**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
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”

по теме “ Построение графиков функций”

Выполнил: ст. группы ПВ-31  
Ковалев Павел Александрович

Проверил: Осипов О.В.

**Белгород 2020**

**Цель работы:** разработка модуля для построения графиков функций одной переменной на декартовой плоскости в среде Qt Creator.

Порядок выполнения работы

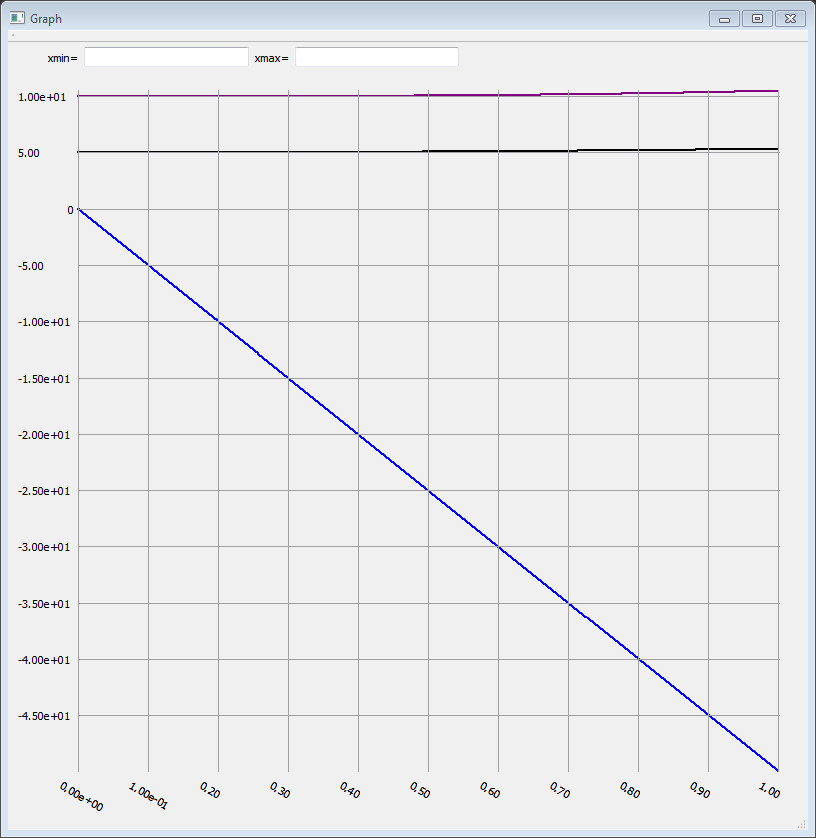
1. Разработать визуальный компонент Chart для построения графиков функций одной переменной на декартовой плоскости.
2. Протестировать и отладить программу с различными функциями.

Требования к программе

1. В качестве входных данных пользователь вводит интервал по оси абсцисс , 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).

Ограничить область значений функций большими константами *y*max, *y*min (для построения, например, гиперболы вблизи *x*=0).

Ход работы



Содержимое файла axis.h:

#ifndef AXIS\_H

#define AXIS\_H

#include "worldtoscreenconverter.h"

#include <QPainter>

*class* **Axis**

{

*public*:

void **step**(double \*k, int h, int \*m, int \*n);

void **draw**(QPainter \*painter, int h, WorldToScreenConverter conv);

};

*class* **AbscAxis** : Axis

{

*public*:

void **draw**(QPainter \*painter, int h, WorldToScreenConverter conv);

};

*class* **OrdAxis** : Axis

{

*public*:

void **draw**(QPainter \*painter, int h, WorldToScreenConverter conv);

};

#endif // AXIS\_H

Содержимое файла chart.h:

#ifndef CHART\_H

#define CHART\_H

#include <QColor>

#include <QPen>

#include <qpainter.h>

*class* **Chart**

{

*public*:

**Chart**();

~**Chart**();

QColor ChartColor;

QPen ChartPen;

*virtual* double ***F***(double x);

void **setColor**(QColor color);

};

*class* **Function1** : *public* Chart

{

*public*:

**Function1**();

*inline* double ***F***(double x);

};

*class* **Function2** : *public* Chart

{

*public*:

**Function2**();

*inline* double ***F***(double x);

};

*class* **Function3** : *public* Chart

{

*public*:

**Function3**();

*inline* double ***F***(double x);

};

#endif *//* *CHART\_H*

Содержимое файла graph.h:

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QMainWindow>

#include "chart.h"

*namespace* **Ui** {

*class* **Graph**;

}

*class* **Graph** : *public* QMainWindow

{

Q\_OBJECT

*public*:

*explicit* **Graph**(QWidget \*parent = 0);

~***Graph***();

double x\_min=0, x\_max=1, y\_min=0, y\_max=1;

QVector<Chart\*> charts;

void **calcYArea**();

void ***paintEvent***(QPaintEvent \*);

*private* slots:

void **on\_lineEdit\_editingFinished**();

void **on\_lineEdit\_2\_editingFinished**();

*private*:

Ui::Graph \*ui;

};

#endif *//* *GRAPH\_H*

Содержимое файла axis.cpp:

#include "axis.h"

#include "worldtoscreenconverter.h"

#include <math.h>

void Axis::**step**(double \*k, int h, int \*m, int \*n){

double h\_m = h/(\*k);

double n1 = log10(h\_m);

double n2 = log10(h\_m/2);

double n5 = log10(h\_m/5);

double min = pow(10, ceil(n1)) - h\_m;

\*k = min + h\_m;

\*m = 1;

\*n = ceil(n1);

double current = pow(10, floor(n1));

*if* (h\_m - current < min){*//floor* *-* *округление* *вниз*

min = h\_m - current;

\*k = current;

\*n = floor(n1);

}

current = 2 \* pow(10, ceil(n2));

*if* (current - h\_m < min){

min = current - h\_m;

\*k = current;

\*m = 2;

\*n = ceil(n2);

}

current = 2 \* pow(10, floor(n2));

*if* (h\_m - current < min){

min = h\_m - current;

\*k = current;

\*m = 2;

\*n = floor(n2);

}

current = 5 \* pow(10, ceil(n5));

*if* (current - h\_m < min){

min = current - h\_m;

\*k = current;

\*m = 5;

\*n = ceil(n5);

}

current = 5 \* pow(10, floor(n5));

*if* (h\_m - current < min){

min = h\_m - current;

\*k = current;

\*m = 5;

\*n = floor(n5);

}

}

void Axis::**draw**(QPainter \*painter, int h, WorldToScreenConverter conv){

QPen pen(Qt::*black*,3,Qt::*SolidLine*, Qt::*RoundCap*, Qt::*RoundJoin*);

painter->setPen(pen);

*if* (conv.get\_Xmin() <= 0 && 0 <= conv.get\_Xmax())

painter->drawLine(conv.convert(0,conv.get\_Ymin()), conv.convert(0, conv.get\_Ymax()));

*else* *if* (conv.get\_Xmin() < conv.get\_Xmax()){

*if* (conv.get\_Xmax() < 0)

painter->drawLine(conv.get\_BR(), conv.get\_TR());

*else*

painter->drawLine(conv.get\_BL(), conv.get\_TL());

}

*if* (conv.get\_Ymin() <= 0 && 0 <= conv.get\_Ymax()) *//ось* *Оx*

painter->drawLine(conv.convert(conv.get\_Xmin(), 0), conv.convert(conv.get\_Xmax(), 0));

*else* *if* (conv.get\_Ymin() < conv.get\_Ymax()){

*if* (conv.get\_Ymax() < 0)

painter->drawLine(conv.get\_TL(), conv.get\_TR());

*else*

painter->drawLine(conv.get\_BL(), conv.get\_BR());

}

painter->setPen(QPen(Qt::*blue*,1,Qt::*SolidLine*,Qt::*RoundCap*,Qt::*RoundJoin*));

}

void OrdAxis::**draw**(QPainter \*painter, int h, WorldToScreenConverter conv){

int m, n;

double k = conv.get\_h()/(conv.get\_Ymax() - conv.get\_Ymin());

step(&k, h, &m, &n);

double k1 = ceil(conv.get\_Ymin()/k)\*k;

*for* (; k1 <= conv.get\_Ymax(); k1 += k){

painter->setPen(Qt::*gray*);

painter->drawLine(conv.convert(conv.get\_Xmin(), k1), conv.convert(conv.get\_Xmax(), k1));

painter->setPen(Qt::*black*);

*if* (fabs(k1) - 1.e-14 < 0){

painter->drawText(QPoint(conv.get\_left()-10, conv.Yscr(k1)+5), "0");

}

*else*{

QString str;

*if*(fabs(k1) < 10)

str = QString::number(k1, 'f', 2);

*else*

str = QString::number(k1, 'e', 2);

painter->drawText(QPoint(conv.get\_left()-60, conv.Yscr(k1)+5), str);

}

}

painter->setPen(QPen(Qt::*blue*,1,Qt::*SolidLine*,Qt::*RoundCap*,Qt::*RoundJoin*));

}

void AbscAxis::**draw**(QPainter \*painter, int h, WorldToScreenConverter conv){

int m, n;

double k = conv.get\_w()/(conv.get\_Xmax() - conv.get\_Xmin());

step(&k, h, &m, &n);

double k1 = ceil(conv.get\_Xmin()/k)\*k;

*for* (; k1 <= conv.get\_Xmax(); k1 += k){

painter->setPen(Qt::*gray*);

painter->drawLine(conv.convert(k1, conv.get\_Ymin()), conv.convert(k1, conv.get\_Ymax()));

painter->setPen(Qt::*black*);

QString str;

*if*(fabs(k1) < 10 && fabs(k1) > 0.1)

str = QString::number(k1, 'f', 2);

*else*

str = QString::number(k1, 'e', 2);

QPoint p(conv.Xscr(k1)-20, conv.get\_bottom()+18);

painter->translate(p);

painter->rotate(30);

painter->drawText(0, 0, str);

painter->rotate(-30);

painter->translate(-p);

}

}

Содержимое файла chart.cpp:

#include "chart.h"

Chart :: **Chart**(){

QPen cPen(Qt::*blue*, 2, Qt::*SolidLine*, Qt::*RoundCap*, Qt::*RoundJoin*);

ChartPen = cPen;

}

double Chart :: ***F***(double x) { *return* 1; }

void Chart :: **setColor**(QColor color){

ChartColor = color;

ChartPen = QPen(color, 2, Qt::*SolidLine*, Qt::*RoundCap*, Qt::*RoundJoin*);

}

Function1 :: **Function1**(){}

double Function1 :: ***F***(double x) { *return* x\*x/3 + 5; }

Function2 :: **Function2**(){}

double Function2 :: ***F***(double x) { *return* -50\*x; }

Function3 :: **Function3**(){}

double Function3 :: ***F***(double x) { *return* x\*x\*x/2 + 10 ;}

Chart :: ~**Chart**(){

}

Содержимое файла graph.cpp:

#include "graph.h"

#include "ui\_graph.h"

#include <qpainter.h>

#include <QWidget>

#include <cfloat>

#include "axis.h"

Graph::**Graph**(QWidget \*parent) :

QMainWindow(parent),

ui(*new* Ui::Graph)

{

ui->setupUi(*this*);

Function1 \*f1 = *new* Function1();

Function2 \*f2 = *new* Function2();

Function3 \*f3 = *new* Function3();

f1->setColor(Qt::*black*);

f2->setColor(Qt::*blue*);

f3->setColor(Qt::*darkMagenta*);

charts.insert(0, f1);

charts.insert(1, f2);

charts.insert(2, f3);

calcYArea();

}

Graph::~***Graph***()

{

*delete* ui;

}

void Graph::**calcYArea**(){

double h = (x\_max - x\_min) / width();

double x = x\_min;

y\_min = charts[0]->*F*(x);

y\_max = y\_min;

QVector<Chart\*>::iterator it;

*for*(it = charts.begin(); it != charts.end(); it++){

x = x\_min;

*for* (; x <= x\_max; x+=h){

double y = (\*it)->*F*(x);

*if* (y\_min > y)

y\_min = y;

*if* (y\_max < y)

y\_max = y;

}

}

}

void Graph::***paintEvent***(QPaintEvent \*){

*//настройка* *рисователя*

QPainter paint(*this*);

QPoint minScreen(70, 60);

QPoint maxScreen(width() - 30, height() - 60);

WorldToScreenConverter converter(minScreen, maxScreen, x\_min, y\_min, x\_max, y\_max);

int h = 70;

*/\*построение* *графика\*/*

*//вычисление* *шага*

double h\_w = (x\_max - x\_min) / converter.get\_w();

double x = x\_min;

*//инициализация* *начальной* *точки*

QVector<Chart\*>::iterator it;

*for*(it = charts.begin(); it != charts.end(); it++) {

paint.setPen((\*it)->ChartPen);

x = x\_min;

QPoint p1 = converter.convert(x, (\*it)->*F*(x));

*for* (x = x + h\_w; x < x\_max; x += h\_w){

QPoint p2 = converter.convert(x, (\*it)->*F*(x));

paint.drawLine(p1, p2);

p1 = p2;

}

}

OrdAxis oy = OrdAxis();

oy.draw(&paint, h, converter); *//отрисовка* *сетки* *для* *оси* *Oy*

AbscAxis ox = AbscAxis();

ox.draw(&paint, h, converter); *//отрисовка* *сетки* *для* *оси* *Ox*

}

void Graph::**on\_lineEdit\_editingFinished**()

{

x\_min = ui->lineEdit->text().toDouble();

calcYArea();

*if* (x\_min < x\_max)

repaint();

}

void Graph::**on\_lineEdit\_2\_editingFinished**()

{

x\_max = ui->lineEdit\_2->text().toDouble();

calcYArea();

*if* (x\_min < x\_max)

repaint();

}

Содержимое файла worldtoscreenconverter.cpp:

#include "worldtoscreenconverter.h"

WorldToScreenConverter::**WorldToScreenConverter**(QPoint minScr, QPoint maxScr, double xMinWrld, double yMinWrld, double xMaxWrld, double yMaxWrld):

Screen(QRect(minScr, maxScr)),

xW\_min(xMinWrld),

yW\_min(yMinWrld),

xW\_max(xMaxWrld),

yW\_max(yMaxWrld)

{

xxx = Screen.width()/(xW\_max-xW\_min);

yyy = Screen.height()/(yW\_max-yW\_min);

xw = (xW\_max-xW\_min)/Screen.width();

yh = (yW\_max-yW\_min)/Screen.height();

}

QPoint WorldToScreenConverter::**convert**(double xWorld, double yWorld){

int x\_local = Screen.left() + (int)((xWorld-xW\_min)\*xxx);

int y\_local = 1 + Screen.bottom() - (int)((yWorld-yW\_min)\*yyy);

*return* QPoint(x\_local, y\_local);

}

int WorldToScreenConverter::**Xscr**(double xWorld){

*return* Screen.left() + (int)((xWorld-xW\_min)\*xxx);

}

int WorldToScreenConverter::**Yscr**(double yWorld){

*return* 1 + Screen.bottom() - (int)((yWorld-yW\_min)\*yyy);

}

double WorldToScreenConverter::**Xwrld**(int xScreen){

*return* ((xScreen-Screen.x())\*xw);

}

double WorldToScreenConverter::**Ywrld**(int yScreen){

*return* ((Screen.y()+Screen.height()-yScreen)\*yh);

}

int WorldToScreenConverter::**get\_w**(){

*return* Screen.width();

}

int WorldToScreenConverter::**get\_h**(){

*return* Screen.height();

}

QPoint WorldToScreenConverter::**get\_TL**(){

*return* Screen.topLeft();

}

QPoint WorldToScreenConverter::**get\_TR**(){

*return* Screen.topRight();

}

QPoint WorldToScreenConverter::**get\_BL**(){

*return* Screen.bottomLeft();

}

QPoint WorldToScreenConverter::**get\_BR**(){

*return* Screen.bottomRight();

}

double WorldToScreenConverter::**get\_Xmin**(){

*return* xW\_min;

}

double WorldToScreenConverter::**get\_Xmax**(){

*return* xW\_max;

}

double WorldToScreenConverter::**get\_Ymin**(){

*return* yW\_min;

}

double WorldToScreenConverter::**get\_Ymax**(){

*return* yW\_max;

}

int WorldToScreenConverter::**get\_left**(){

*return* Screen.left();

}

int WorldToScreenConverter::**get\_bottom**(){

*return* Screen.bottom();

}

Содержимое файла worldtoscreenconverter.h

#ifndef WORLDTOSCREENCONVERTER\_H

#define WORLDTOSCREENCONVERTER\_H

#include <QRect>

#include <QPoint>

*class* **WorldToScreenConverter**

{

*public*:

**WorldToScreenConverter**(QPoint minScr, QPoint maxScr, double xMinWrld, double yMinWrld, double xMaxWrld, double yMaxWrld);

QRect Screen;

double xW\_min, xW\_max, yW\_min, yW\_max;

double xxx, yyy, xw, yh;

QPoint **convert**(double xWorld, double yWorld);

int **Xscr**(double xWorld);

int **Yscr**(double yWorld);

double **Xwrld**(int xScreen);

double **Ywrld**(int yScreen);

*inline* int **get\_w**();

*inline* int **get\_h**();

*inline* QPoint **get\_TL**();

*inline* QPoint **get\_TR**();

*inline* QPoint **get\_BL**();

*inline* QPoint **get\_BR**();

*inline* double **get\_Xmin**();

*inline* double **get\_Xmax**();

*inline* double **get\_Ymin**();

*inline* double **get\_Ymax**();

*inline* int **get\_left**();

*inline* int **get\_bottom**();

};

#endif *//* *WORLDTOSCREENCONVERTER\_H*