**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 1**

Тема: Построение изображений 2D-кривых

Студент: Шиляева Н. С.

Группа: 80-304

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Написать и отладить программу, строящую изображение заданной замечательной кривой.

Вариант № 20. 

x,y – декартовы координаты.

a,B - константы, значения которых выбираются пользователем (вводятся в окне программы). a>0

Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

1. Решение задачи

Используемый язык программирования: C#.

В программе прописаны функции для параметров (их изменения) а, В; далее для шага, зума, поворота, сдвига, границ.

Далее производится отрисовка осей графика, штрихов и чисел на осях, отрисовка точек графика.

Потом идёт функция вычисления точек графика, вычисление границ картинки.

Используемые библиотеки и точка входа приложения:

#region Директивы using (подключаемые библиотеки) и точка входа приложения

using Device = CGLabPlatform.GDIDevice;

using DeviceArgs = CGLabPlatform.GDIDeviceUpdateArgs;

using System;

using CGLabPlatform;

using CGApplication = AppMain;

using System.Drawing;

using System.Collections.Generic;

using System.Linq;

using Microsoft.SqlServer.Server;

using System.Globalization;

using System.Windows.Forms;

public abstract class AppMain : CGApplicationTemplate<Application, Device, DeviceArgs>

{

[STAThread]

static void Main()

{

RunApplication();

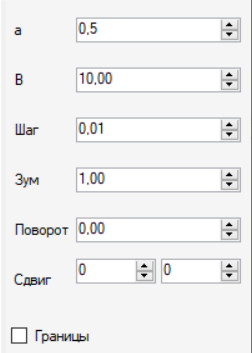
}

}

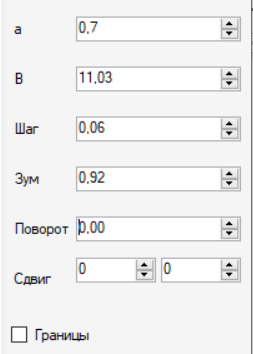
#endregion

1. Набор тестов

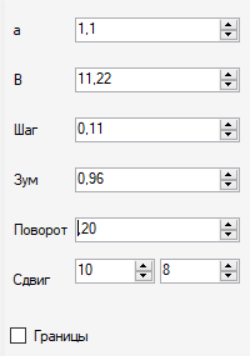
Тест 1:



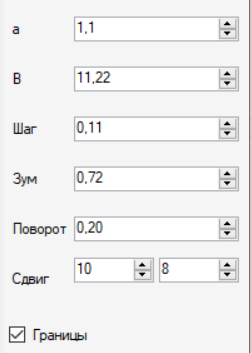
Тест 2:



Тест 3:

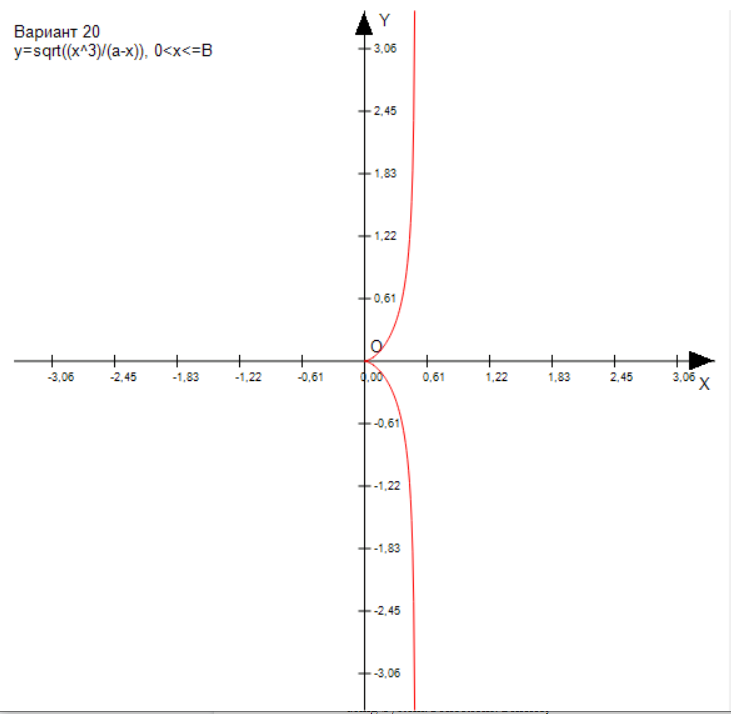


Тест 4:

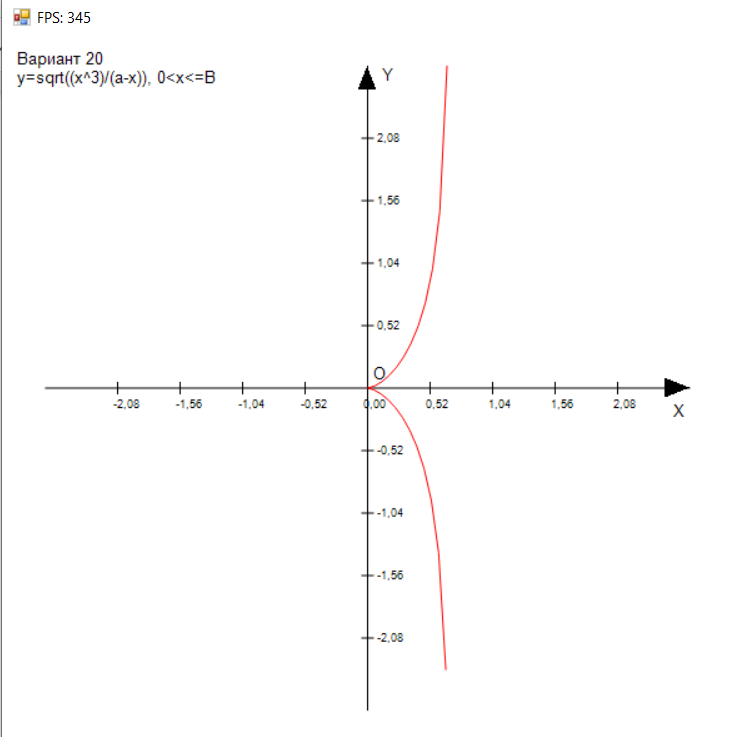


1. Результаты выполнения тестов

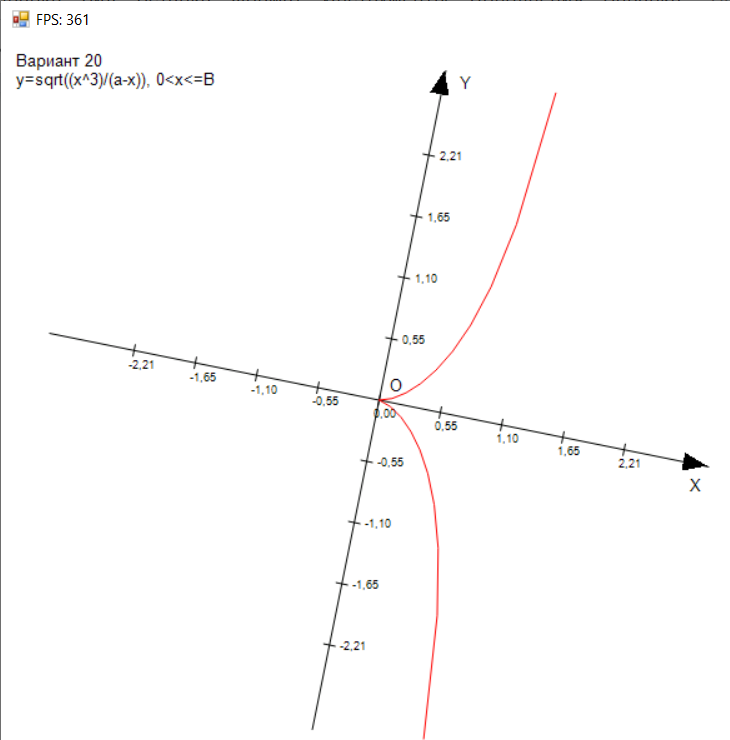
Тест 1:



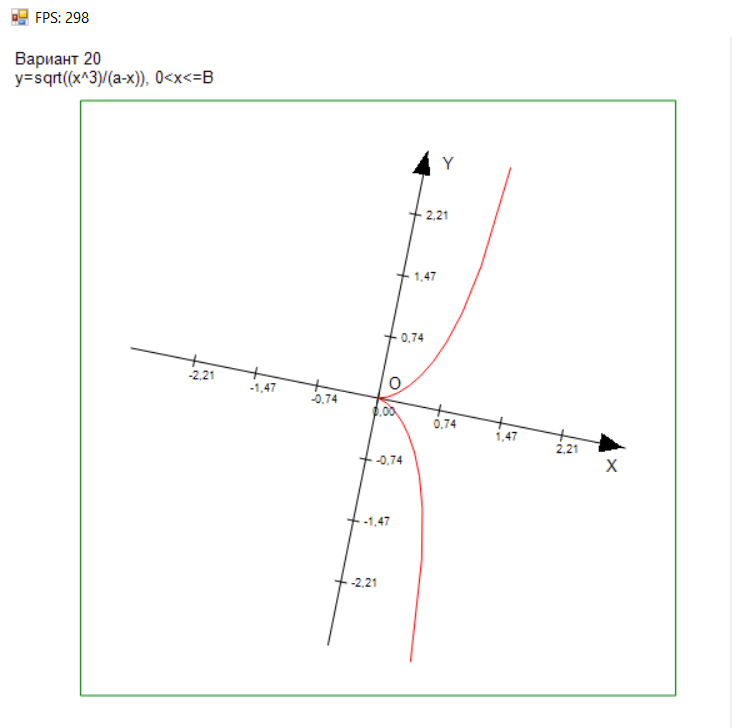
Тест 2:



Тест 3:



Тест 4:



1. Листинг программы

Код программы:

#region Директивы using (подключаемые библиотеки) и точка входа приложения

using Device = CGLabPlatform.GDIDevice;

using DeviceArgs = CGLabPlatform.GDIDeviceUpdateArgs;

using System;

using CGLabPlatform;

using CGApplication = AppMain;

using System.Drawing;

using System.Collections.Generic;

using System.Linq;

using Microsoft.SqlServer.Server;

using System.Globalization;

using System.Windows.Forms;

public abstract class AppMain : CGApplicationTemplate<Application, Device, DeviceArgs>

{

[STAThread]

static void Main()

{

RunApplication();

}

}

#endregion

public abstract class Application : CGApplication

{

#region элементы GUI

[DisplayNumericProperty(0.5, 0.1, "a", 0.1, 100)]

public virtual double a

{

get => Get<double>();

set

{

Set(value);

CalculatePoints();

}

}

[DisplayNumericProperty(10, 0.01, "B", 0.01, 15)]

public virtual double B

{

get => Get<double>();

set

{

Set(value);

CalculatePoints();

}

}

[DisplayNumericProperty(0.01, 0.01, "Шаг", 0.01, 10)]

public virtual double step

{

get => Get<double>();

set

{

Set(value);

CalculatePoints();

}

}

[DisplayNumericProperty(1, 0.01, "Зум", 0.1, 500)]

public virtual double zoom

{

get => Get<double>();

set

{

Set(value);

CalculateBounds();

}

}

[DisplayNumericProperty(0, 0.01, "Поворот", -2 \* Math.PI, 2 \* Math.PI)]

public virtual double angle

{

get => Get<double>();

set

{

var new\_value = value;

while (new\_value >= 2 \* Math.PI)

{

new\_value -= 2 \* Math.PI;

}

while (new\_value <= -2 \* Math.PI)

{

new\_value += 2 \* Math.PI;

}

Set(new\_value);

CalculateBounds();

}

}

[DisplayNumericProperty(new[] { 0d, 0d }, 1, "Сдвиг", -1000, 1000)]

public virtual DVector2 Shift

{

get => Get<DVector2>();

set

{

Set(value);

CalculateBounds();

}

}

[DisplayCheckerProperty(false, "Границы")]

public virtual bool DrawBounds { get; set; }

#endregion

DVector2 centerPoint; // центр экрана

List<DVector2> points; // точки графика

DVector2? prevLocation = null; // нужно в обработчике движения мышки с зажатой ПКМ

double axisLen; // длина оси

double fitMultiplier; // множитель масштаба, рассчитывающийся динамически в зависимости от размеров окна

// левая, правая, нижняя, верхняя границы картинки

double left\_bound, right\_bound, lower\_bound, upper\_bound;

protected override void OnMainWindowLoad(object sender, EventArgs args)

{

RenderDevice.BufferBackCol = 0xFF; // белый цвет фона

centerPoint = new DVector2(RenderDevice.Width, RenderDevice.Height) / 2;

fitMultiplier = Math.Min(RenderDevice.Width, RenderDevice.Height) / (axisLen \* 2);

// изменение масштаба колёсиком мыши

RenderDevice.MouseWheel += (\_, e) => zoom += e.Delta \* 0.001;

// двигание графика с зажатой ЛКМ

RenderDevice.MouseMoveWithLeftBtnDown += (\_, e) => Shift += new DVector2(e.MovDeltaX, e.MovDeltaY);

points = new List<DVector2>(); // список точек графика

CalculatePoints();

CalculateBounds();

// событие изменения размеров рабочей области

RenderDevice.SizeChanged += (\_, e) =>

{

centerPoint = new DVector2(RenderDevice.Width, RenderDevice.Height) / 2;

fitMultiplier = Math.Min(RenderDevice.Width, RenderDevice.Height) / (axisLen \* 2);

CalculateBounds();

// сдвиг картинки, если её границы выходят за границы рабочей области

if (left\_bound < 0 && Shift.X <= 0 && Shift.X <= left\_bound)

{

Shift += new DVector2(-left\_bound, 0);

}

if (right\_bound > RenderDevice.Width && Shift.X >= 0 && Shift.X >= right\_bound - RenderDevice.Width)

{

Shift -= new DVector2(right\_bound - RenderDevice.Width, 0);

}

if (upper\_bound < 0 && Shift.Y <= 0 && Shift.Y <= upper\_bound)

{

Shift += new DVector2(0, -upper\_bound);

}

if (lower\_bound > RenderDevice.Height && Shift.Y >= 0 && Shift.Y >= lower\_bound - RenderDevice.Height)

{

Shift -= new DVector2(0, lower\_bound - RenderDevice.Height);

}

};

// поворот графика с зажатой ПКМ

RenderDevice.MouseMoveWithRightBtnDown += (\_, e) =>

{

// сохранение текущего положения как прошлого, если ПКМ была зажата только что

if (prevLocation == null)

{

prevLocation = new DVector2(e.PressedLocation);

}

var b = centerPoint + Shift - new DVector2(e.Location); // вектор из центра картинки в место, где сейчас курсор

var c = centerPoint + Shift - (DVector2)prevLocation; // вектор из центра картинки в место, где курсор был прошлый раз

var cos = c.DotProduct(b) / (b.GetLength() \* c.GetLength()); // косинус угла поворота

var sin = c.CrossProduct(b) / (b.GetLength() \* c.GetLength()); // синус угла поворота

prevLocation = new DVector2(e.Location); // сохранение текущего положения курсора как прошлого

angle += Math.Atan2(sin, cos); // вычисление угла поворота по синусу и косинусу

// вычитание или прибавление 2\*pi во избежание переполнения переменной

while (angle > 2 \* Math.PI)

{

angle -= 2 \* Math.PI;

}

while (angle < 2 \* -Math.PI)

{

angle += 2 \* Math.PI;

}

CalculateBounds();

};

// обнуление прошлого положения, если ПКМ отжата

RenderDevice.MouseUp += (\_, e) => prevLocation = null;

}

protected override void OnDeviceUpdate(object s, DeviceArgs e)

{

if (points == null || points.Count == 0) return;

#region Рисование осей

var x\_head = new DVector2(axisLen, 0); // начало оси OX

var x\_tail = new DVector2(-axisLen, 0); // конец оси OX

var y\_head = new DVector2(0, axisLen); // начало оси OY

var y\_tail = new DVector2(0, -axisLen); // конец оси OY

e.Surface.DrawLine(0, ToUserScreen(x\_head), ToUserScreen(x\_tail)); // отрисовка оси OX

e.Surface.DrawLine(0, ToUserScreen(y\_head), ToUserScreen(y\_tail)); // отрисовка оси OY

var wholeScale = fitMultiplier \* zoom;

// стрелка оси OY

e.Surface.DrawTriangle(0,

ToUserScreen(y\_head),

ToUserScreen(y\_head + new DVector2(8, -20) / wholeScale),

ToUserScreen(y\_head + new DVector2(-8, -20) / wholeScale));

// стрелка оси OX

e.Surface.DrawTriangle(0,

ToUserScreen(x\_head),

ToUserScreen(x\_head + new DVector2(-20, 8) / wholeScale),

ToUserScreen(x\_head + new DVector2(-20, -8) / wholeScale));

// подпись оси OY

var (text\_x, text\_y) = ToUserScreen(y\_head + new DVector2(10, 0) / wholeScale);

e.Graphics.DrawString("Y", new Font("Arial", 10f), Brushes.Black, text\_x, text\_y);

// подпись оси OX

(text\_x, text\_y) = ToUserScreen(x\_head + new DVector2(-15, -10) / wholeScale);

e.Graphics.DrawString("X", new Font("Arial", 10f), Brushes.Black, text\_x, text\_y);

// подпись точки O

(text\_x, text\_y) = ToUserScreen(new DVector2(3, 20) / wholeScale);

e.Graphics.DrawString("O", new Font("Arial", 10f), Brushes.Black, text\_x, text\_y);

#endregion

#region Штрихи и числа на осях

for (double x = 0; x < x\_head.X - 30 / wholeScale; x += 50 / wholeScale)

{

e.Surface.DrawLine(0, ToUserScreen(new DVector2(x, 5 / wholeScale)), ToUserScreen(new DVector2(x, -5 / wholeScale)));

(text\_x, text\_y) = ToUserScreen((x, 0).ToDVector2() - (5, 7).ToDVector2() / wholeScale);

e.Graphics.DrawString($"{x:F2}", new Font("Arial", 7), Brushes.Black, text\_x, text\_y);

}

for (double x = 0; x > x\_tail.X + 25 / wholeScale; x -= 50 / wholeScale)

{

if (x == 0) continue;

e.Surface.DrawLine(0, ToUserScreen(new DVector2(x, 5 / wholeScale)), ToUserScreen(new DVector2(x, -5 / wholeScale)));

(text\_x, text\_y) = ToUserScreen((x, 0).ToDVector2() - (5, 7).ToDVector2() / wholeScale);

e.Graphics.DrawString($"{x:F2}", new Font("Arial", 7), Brushes.Black, text\_x, text\_y);

}

for (double y = 0; y < y\_head.Y - 30 / wholeScale; y += 50 / wholeScale)

{

if (y == 0) continue;

e.Surface.DrawLine(0, ToUserScreen(new DVector2(5 / wholeScale, y)), ToUserScreen(new DVector2(-5 / wholeScale, y)));

(text\_x, text\_y) = ToUserScreen((0, y).ToDVector2() + (6, 6).ToDVector2() / wholeScale);

e.Graphics.DrawString($"{y:F2}", new Font("Arial", 7), Brushes.Black, text\_x, text\_y);

}

for (double y = 0; y > y\_tail.Y + 25 / wholeScale; y -= 50 / wholeScale)

{

if (y == 0) continue;

e.Surface.DrawLine(0, ToUserScreen(new DVector2(5 / wholeScale, y)), ToUserScreen(new DVector2(-5 / wholeScale, y)));

(text\_x, text\_y) = ToUserScreen((0, y).ToDVector2() + (6, 6).ToDVector2() / wholeScale);

e.Graphics.DrawString($"{y:F2}", new Font("Arial", 7), Brushes.Black, text\_x, text\_y);

}

#endregion

// отрисовка точек графика

var previousPoint = points[0];

for (int i = 1; i < points.Count; ++i)

{

var currentPoint = points[i];

e.Surface.DrawLine(Color.Red.ToArgb(), ToUserScreen(previousPoint), ToUserScreen(currentPoint));

previousPoint = currentPoint;

}

// номер варианта и задание

e.Graphics.DrawString("Вариант 20\ny=sqrt((x^3)/(a-x)), 0<x<=B", new Font("Arial", 10f), Brushes.Black, 10f, 10f);

// отрисовка границ, если стоит флаг

if (DrawBounds)

{

e.Surface.DrawLine(Color.Green.ToArgb(), new DVector2(left\_bound, lower\_bound), new DVector2(right\_bound, lower\_bound));

e.Surface.DrawLine(Color.Green.ToArgb(), new DVector2(left\_bound, upper\_bound), new DVector2(right\_bound, upper\_bound));

e.Surface.DrawLine(Color.Green.ToArgb(), new DVector2(right\_bound, upper\_bound), new DVector2(right\_bound, lower\_bound));

e.Surface.DrawLine(Color.Green.ToArgb(), new DVector2(left\_bound, upper\_bound), new DVector2(left\_bound, lower\_bound));

}

}

// вычисление точек графика

void CalculatePoints()

{

if (points == null) return;

lock (RenderDevice.LockObj) // чтобы не изменить список во время обращения к нему из другого потока

{

points.Clear();

for (double x = B; x >= 0; x -= step)

{

if (a - x == 0) continue;

if (Math.Pow(x, 3) / (a - x) < 0) continue;

points.Add(new DVector2(x, -Math.Sqrt(Math.Pow(x, 3) / (a - x))));

}

for (double x = 0; x <= B; x += step)

{

if (a - x == 0) continue;

if (Math.Pow(x, 3) / (a - x) < 0) continue;

points.Add(new DVector2(x, Math.Sqrt(Math.Pow(x, 3) / (a - x))));

}

// вычисление длины оси и корректировка масштаба, чтобы сохранить исходный размер картинки

axisLen = Math.Max(points.Max(p => Math.Abs(p.X)), points.Max(p => Math.Abs(p.Y)));

fitMultiplier = Math.Min(RenderDevice.Width, RenderDevice.Height) / (axisLen \* 2);

}

CalculateBounds();

}

// вычисление границ картинки

void CalculateBounds()

{

if (axisLen == 0) return;

var corners = new List<DVector2>()

{

ToUserScreen(new DVector2(-axisLen, axisLen)),

ToUserScreen(new DVector2(axisLen, axisLen)),

ToUserScreen(new DVector2(-axisLen, -axisLen)),

ToUserScreen(new DVector2(axisLen, -axisLen))

};

var max\_horizontal = corners.Max(c => Math.Abs(c.X));

var max\_vertical = corners.Max(c => Math.Abs(c.Y));

left\_bound = -max\_horizontal + 2 \* (centerPoint.X + Shift.X);

right\_bound = max\_horizontal;

upper\_bound = -max\_vertical + 2 \* (centerPoint.Y + Shift.Y);

lower\_bound = max\_vertical;

}

// перевод вектора в экранные координаты

DVector2 ToScreenSpace(DVector2 vector)

{

return vector.Multiply((1, -1).ToDVector2()) \* fitMultiplier + centerPoint;

}

// приведение вектора в вид, в котором он будет выводиться на экран пользователю

DVector2 ToUserScreen(DVector2 vector)

{

var (x, y) = ToScreenSpace(vector) - centerPoint;

var rotated = new DVector2(

x \* Math.Cos(angle) - y \* Math.Sin(angle),

x \* Math.Sin(angle) + y \* Math.Cos(angle));

return rotated \* zoom + centerPoint + Shift;

}

}

// расширение класса DVector2

public static class DVectorExtensions

{

// косое произведение с одним аргументом

public static double CrossProduct(this DVector2 this\_vec, DVector2 vector)

{

return this\_vec.X \* vector.Y - vector.X \* this\_vec.Y;

}

// косое произведение с двумя аргументами

public static double CrossProduct(this DVector2 \_, DVector2 left, DVector2 right)

{

return left.X \* right.Y - right.X \* left.Y;

}

// деконструкция в отдельные переменные: var (x, y) = vector

public static void Deconstruct(this DVector2 vector, out double x, out double y)

{

x = vector.X;

y = vector.Y;

}

// преобразование любого кортежа из двух элементов в DVector2: (x, y).ToDvector2()

public static DVector2 ToDVector2<T, U>(this (T, U) tuple) where T : struct, IConvertible

where U : struct, IConvertible

{

return new DVector2(tuple.Item1.ToDouble(CultureInfo.CurrentCulture),

tuple.Item2.ToDouble(CultureInfo.CurrentCulture));

}

}

1. Вывод

Успешно построено изображение 2D - кривой. Написана и отлажена программа, строящая изображение заданной замечательной кривой. Обеспечено автоматическое масштабирование и центрирование кривой при изменении размеров окна.

Список литературы

1. Шилдт, Герберт. Ш57 С# 4.0: полное руководство. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 1056 с.: ил. — Парал. тит. англ. (дата обращения: 21.09.2020).
2. Прайс. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов, 3-е издание. -М.: Питер, 2018 - 640 с. (дата обращения: 20.09.2020).