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: $y^2 = x^2(a - x)/(a + x)$, -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.

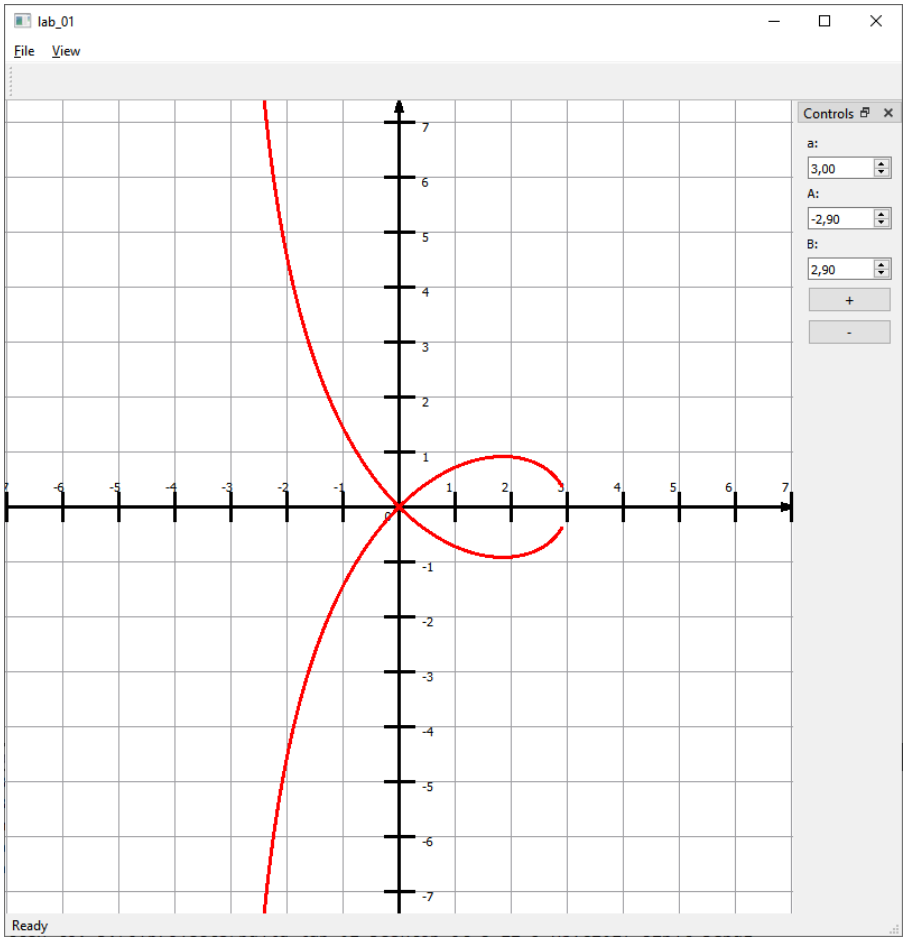
2. Решение задачи.

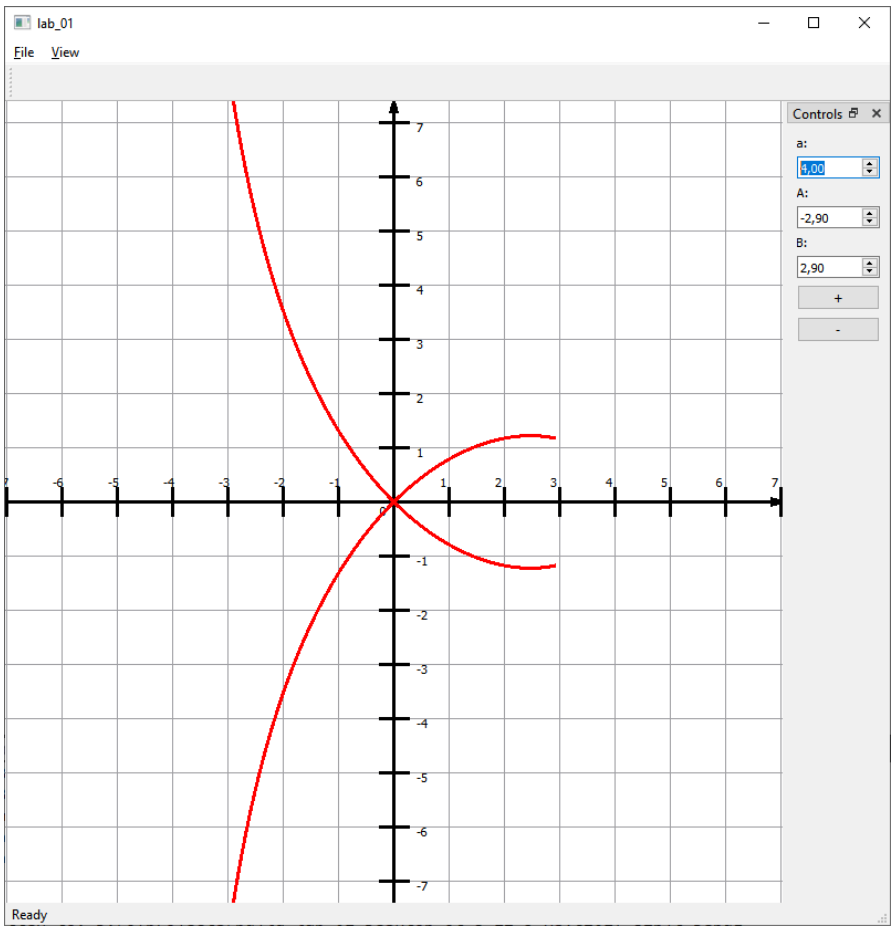
Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter для отрисовки точек и линий.

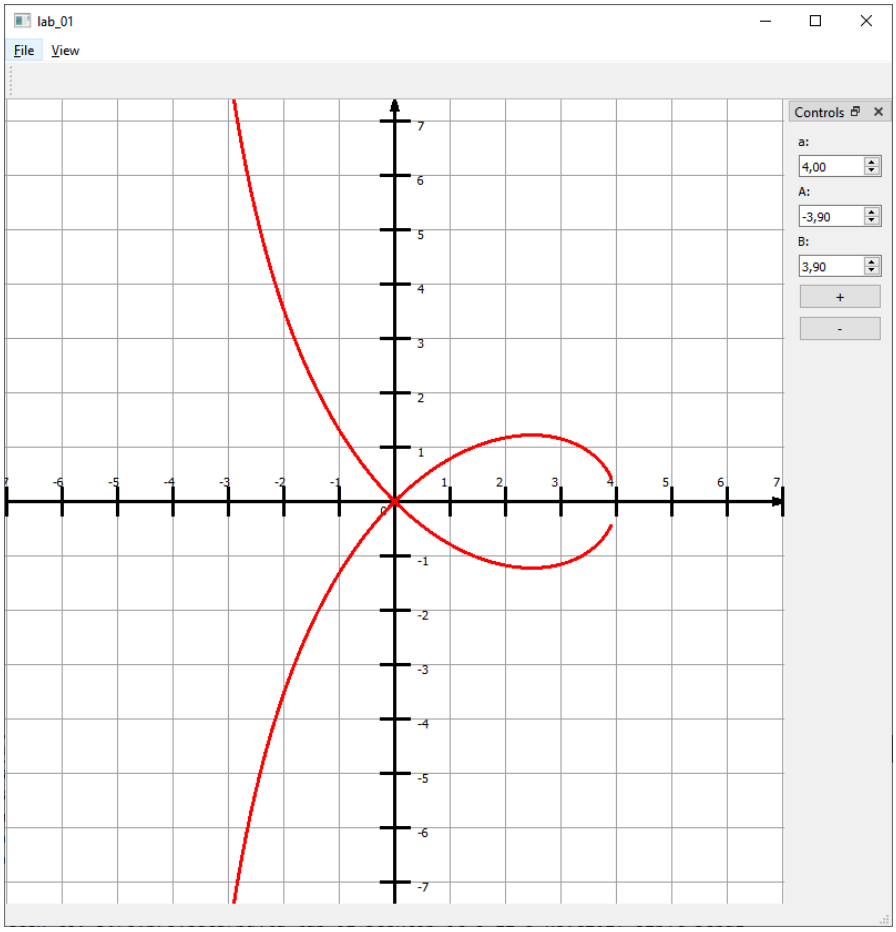
Я создал координатную плоскость, строю график, предлагаемый по умолчанию, со значениями констант: a = 3.0, A = - 2.9, B = 2.9.

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.




lab_01
File
View
Controls
a:
3,00
A:
-2,90
B:
2,90
+
-
Ready


lab_01
File
View
Controls
a:
4,00
A:
-3,90
B:
3,90
+
-

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->old-
Size().width());
    double coef_y = height() / static_cast<double>(r_event->old-
Size().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<dou-
ble>(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_cen-
ter < 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_cen-
ter < 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.