *xRange* *<<* *yRange;*

*//* *Если* *график* *слишком* *близко* *к* *границе* *или* *выходит* *за* *неё,* *то* *меняем* *границы*

*if* (xMin - xRange.lower < xThreshold|| yMin - yRange.lower < yThreshold ||

xRange.upper - xMax < xThreshold || yRange.upper - yMax < yThreshold) {

basePointsGraph->parentPlot()->xAxis->setRange(xMin - 2 \* xThreshold, xMax + 2 \* xThreshold);

basePointsGraph->parentPlot()->yAxis->setRange(yMin - 2 \* yThreshold, yMax + 2 \* yThreshold);

}

}

basePointsGraph->parentPlot()->replot();

DrawSplines();

}

void HermiteBuilder::**DrawSplines**() {

*if* (values->size() < 2) *return*;

*//* *Загрузка* *модели* *сплайна*

alglib::spline1dinterpolant \*spline = *new* alglib::spline1dinterpolant();

alglib::real\_1d\_array xArray, yArray, dArray;

xArray.setlength(values->size());

yArray.setlength(values->size());

dArray.setlength(values->size());

*for* (int i = 0; i < values->size(); i = i + 1) {

xArray[i] = values->at(i).point.x();

yArray[i] = values->at(i).point.y();

dArray[i] = values->at(i).derivative;

}

*//* *Построение* *интерполяционной* *кривой*

alglib::spline1dbuildhermite(xArray, yArray, dArray, *\*spline*);

*//* *Отрисовка*

QCPGraphDataContainer\* container = *new* QCPGraphDataContainer();

*for* (int i = 0; i < values->size() - 1; i = i + 1) { *//* *Для* *каждой* *пары* *точек*

QPointF point1 = values->at(i).point,

point2 = values->at(i + 1).point;

int amountOfSteps = 10 + 20 \* std::sqrt(std::pow(point2.x() - point1.x(), 2) + std::pow(point2.y() - point1.y(), 2));

qDebug() << amountOfSteps;

*for* (int j = 0; j <= amountOfSteps; j = j + 1) { *//* *Для* *каждой* *промежуточной* *точки*

double x = point1.x() + *static\_cast*<float>(j) / amountOfSteps \* (point2.x() - point1.x()), *//* *линейно* *интерполируем* *по* *X*

y = alglib::spline1dcalc(\*spline, x); *//* *получаем* *результат* *по* *Y*

container->add(QCPGraphData(x, y));

}

}

*this*->splineGraph->setData(QSharedPointer<QCPGraphDataContainer>(*container*));

splineGraph->parentPlot()->replot();

}

void HermiteBuilder::**OnMouseClick**(QMouseEvent \*event) {

*if* (*this*->selectedPointIndex == -1) *return*;

*//* *Запомним* *вектор* *от* *мыши* *до* *точки,* *чтобы* *при* *переносе* *соблюдать* *его*

QPointF vector;

*this*->basePointsGraph->pixelsToCoords(event->localPos(), *vector.rx()*, *vector.ry()*);

*this*->offsetBeforeDragging = *new* QPointF(vector - *this*->values->at(selectedPointIndex).point);

*//qDebug()* *<<* *"Offset* *=* *"* *<<* *\*offsetBeforeDragging;*

*this*->isDragging = *true*;

}

void HermiteBuilder::**OnMouseMove**(QMouseEvent \*event) {

*if* (*this*->selectedPointIndex == -1 || *this*->isDragging == *false*) *return*;

*//* *Новое* *положение* *точки* *=* *положение* *мыши* *-* *offset*

QPointF vector;

basePointsGraph->pixelsToCoords(event->localPos(), *vector.rx()*, *vector.ry()*);

double temp\_d = values->at(selectedPointIndex).derivative;

*this*->values->replace(*this*->selectedPointIndex, HermitePoint(vector - \*(*this*->offsetBeforeDragging), temp\_d));

Redraw(*false*);

}

void HermiteBuilder::**OnMouseRelease**(QMouseEvent \*event) {

*if* (*this*->selectedPointIndex == -1) *return*;

*this*->offsetBeforeDragging = *new* QPointF(0.f, 0.f);

*this*->isDragging = *false*;

Redraw(*true*);

}

void HermiteBuilder::**OnAddingNewPoint**() {

*//qDebug()* *<<* *"Добавляем* *точку";*

AddPoint();

}

void HermiteBuilder::**OnRemovingPoint**() {

*if* (selectedPointIndex >= 0) {

*//qDebug()* *<<* *"Удаляем* *точку";*

RemovePoint(selectedPointIndex);

} *else* {

*//qDebug()* *<<* *"Не* *выбрана* *точка* *для* *удаления";*

}

}

void HermiteBuilder::**OnDerivativeChanged**(int index, double newValue) {

*if* (index == -1) *return*;

qDebug() << "Derivative changed: " << newValue << " for index " << index;

QPointF temp\_p(values->at(index).point);

*this*->values->replace(index, HermitePoint(temp\_p, newValue));

Redraw(*false*);

}

void HermiteBuilder::**OnChangingSelection**() {

QCPDataSelection selected = *this*->basePointsGraph->selection();

*if* (selected.dataRangeCount() == 0) { *//* *Ставим* *-1* *для* *корректной* *работы* *в* *других* *методах*

*this*->selectedPointIndex = -1;

*//qDebug()* *<<* *selectedPointIndex;*

} *else* {

*auto* point = basePointsGraph->data()->at(selected.dataRange(0).begin());

selectedPointIndex = HermitePoint::IndexOfPoint(*this->values*, QPointF(point->key, point->value));

*//qDebug()* *<<* *selected.dataRange(0).begin()* *<<* *selected.dataRange(0).end()* *<<* *point->key* *<<* *point->value;*

}

emit SelectedPointChanged(selectedPointIndex);

}

HermiteBuilder::~***HermiteBuilder***() {

*delete* values;

*delete* offsetBeforeDragging;

*delete* basePointsGraph;

*delete* splineGraph;

}

# Класс DerivativeEditor

Данный класс выделен для работы со значениями производных у контрольных точек.

**derivativeeditor.h**

#ifndef DERIVATIVEEDITOR\_H

#define DERIVATIVEEDITOR\_H

#include <QWidget>

#include <QLabel>

#include "mainwindow.h"

#include "hermitebuilder.h"

#include <QSlider>

#include <QSpinBox>

*class* **HermitePoint**;

*class* **DerivativeEditor** : *public* QWidget

{

Q\_OBJECT

*public*:

*explicit* **DerivativeEditor**(QWidget \*parent = *nullptr*);

**DerivativeEditor**(QSlider \*slider, QLabel \*valueLabel, QSpinBox\* spinBox);

void **SetValuesListPointer**(QList<HermitePoint> \*pointer);

signals:

void **sliverDoubleValueChanged**(int index, double newDoubleValue);

*private*:

*//* *Ползунок* *для* *изменения* *значения* *производной*

QSlider \*slider;

*//* *Текстовое* *поле* *для* *отображения* *текущего* *значения* *производной*

QLabel \*valueLabel;

*//* *Поле* *для* *выбора* *точки,* *значение* *которой* *меняем*

QSpinBox \*indexSpinBox;

*//* *Указатель* *на* *список,* *с* *которым* *работаем*

QList<HermitePoint> \*values;

*public* slots:

void **OnSpinBoxValueChanged**(int index);

void **OnSliderValueChanged**(int newValue);

void **OnPointsResize**(int newSize);

void **OnSelectedPointChange**(int newIndex);

};

#endif *//* *DERIVATIVEEDITOR\_H*

**derivativeeditor.cpp**

#include "derivativeeditor.h"

#include "mainwindow.h"

#include <QDebug>

*class* **HermitePoint**;

DerivativeEditor::**DerivativeEditor**(QWidget \*parent) : QWidget(*parent*)

{

}

DerivativeEditor::**DerivativeEditor**(QSlider \*slider, QLabel \*valueLabel, QSpinBox \*spinBox) {

*this*->valueLabel = valueLabel;

*this*->slider = slider;

*this*->indexSpinBox = spinBox;

*//* *При* *движении* *слайдера* *нужно* *обновлять* *значения* *в* *текстовом* *поле,* *а* *так* *же* *в* *изначальном* *массиве* *данных*

connect(slider, &QSlider::valueChanged, *this*, &DerivativeEditor::OnSliderValueChanged);

*//* *При* *смене* *точки,* *для* *которой* *меняем* *производную,* *нужно* *подгрузить* *записанное* *значение*

connect(spinBox, QOverload<int>::of(&QSpinBox::valueChanged), *this*, &DerivativeEditor::OnSpinBoxValueChanged);

}

*//* *Чтобы* *знать,* *откуда* *брать* *значения* *и* *куда* *их* *класть*

void DerivativeEditor::**SetValuesListPointer**(QList<HermitePoint> \*pointer) {

*this*->values = pointer;

}

*//* *SpinBox* *отвечает* *за* *выбор* *текущей* *рабочей* *точки*

*//* *При* *изменении* *точки* *загружаем* *в* *ползунок* *уже* *имеющееся* *значение* *из* *массива*

void DerivativeEditor::**OnSpinBoxValueChanged**(int index) {

qDebug() << "Spin box index changed:" << index << "; expected value = " << values->at(index).derivative;

*//* *Слайдер* *работает* *только* *с* *целочисленными* *значениями,* *а* *нам* *нужно* *иметь* *возможность* *задавать* *действительные* *числа*

*//* *Поэтому* *у* *слайдера* *диапазон* *[-100;* *100],* *но* *в* *реальных* *данных* *всё* *делится* *на* *10*

slider->setValue(*static\_cast*<int>(values->at(index).derivative) \* 10);

valueLabel->setText(QString::number(values->at(index).derivative));

}

*//* *Slider* *отвечает* *за* *значение* *производной* *в* *выбранной* *рабочей* *точке*

*//* *При* *изменении* *значения* *загружаем* *новое* *в* *текстовое* *поле* *и* *в* *массив*

void DerivativeEditor::**OnSliderValueChanged**(int newValue) {

qDebug() << "Slider value changed: " << newValue;

*//* *Слайдер* *работает* *только* *с* *целочисленными* *значениями,* *а* *нам* *нужно* *иметь* *возможность* *задавать* *действительные* *числа*

*//* *Поэтому* *у* *слайдера* *диапазон* *[-100;* *100],* *но* *в* *реальных* *данных* *всё* *делится* *на* *10*

double newDoubleValue = *static\_cast*<double>(newValue) / 10;

valueLabel->setText(QString::number(newDoubleValue));

emit sliverDoubleValueChanged(indexSpinBox->value(), newDoubleValue);

}

void DerivativeEditor::**OnPointsResize**(int newSize) {

qDebug() << "Points amount changed: " << newSize;

*if* (newSize == 0) {

*//* *Если* *точек* *пока* *нет,* *то* *работать* *не* *с* *чем*

*this*->indexSpinBox->setMinimum(-1);

*this*->indexSpinBox->setMaximum(-1);

} *else* {

*this*->indexSpinBox->setMinimum(0);

*this*->indexSpinBox->setMaximum(newSize - 1);

}

}

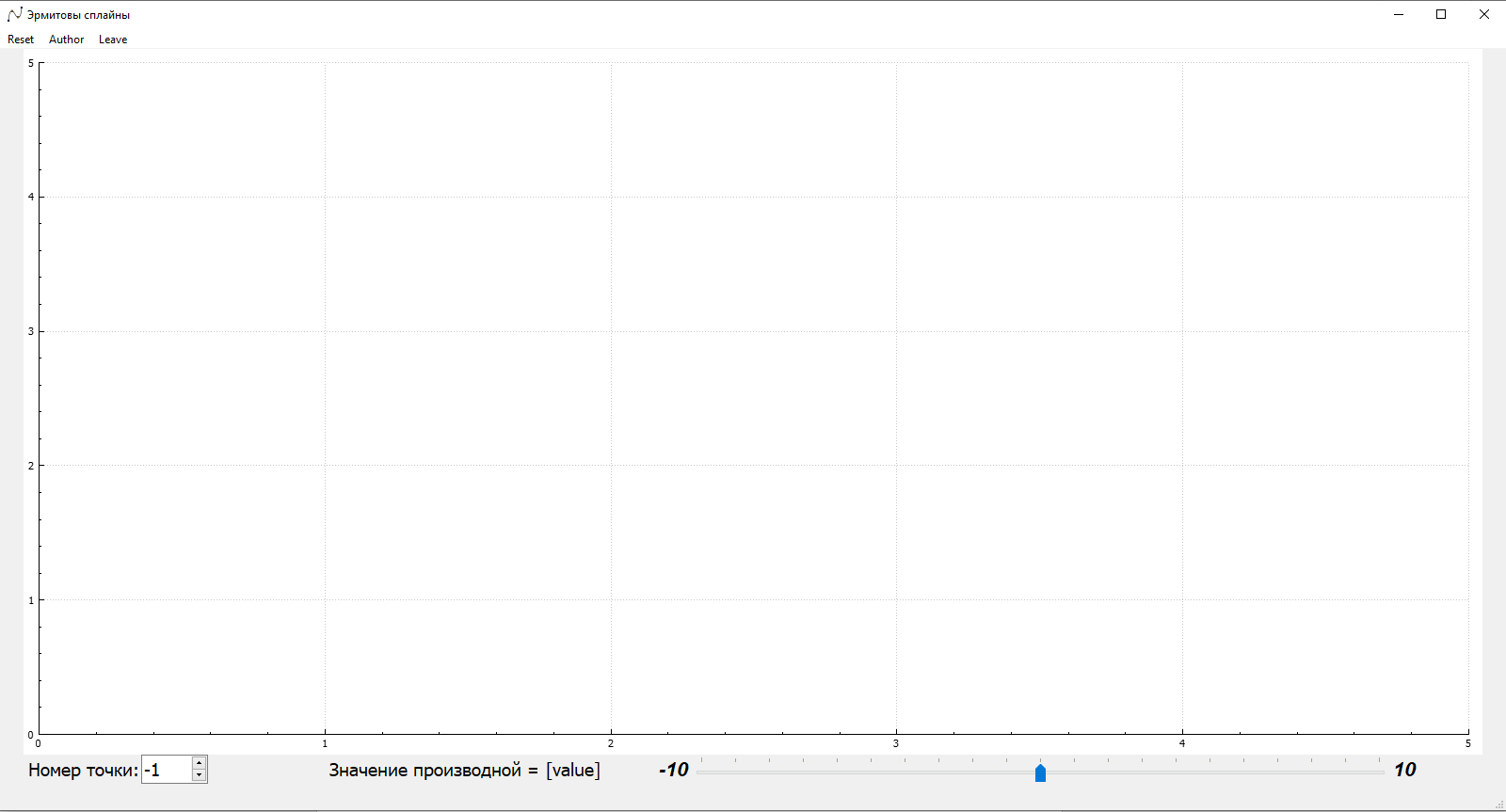
void DerivativeEditor::**OnSelectedPointChange**(int index) {

indexSpinBox->setValue(index);

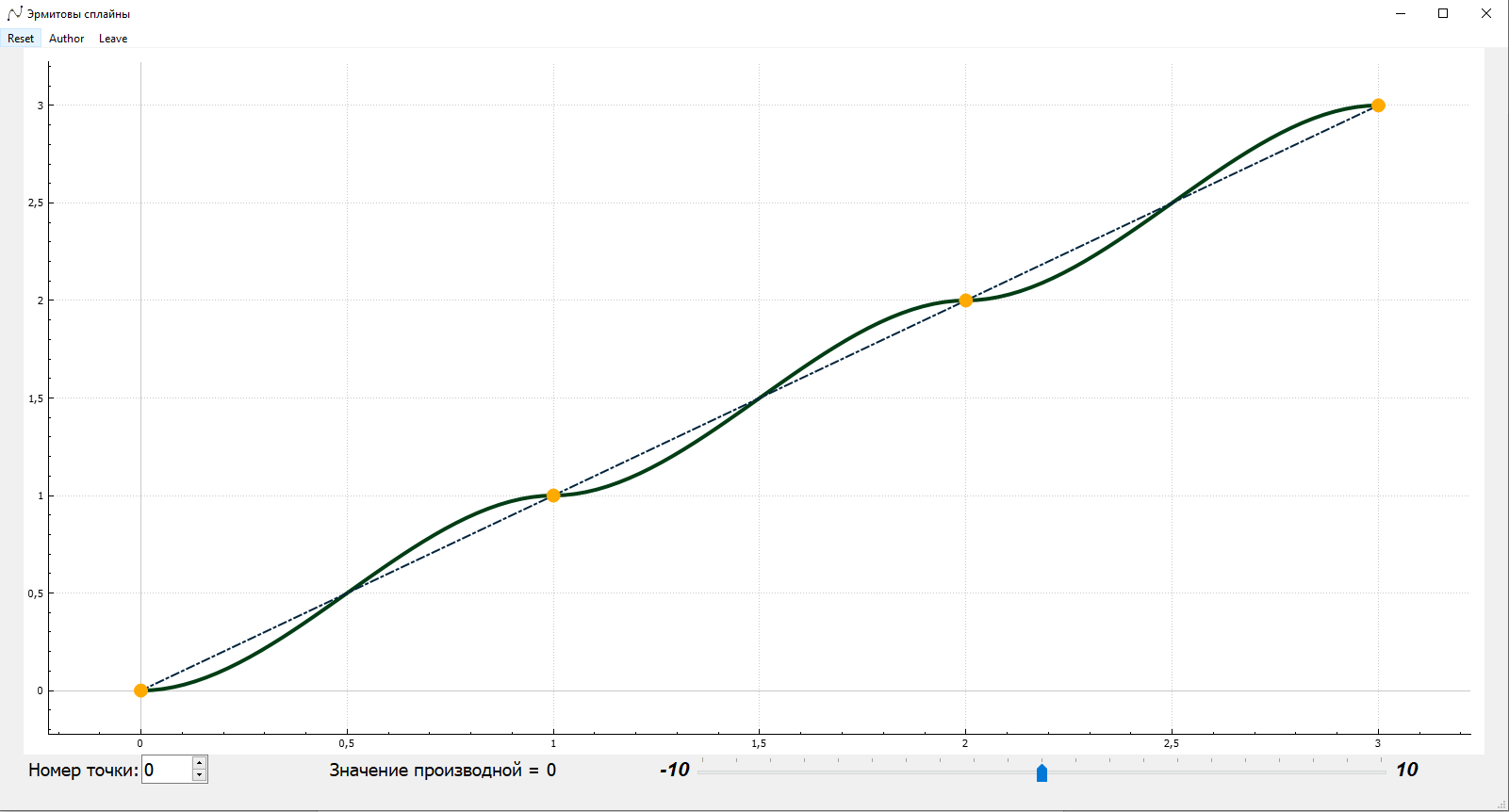
}

# Результаты работы

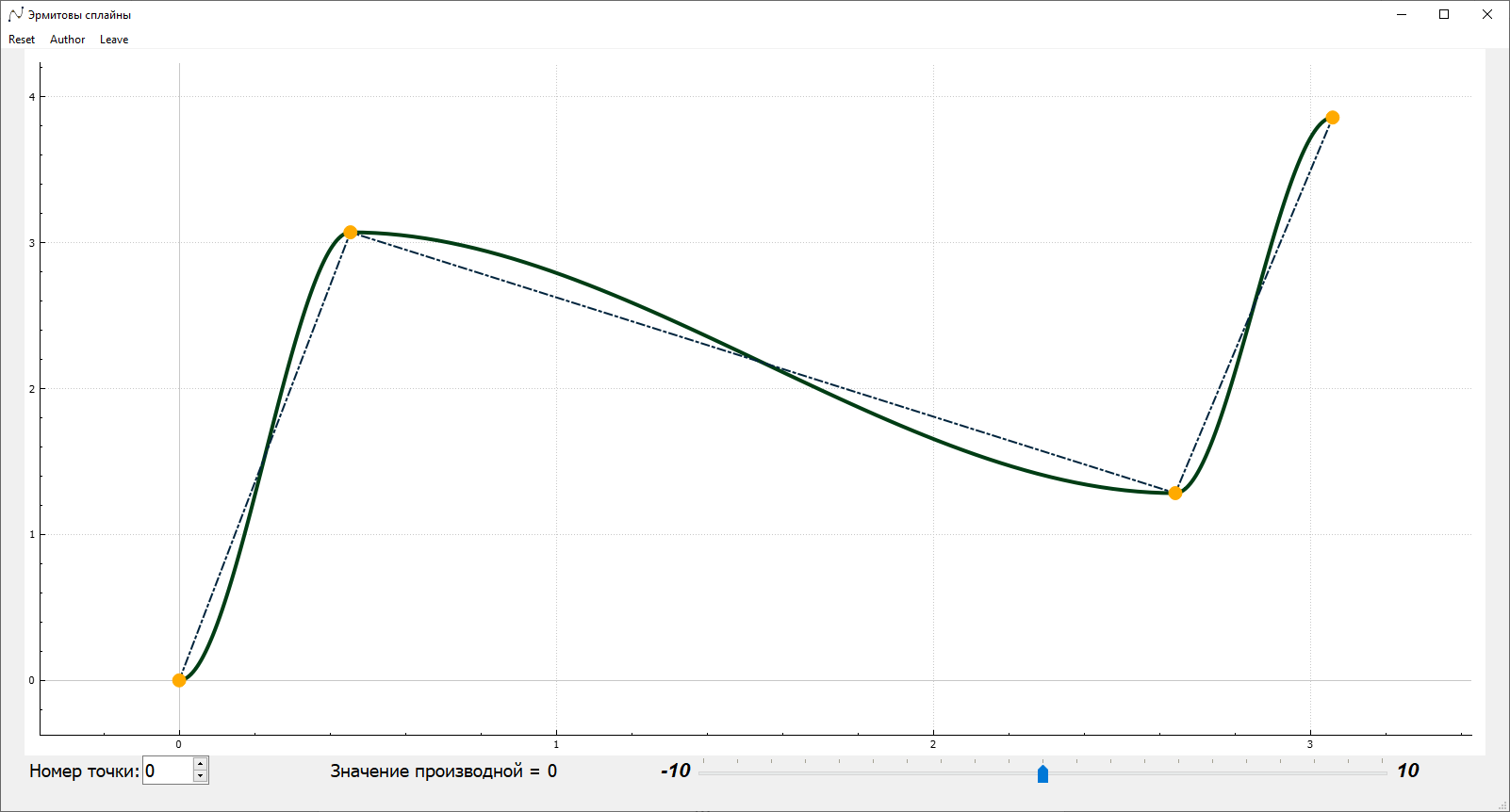
Изначальный вид рабочей области:



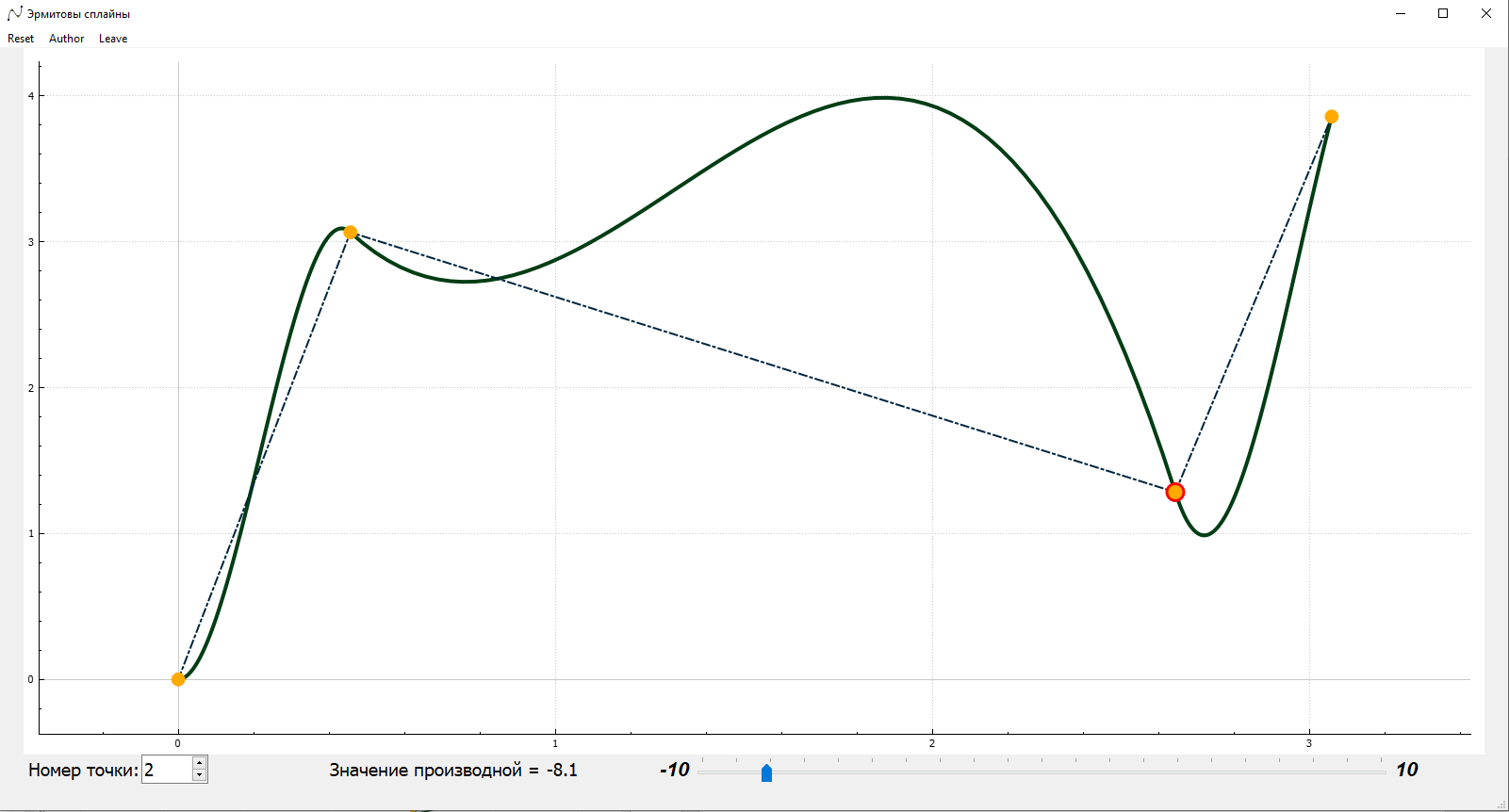
Добавляем несколько точек:



Передвигаем их:



Изменяем производные:



Ссылка на программу: <https://github.com/Zaburunier/MT_HW>