Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа №1**

**по курсу «Компьютерная графика»**

|  |  |
| --- | --- |
| Студент: | Аксенов А.Е. |
| Группа: | М80-308Б-18 |
| Преподаватель: | Филиппов Г.С. |
| Оценка: |  |
| Дата: |  |

Москва

2020

**1. Постановка задачи.**

Написать и отладить программу, строящую изображение заданной замечательной кривой.

Вариант №16: , -a<A<=x<=B<a, где

x, y — декартовы координаты, a, A, B — константы, значения которых выбирается пользователем (вводится в окне программы).

Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

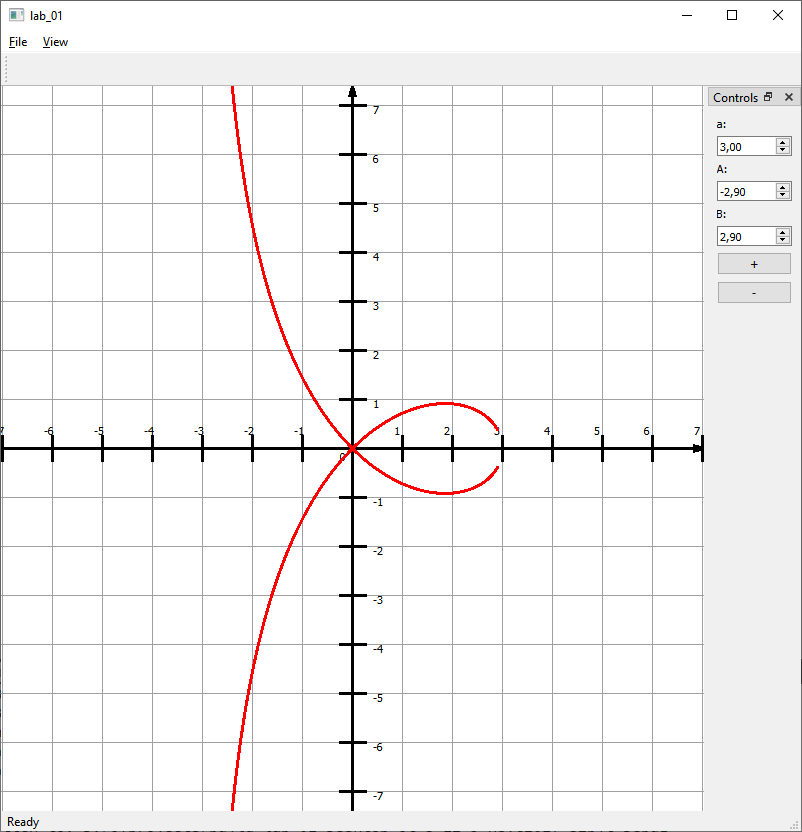
**2. Решение задачи.**

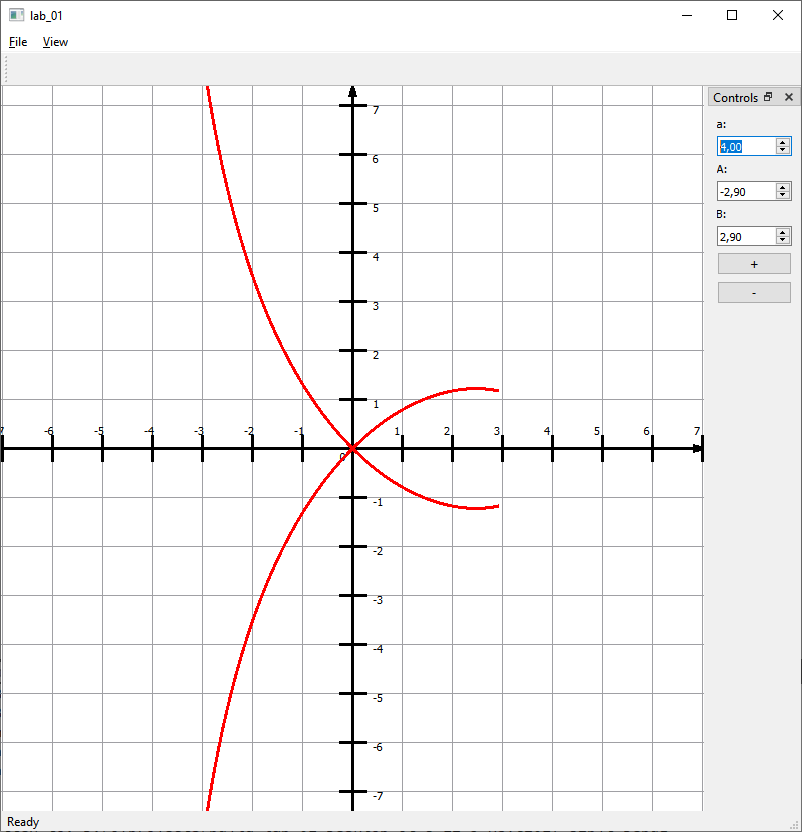
Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter для отрисовки точек и линий.

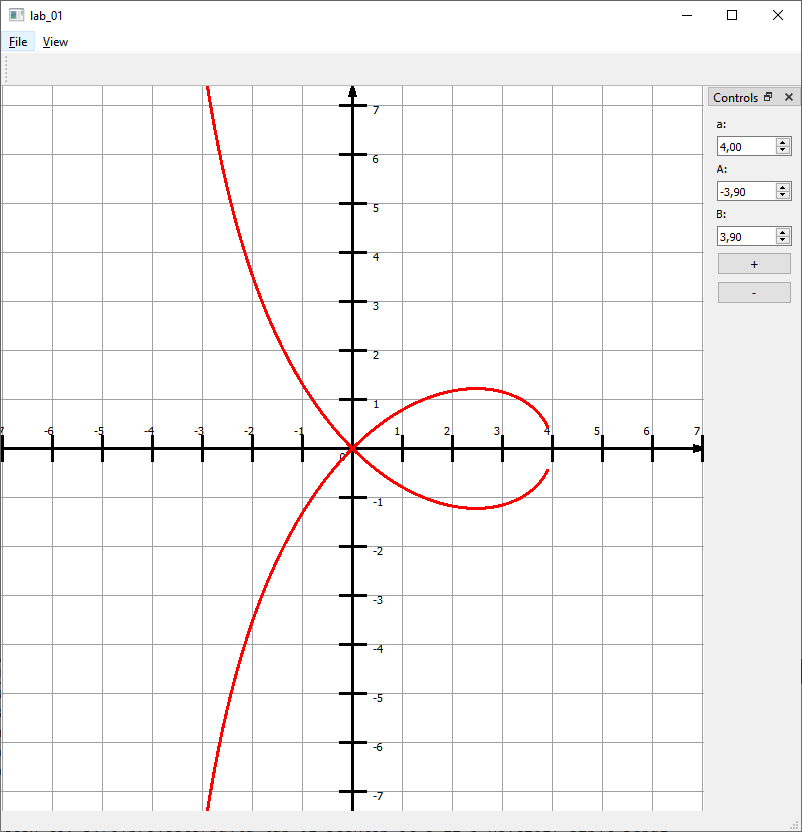
Я создал координатную плоскость, строю график, предлагаемый по умолчанию, со значениями констант: a = 3.0, A = - 2.9, B = 2.9.

При изменении размера окна вызывается функция resizeEvent(), которая вычисляет коэффициенты изменения ширины и высоты окна, а затем при учете этих коэффициентов график перерисовывается. Таким образом реализуется автоматическое масштабирование.

**3. Демонстрация работы программы.**







**4. Листинг программы.**

Здесь будут отображены только важные части кода.

// Функция, отвечающая за отрисовку системы координат и функции.

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

{

if (!pan)

return;

draw\_coords(this);

draw\_func(this);

}

// Функция, которая вызывается при изменении размера окна

void View::***resizeEvent***(QResizeEvent \*r\_event)

{

if (r\_event->oldSize().width() == -1 || r\_event->oldSize().height() == -1)

return;

double coef\_x = width() / static\_cast<double>(r\_event->oldSize().width());

double coef\_y = height() / static\_cast<double>(r\_event->oldSize().height());

if (step\_x \* coef\_x < 1 || step\_y \* coef\_y < 1) {

update();

return;

}

step\_x \*= coef\_x;

step\_y \*= coef\_y;

x\_center \*= coef\_x;

y\_center \*= coef\_y;

update();

}

// Функция, предназначенная для получения координаты курсора, при нажатии

// на кнопку мыши

void View::***mousePressEvent***(QMouseEvent \*event) {

previousPoint = event->pos();

}

// Функция, предназначенная для перемещения графика внутри окна

void View::***mouseMoveEvent***(QMouseEvent \*event) {

QPointF newPoint = event->pos();

double delta\_x = newPoint.x() - previousPoint.x();

double delta\_y = newPoint.y() - previousPoint.y();

x\_center += delta\_x;

y\_center += delta\_y;

previousPoint = newPoint;

update();

}

// Прорисовка системы координат

void **draw\_coords**(View \*v)

{

const int div\_x = v->set\_x;

const int div\_y = v->set\_y;

const double pi = atan(1) \* 4;

v->x\_center = v->width() / 2.0;

v->y\_center = v->height() / 2.0;

QPainter ptr{v};

ptr.setPen(QPen(Qt::black, 3));

//const QPointF center(v->width() / 2.0, v->height() / 2.0);

// drawing axis x

QPoint p1{0, static\_cast<int>(v->y\_center)};

QPoint p2{v->width(), static\_cast<int>(v->y\_center)};

const int arrow\_length = 10;

ptr.drawLine(p1, p2);

//axis x - arrow

QPointF p\_branch1{static\_cast<double>(v->width()), static\_cast<double>(v->y\_center)};

QPointF p\_branch2{arrow\_length \* cos(-11 \* pi / 12) + v->width(),

arrow\_length \* sin(-11 \* pi / 12) + v->y\_center};

ptr.drawLine(p\_branch1, p\_branch2);

p\_branch2 = {arrow\_length \* cos(11 \* pi / 12) + v->width(),

arrow\_length \* sin(11 \* pi / 12) + v->y\_center};

ptr.drawLine(p\_branch1, p\_branch2);

// drawing axis y

p1.setX(static\_cast<int>(v->x\_center));

p1.setY(0);

p2.setX(static\_cast<int>(v->x\_center));

p2.setY(v->height());

ptr.drawLine(p1, p2);

//axis y - arrow

p\_branch1 = {static\_cast<double>(v->x\_center), 0};

p\_branch2 = {arrow\_length \* cos(5 \* pi / 12) + v->x\_center,

arrow\_length \* sin(5 \* pi / 12)};

ptr.drawLine(p\_branch1, p\_branch2);

p\_branch2 = {arrow\_length \* cos(7 \* pi / 12) + v->x\_center,

arrow\_length \* sin(7 \* pi / 12)};

ptr.drawLine(p\_branch1, p\_branch2);

ptr.setPen(QPen(Qt::black, 3));

ptr.drawText(QPointF(v->x\_center - (v->step\_x / 4), v->y\_center + (v->step\_y / 4)),

QString::number(0));

//drawing grid

p\_branch1.setY(v->y\_center + v->step\_y / 4);

p\_branch2.setY(v->y\_center - v->step\_y / 4);

p1.setY(0);

p2.setY(v->height());

for (int x = static\_cast<int>(v->step\_x), num = 0; x + v->x\_center < v->width() || v->x\_center - x > 0;

x += v->step\_x, num += div\_x) {

//grid sticks - positive

ptr.setPen(Qt::gray);

p1.setX(static\_cast<int>(x + v->x\_center));

p2.setX(static\_cast<int>(x + v->x\_center));

ptr.drawLine(p1, p2);

//points - positive

ptr.setPen(QPen(Qt::black, 3));

p\_branch1.setX(x + v->x\_center);

p\_branch2.setX(x + v->x\_center);

ptr.drawLine(p\_branch1, p\_branch2);

ptr.drawText(QPointF(x + v->x\_center - v->step\_x / 6, static\_cast<int>(p\_branch1.y()) - v->step\_y / 2),

QString::number(num + div\_x));

//grid sticks - negative

ptr.setPen(Qt::gray);

p1.setX(static\_cast<int>(v->x\_center - x));

p2.setX(static\_cast<int>(v->x\_center - x));

ptr.drawLine(p1, p2);

//points - negative

ptr.setPen(QPen(Qt::black, 3));

p\_branch1.setX(v->x\_center - x);

p\_branch2.setX(v->x\_center - x);

ptr.drawLine(p\_branch1, p\_branch2);

ptr.drawText(QPointF(v->x\_center - x - v->step\_x / 6, static\_cast<int>(p\_branch1.y()) - v->step\_y / 2),

QString::number(-1 \* (num + div\_x)));

}

p\_branch1.setX(v->x\_center + v->step\_x / 4);

p\_branch2.setX(v->x\_center - v->step\_x / 4);

p1.setX(0);

p2.setX(v->width());

for (int y = static\_cast<int>(v->step\_y), num = 0; y + v->y\_center < v->height() || v->y\_center - y > 0;

y += v->step\_y, num += div\_y) {

//grid sticks - negative

ptr.setPen(Qt::gray);

p1.setY(static\_cast<int>(y + v->y\_center));

p2.setY(static\_cast<int>(y + v->y\_center));

ptr.drawLine(p1, p2);

//points - negative

ptr.setPen(QPen(Qt::black, 3));

p\_branch1.setY(y + v->y\_center);

p\_branch2.setY(y + v->y\_center);

ptr.drawLine(p\_branch1, p\_branch2);

ptr.drawText(QPointF(static\_cast<int>(p\_branch1.x()) + v->step\_x / 6, v->y\_center - y + v->step\_y / 6),

QString::number(num + div\_y));

//grid sticks - positive

ptr.setPen(Qt::gray);

p1.setY(static\_cast<int>(v->y\_center - y));

p2.setY(static\_cast<int>(v->y\_center - y));

ptr.drawLine(p1, p2);

//points - positive

ptr.setPen(QPen(Qt::black, 3));

p\_branch1.setY(v->y\_center - y);

p\_branch2.setY(v->y\_center - y);

ptr.drawLine(p\_branch1, p\_branch2);

ptr.drawText(QPointF(static\_cast<int>(p\_branch1.x()) + v->step\_x / 6, v->y\_center + y + v->step\_y / 6),

QString::number(-1 \* (num + div\_y)));

}

}

// Прорисовка графика функции

void **draw\_func**(View \*v) {

QPainter ptr{v};

ptr.setPen(QPen(Qt::red, 3));

//v->x\_center = v->width() / 2.0;

//v->y\_center = v->height() / 2.0;

const double step = 0.01;

QPointF p1{static\_cast<double>(v->x\_center), (v->y\_center)};

QPointF p2{};

QPointF p3{p1};

QPointF p4{};

QPointF p5{p1};

QPointF p6{};

QPointF p7{p1};

QPointF p8{};

for (double x = step; v->pan->set\_a() + x != 0 && v->pan->set\_B() < v->pan->set\_a() && v->pan->set\_A() > -(v->pan->set\_a()) && x <= v->pan->set\_B() && x >= v->pan->set\_A(); x += step) {

p2 = {v->x\_center + x \* v->step\_x/ v->set\_x, v->y\_center - sqrt((pow(x, 2) \* (v->pan->set\_a() - x ) / (v->pan->set\_a() + x))) \* v->step\_y / v->set\_y };

ptr.drawLine(p1, p2);

p1 = p2;

p4 = {v->x\_center + x \* v->step\_x/ v->set\_x, v->y\_center + sqrt((pow(x, 2) \* (v->pan->set\_a() - x ) / (v->pan->set\_a() + x))) \* v->step\_y / v->set\_y };

ptr.drawLine(p3, p4);

p3 = p4;

p6 = {v->x\_center - x \* v->step\_x/ v->set\_x, v->y\_center + sqrt((pow(x, 2) \* (v->pan->set\_a() + x ) / (v->pan->set\_a() - x))) \* v->step\_y / v->set\_y };

ptr.drawLine(p5, p6);

p5 = p6;

p8 = {v->x\_center - x \* v->step\_x/ v->set\_x, v->y\_center - sqrt((pow(x, 2) \* (v->pan->set\_a() + x ) / (v->pan->set\_a() - x))) \* v->step\_y / v->set\_y };

ptr.drawLine(p7, p8);

p7 = p8;

}

}

//Реализация боковой панели:

Panel::**Panel**(QWidget \*parent) : QWidget(parent)

{

QLabel\* lbl\_a(new QLabel("a:"));

a = new QDoubleSpinBox;

a->setRange(-1000, 1000);

a->setSingleStep(0.1);

a->setValue(3);

QLabel\* lbl\_A(new QLabel("A:"));

A = new QDoubleSpinBox;

A->setRange(-999.9, 999.9);

A->setSingleStep(0.1);

A->setValue(-2.9);

QLabel\* lbl\_B(new QLabel("B:"));

B = new QDoubleSpinBox;

B->setRange(-999.9, 999.9);

B->setSingleStep(0.1);

B->setValue(2.9);

inc = new QPushButton("+", this);

dec = new QPushButton("-", this);

QVBoxLayout\* lout(new QVBoxLayout);

lout->addWidget(lbl\_a);

lout->addWidget(a);

lout->addWidget(lbl\_A);

lout->addWidget(A);

lout->addWidget(lbl\_B);

lout->addWidget(B);

lout->addWidget(inc);

lout->addWidget(dec);

lout->addStretch();

setLayout(lout);

connect(a, SIGNAL(valueChanged(double)),

this, SIGNAL(a\_changed(double)));

connect(A, SIGNAL(valueChanged(double)),

this, SIGNAL(A\_changed(double)));

connect(B, SIGNAL(valueChanged(double)),

this, SIGNAL(B\_changed(double)));

connect(inc, SIGNAL(clicked(bool)),

this, SIGNAL(scale\_inc(bool)));

connect(dec, SIGNAL(clicked(bool)),

this, SIGNAL(scale\_dec(bool)));

}

double Panel::**set\_a**() const

{

return a->value();

}

double Panel::**set\_A**() const

{

return A->value();

}

double Panel::**set\_B**() const

{

return B->value();

}

**5. Вывод.**

В ходе выполнения данной лабораторной работы я познакомился с фреймворком Qt, нарисовал систему координат и график заданной функции. Знания, полученные в ходе выполнения работы, понадобятся мне, т.к. задачи визуализации данных встречаются довольно часто.