МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

(БГТУ им В. Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №2

Дисциплина: Компьютерная графика

Тема: «Построение графиков»

Выполнил: ст. группы ПВ-31

Донцов А.А.

Проверил: Осипов О. В.

Белгород, 2019

**Цель работы**

Разработка модуля для построения графиков функций одной переменной на декартовой плоскости в среде Qt Creator.

**Порядок выполнения работы**

1. Разработать визуальный компонент Chart для построения графиков функций одной переменной на декартовой плоскости.
2. Протестировать и отладить программу с различными функциями.

**Требования к программе**

1. В качестве входных данных пользователь вводит интервал по оси абсцисс 𝑥∈[𝑎,𝑏], на котором строится график. Программа при этом определяет автоматически, в каких пределах изменяются значения y.
2. Реализовать функцию step для разметки осей.
3. Реализовать классы, описанные в разделе 3.
4. Предусмотреть возможность построения нескольких функций на одном графике.
5. Программа должна при изменении размера окна пересчитывать координаты разметок и перестраивать график.
6. Числовые подписи к разметкам (линиям сетки) должны быть выравнены строго по центру разметки (как на рис. 2 и рис. 3).
7. Ограничить область значений функций большими константами ymax, ymin (для построения, например, гиперболы вблизи x=0).

**Chart.h**

#ifndef CHART\_H

#define CHART\_H

#include <QWidget>

#include <QtMath>

#include <QPainter>

#include <QResizeEvent>

class WorldToScreenConverter

{

public:

// Экранная система координат

QRect screen;

// Мировая система координат

QPointF point1, point2;

// Преобразование из мировой системы координат в экранную

QPoint convert(QPointF p);

// Преобразование X из экранной системы координат в мировую

qreal x(int X);

// Преобразование Y из экранной системы координат в мировую

qreal y(int Y);

// Коэффициенты в формуле перевода кординат

qreal k\_x, k\_y;

// Установить область определения функции [x1, x2]

void setIntervalX(qreal x1, qreal x2);

// Установить область определения функции [y1, y2]

void setIntervalY(qreal y1, qreal y2);

};

// Абстрактный класс для рисования осей и их разметки

class Axis

{

public:

// Вычисление шага разметки в экранной H и мировой k системах координат

void step(qreal& k, qreal& H, char& m, int& n);

// Рисование осей и разметки

virtual void *draw*(QPainter \*painter, WorldToScreenConverter \*coords\_converter, qreal H) = 0;

};

// Класс абсцисс

class AbscissaAxis : public Axis

{

public:

// Рисование осей и разметки оси абсцисс

void *draw*(QPainter \*painter, WorldToScreenConverter \*coords\_converter, qreal H);

};

// Класс ординат

class OrdinateAxis : public Axis

{

public:

// Рисование осей и разметки оси ординат

void *draw*(QPainter \*painter, WorldToScreenConverter \*coords\_converter, qreal H);

};

// Класс для хранения информации о выводимой функции

class CurveData

{

// Цвет кривой

QColor curveColor;

// Стиль пера

QPen curvePen;

public:

// Функция для рисования

virtual double *func*(qreal x) = 0;

QColor getCurveColor();

void setCurveColor(QColor color);

QPen getCurvePen();

void setCurvePen(QPen pen);

};

// Класс для рисования синуса

class SinCurveData : public CurveData

{

public:

qreal *func*(qreal x);

};

// Класс для рисования косинуса

class CosCurveData : public CurveData

{

public:

qreal *func*(qreal x);

};

// Класс для рисования тангенса

class TanCurveData : public CurveData

{

public:

qreal *func*(qreal x);

};

class Chart : public QWidget

{

Q\_OBJECT

public:

explicit Chart(QWidget \*parent = 0);

// Массив указателей на структуры, содержащие информацию о функциях f(x)

QVector<CurveData\*> curves;

// Функция устанавливает область определения графика функции

void setRangeX(double x1, double x2);

private:

// Объект для преобразования мировых координат в экранные и наоборот

WorldToScreenConverter coords\_converter;

void *paintEvent*(QPaintEvent \*);

void *resizeEvent*(QResizeEvent \*);

};

#endif // CHART\_H

**Chart.cpp**

#include "Chart.h"

QPoint WorldToScreenConverter::convert(QPointF p)

{

return QPoint(screen.left() + k\_x \* (p.x() - point1.x()), screen.bottom() - k\_y \* (p.y() - point1.y()));

}

qreal WorldToScreenConverter::x(int X)

{

return ((X - screen.left()) / k\_x + point1.x());

}

qreal WorldToScreenConverter::y(int Y)

{

return ((screen.bottom() - Y) / k\_y + point1.y());

}

void WorldToScreenConverter::setIntervalX(qreal x1, qreal x2)

{

point1.setX(x1);

point2.setX(x2);

}

void WorldToScreenConverter::setIntervalY(qreal y1, qreal y2)

{

point1.setY(y1);

point2.setY(y2);

}

void Axis::step(qreal &k, qreal &H, char &m, int &n)

{

//k = W / (b - a)

qreal p = H / k;

// Поиск степени

qreal n1 = log10(p),

n2 = log10(p / 2),

n5 = log10(p / 5);

// H = h \* k

{

double tmp1 = qPow(10, ceil(n1)) - p;

double tmp2 = p - qPow(10, floor(n1));

m = 1;

n = floor(n1);

if(tmp1 < tmp2)

{

m = 1;

n = ceil(n1);

tmp2 = tmp1;

}

tmp1 = p - 2 \* qPow(10, floor(n2));

if(tmp1 < tmp2)

{

m = 2;

n = floor(n2);

tmp2 = tmp1;

}

tmp1 = 2 \* qPow(10, ceil(n2)) - p;

if(tmp1 < tmp2)

{

m = 2;

n = ceil(n2);

tmp2 = tmp1;

}

tmp1 = p - 5 \* qPow(10, floor(n5));

if(tmp1 < tmp2)

{

m = 5;

n = floor(n5);

tmp2 = tmp1;

}

tmp1 = 5 \* qPow(10, ceil(n5)) - p;

if(tmp1 < tmp2)

{

m = 5;

n = ceil(n5);

}

H = k;

// k = m \* qPow(10, n) - действительный шаг разметки в мировой системе координат

k = m \* qPow(10, n);

H \*= k;

}

}

void AbscissaAxis::*draw*(QPainter \*painter, WorldToScreenConverter \*coords\_converter, qreal H)

{

// k - количество пикселей на одну единицу в мировой системе координат

qreal k = coords\_converter->screen.width() / (coords\_converter->point2.x() - coords\_converter->point1.x());

// k = m \* qPow(10, n) - действительный шаг разметки в мировой системе координат

char m = 0;

int n = 0;

// Вычисление шага разметки в экранной H и мировой k = m \* qPow(10, n) системах координат

this->step(k, H, m, n);

// Координаты первой разметки

qreal start = floor(coords\_converter->point1.x() / k + 1) \* k;

int j = 0;

// Ручка для рисование оси Ox

QPen penOx;

penOx.setWidth(2);

penOx.setColor(Qt::black);

// Рисование Ox

painter->setPen(penOx);

// Координаты левого края Ox

QPointF p1 = coords\_converter->convert(coords\_converter->point1);

p1.setX(p1.x() + penOx.width() + 40);

p1.setY(p1.y() - penOx.width() - 40);

// Координаты правого края Ox

QPointF p2 = coords\_converter->convert(QPointF(coords\_converter->point2.x(), coords\_converter->point1.y()));

p2.setX(p2.x() + penOx.width() + 40);

p2.setY(p2.y() - penOx.width() - 40);

// Рисования Ox

painter->drawLine(p1, p2);

// Рисование разметки и значений разметок

for (auto i = p1.x() + H; i < p2.x() - 0.5 \* H; i += H)

{

painter->drawLine(i, p1.y() - 4, i, p1.y() + 4);

painter->drawText(QRectF(QPointF(i - 30, p1.y() + 5), QPointF(i + 30, p1.y() + 25)), Qt::AlignCenter, QString::number(start + (j++) \* k));

}

}

void OrdinateAxis::*draw*(QPainter \*painter, WorldToScreenConverter \*coords\_converter, qreal H)

{

// k - количество пикселей на одну единицу в мировой системе координат

qreal k = coords\_converter->screen.height() / (coords\_converter->point2.y() - coords\_converter->point1.y());

// k = m \* qPow(10, n) - действительный шаг разметки в мировой системе координат

char m = 0;

int n = 0;

// Вычисление шага разметки в экранной H и мировой k = m \* qPow(10, n) системах координат

this->step(k, H, m, n);

// Координаты первой разметки

qreal start = floor(coords\_converter->point1.y() / k + 1) \* k;

int j = 0;

// Ручка для рисование оси Oy

QPen penOy;

penOy.setWidth(2);

penOy.setColor(Qt::black);

// Рисование Oy

painter->setPen(penOy);

// Координаты нижнего края Oy

QPointF p1 = coords\_converter->convert(coords\_converter->point1);

p1.setX(p1.x() + penOy.width() + 40);

p1.setY(p1.y() - penOy.width() - 40);

// Координаты верхнего края Oy

QPointF p2 = coords\_converter->convert(QPointF(coords\_converter->point1.x(), coords\_converter->point2.y()));

p2.setX(p2.x() + penOy.width() + 40);

p2.setY(p2.y() - penOy.width() - 40);

// Рисования Ox

painter->drawLine(p1, p2);

// Рисование разметки и значений разметок

for (auto i = p1.y() - H; i > p2.y() + 0.5 \* H; i -= H)

{

painter->drawLine(p1.x() - 4, i, p1.x() + 4, i);

painter->drawText(QRectF(QPointF(p1.x() - 45, i - 10), QPointF(p1.x() - 5, i + 10)), Qt::AlignCenter, QString::number(start + (j++) \* k));

}

}

QColor CurveData::getCurveColor()

{

return curveColor;

}

void CurveData::setCurveColor(QColor color)

{

curveColor = color;

}

QPen CurveData::getCurvePen()

{

return curvePen;

}

void CurveData::setCurvePen(QPen pen)

{

curvePen = pen;

}

qreal SinCurveData::*func*(qreal x)

{

return qSin(x);

}

qreal CosCurveData::*func*(qreal x)

{

return qCos(x);

}

qreal TanCurveData::*func*(qreal x)

{

return qTan(x);

}

Chart::Chart(QWidget \*parent) : QWidget(parent)

{

}

void Chart::*paintEvent*(QPaintEvent \*)

{

QPainter painter(this);

// Рисование всего, что связано с Ox

{

AbscissaAxis \_\_X;

qreal H = 100;

\_\_X.*draw*(&painter, &coords\_converter, H);

}

// Рисование всего, что связано с Oy

{

OrdinateAxis \_\_Y;

qreal H = 100;

\_\_Y.*draw*(&painter, &coords\_converter, H);

}

// Рисование графиков функций

{

QVector<CurveData\*> funcs;

funcs.push\_back(new SinCurveData);

funcs.push\_back(new CosCurveData);

funcs.push\_back(new TanCurveData);

for (int i = 0; i < funcs.size(); ++i)

{

funcs[i]->setCurveColor(QColor(0, ((i + 1) \* 255 / funcs.size()) & 255, ((funcs.size() - i) \* 255 / funcs.size()) & 255, 255));

QPen pen;

pen.setWidth(2);

pen.setColor(funcs[i]->getCurveColor());

painter.setPen(pen);

QPoint tmp1 = coords\_converter.convert(QPointF(coords\_converter.point1.x(), funcs[i]->*func*(coords\_converter.point1.x())));

tmp1.setX(tmp1.x() + pen.width() + 40);

tmp1.setY(tmp1.y() - pen.width() - 40);

QPoint tmp2;

for (auto j = coords\_converter.point1.x(); j <= coords\_converter.point2.x(); j += 0.01)

{

tmp2 = coords\_converter.convert(QPointF(j, funcs[i]->*func*(j)));

tmp2.setX(tmp2.x() + pen.width() + 40);

tmp2.setY(tmp2.y() - pen.width() - 40);

if ((tmp1.y() < coords\_converter.convert(coords\_converter.point1).y() - 40) &&

(tmp2.y() < coords\_converter.convert(coords\_converter.point1).y() - 40))

painter.drawLine(tmp1, tmp2);

tmp1 = tmp2;

}

}

}

if (QWidget::width() > QWidget::parentWidget()->width() - 60)

QWidget::resize(QWidget::parentWidget()->width() - 50, QWidget::height());

else if (QWidget::width() < QWidget::parentWidget()->width() - 90)

QWidget::resize(QWidget::parentWidget()->width() - 50, QWidget::height());

if (QWidget::height() > QWidget::parentWidget()->height())

QWidget::resize(QWidget::width(), QWidget::parentWidget()->height());

else if (QWidget::height() < QWidget::parentWidget()->height() + 1)

QWidget::resize(QWidget::width(), QWidget::parentWidget()->height());

}

void Chart::*resizeEvent*(QResizeEvent \*)

{

coords\_converter.screen = QWidget::geometry();

// Установка точек в мировой системе координат

coords\_converter.setIntervalX(-4, 4);

coords\_converter.setIntervalY(-2, 2);

coords\_converter.k\_x = (coords\_converter.screen.right() - coords\_converter.screen.left()) / (coords\_converter.point2.x() - coords\_converter.point1.x());

coords\_converter.k\_y = (coords\_converter.screen.bottom() - coords\_converter.screen.top()) / (coords\_converter.point2.y() - coords\_converter.point1.y());

}

**MainWindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**MainWindow.cpp**

#include "MainWindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~*MainWindow*()

{

delete ui;

}

**main.cpp**

#include "MainWindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QStringList paths = QCoreApplication::libraryPaths();

paths.append(".");

paths.append("platforms");

QCoreApplication::setLibraryPaths(paths);

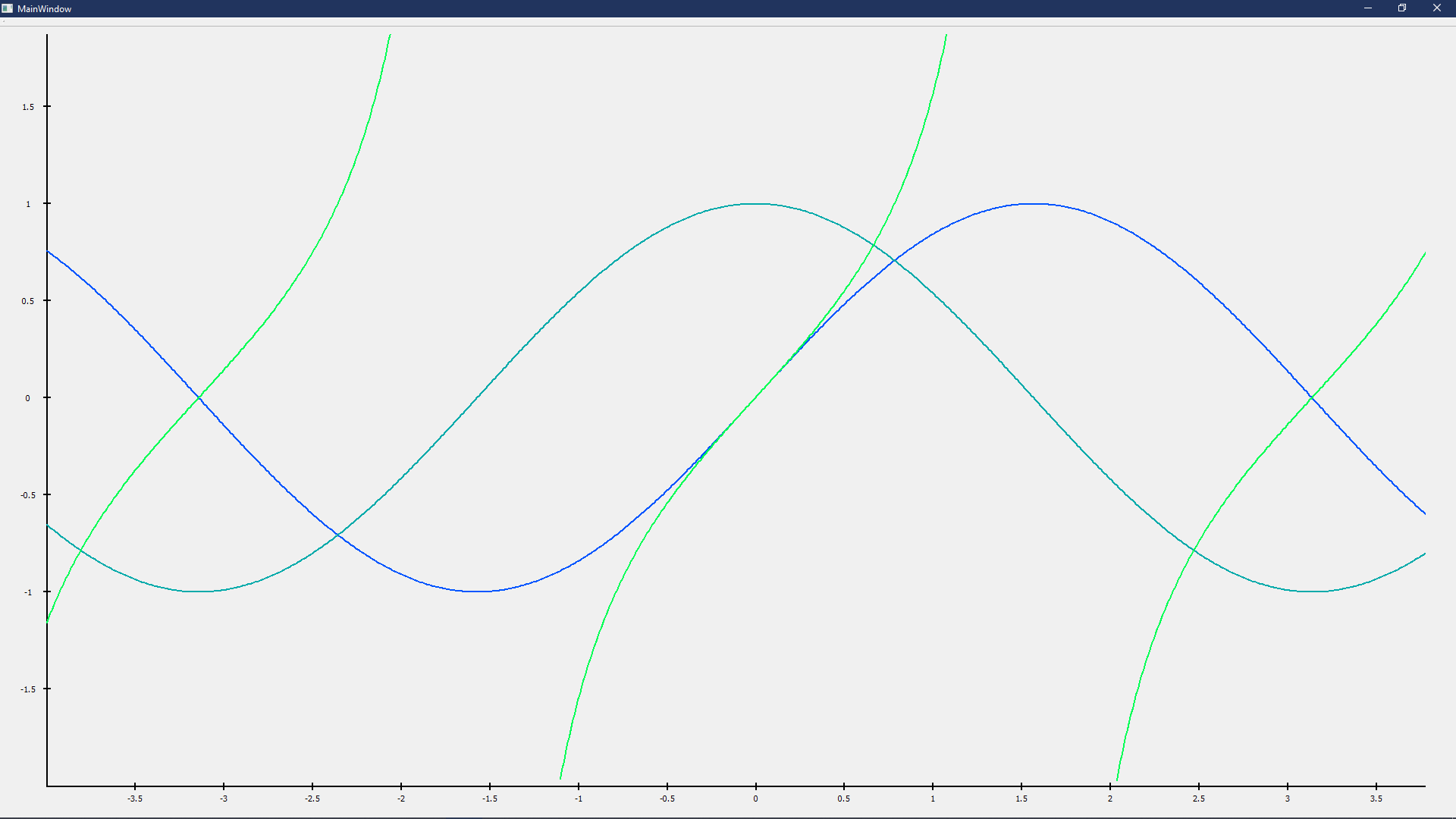
QApplication a(argc, argv);

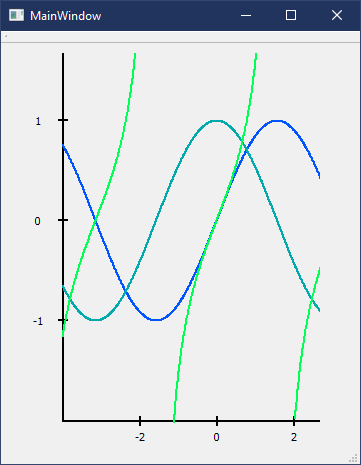
MainWindow w;

w.show();

return a.exec();

}





Вывод: выполнение данной лабораторной работы помогло изучить разработку модуля для построения графиков функций одной переменной на декартовой плоскости в среде Qt Creator.