|  |
| --- |
|  |

Факультет компьютерных и фундаментальных наук

Кафедра программного обеспечения вычислительной техники  
 и автоматизированных систем

ОТЧЕТ

о прохождении практики

студента группы ПО(аб)-81 Пшеничного Д.О.

Направление обучения 09.03.04 Программная инженерия

Наименование практики: Учебная практика: практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

Сроки прохождения практики: 09.02.20-13.06.20

Место прохождения практики: Тихоокеанский государственный университет,   
 кафедра программного обеспечения ВТ и АС

Руководитель практики от кафедры: доцент каф. ПОВТАС Вихтенко Э.М.

Руководитель практики от профильной организации:   
и.о.завкафедрой ПОВТАС Син А.З.

Оценка за практику:

Руководитель практики

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

Хабаровск 2020 г.

**Содержание**

[Введение 3](#_Toc43847816)

[Постановка задачи 4](#_Toc43847817)

[Задание на практику 5](#_Toc43847818)

[Метод Рунге-Кутты 6](#_Toc43847819)

[Программное обеспечение 8](#_Toc43847820)

[Результаты тестирования 9](#_Toc43847821)

[Результаты решения задачи 13](#_Toc43847822)

[Заключение 17](#_Toc43847823)

[Список используемой литературы 18](#_Toc43847824)

[Приложение А 19](#_Toc43847825)

### Введение

В ходе практической работы нам необходимо выполнить следующие пункты:

1) Написать программу решения задачи Коши для произвольной системы ОДУ заданным методом Рунге – Кутты.

2) Решить тестовую задачу с целью отладки написанной программы и экспериментального подтверждения теоретического порядка точности реализованного метода Рунге – Кутты.

3) Решить основную задачу — заданную задачу Коши, содержащую несколько параметров. Изучить влияние одного из параметров на качественные и количественные свойства решения.

В первом пункте задания дается краткое описание предмета моделирования, основных предположений и законов, положенных в основу модели. Этот пункт написан для того, чтобы дать некоторые представления о том, каким примерно образом была получена соответствующая математическая модель. Как правило, исходная модель записывается в терминах размерных зависимых и независимых переменных. Простым масштабированием (с математической точки зрения) от размерных переменных и уравнений осуществляется переход к безразмерным переменным и уравнениям (основной задаче), которые и решаются.

Стоит отметить, что числовые данные основных задач имеют математический смысл и подобраны так, чтобы обеспечить необходимые свойства решений.

### Постановка задачи

Клетки опухоли содержат особые вещества (антигены), которые вызывают резкую иммунную реакцию у больного. Эта реакция состоит в том, что производятся клетки — лимфоциты, которые атакуют и уничтожают клетки опухоли. Модель оперирует следующими переменными (имеется в виду плотность названной популяции клеток):

• L — свободные лимфоциты на поверхности опухоли;

• C — опухолевые клетки внутри опухоли и на ее поверхности;

• CS — опухолевые клетки на поверхности опухоли;

• CN — опухолевые клетки на поверхности опухоли, не связанные лимфоцитами;

• CF — опухолевые клетки внутри и на поверхности опухоли, не связанные лимфоцитами;

Ясно, что C = CF − CN + CS. Предполагается, что опухоль всегда имеет форму шара, так что CS = K1C2/3 , где K1 — постоянная, и что взаимодействие опухолевых клеток с лимфоцитами происходит только на поверхности опухоли. Будем считать, что между количеством свободных и связанных лимфоцитами клеток опухоли выполняется соотношение CS − CN = K2CNL (правдоподобно ли это предположение?). Из приведенных соотношений следует, что

CF = C − K1K2LC2/3/(1 + K2L), CN = K1C2/3/(1 + K2L), т. е. переменные L и C можно взять за основные переменные модели, которая сводится к системе уравнений

L′ = (−λ1 + α1CN (1 − L/LM))L,

C′ = λ2CF − α2CN L.

Здесь λ1 характеризует уровень естественной смертности лимфоцитов, следующее слагаемое — их стимуляцию: когда L мало, стимуляция свободных лимфоцитов возрастает линейно с ростом CN и что существует максимальный размер популяции LM, при котором стимуляция обращается в нуль. Первое слагаемое во втором уравнении описывает рост опухоли, не подвергающейся атакам лимфоцитов, а второй член учитывает взаимодействие свободных лимфоцитов с опухолевыми клетками на поверхности опухоли. Подставляя в уравнения значения CN и CF , можно переписать их в виде

x′ = ( −λ1 + β1y 2/3 (1 − x/c)/(1 + x) ) x,

y′ = λ2y − β2xy2/3/(1 + x), (1)

где x = K2L,c = K2LM, y = K1C, а λ1, λ2, β1, β2 — положительные параметры. Так как x и y — размеры популяций, они должны быть неотрицательными, а x не может превышать c, поскольку L ограничено сверху величиной LM. Уравнения (1) дополняются начальными условиями

x(0) = x0, y(0) = y0. (2)

### Задание на практику

1. Напишите программу интегрирования задачи Коши для системы из *n* уравнений первого порядка вида

, , ,

на произвольном отрезке *[a, b],* используя метод Рунге-Кутты третьего порядка точности с постоянным шагом *h*:

,

1. Протестируйте программу на примере системы уравнений

На отрезке [0, 5] c точным решением

*, .*

1. Для тестовой задачи постройте графики зависимости максимальной погрешности решения и от выбранного шага .
2. Для значений параметра β2 = 3; 3.48; 5 при помощи разработанной процедуры рассчитайте динамику популяции при различных начальных значениях размера опухоли y0 ∈ [0.5, 9]. Постройте графики наиболее характерных решений в координатах (x, y) и дайте их интерпретацию. Параметры: λ1 = λ2 = 1, β1 = 1, c = 3, t ∈ [0, 20].

### Метод Рунге-Кутты

Наиболее эффективными и часто встречаемыми методами решениями задачи Коши являются методы Рунге - Кутта. Они основаны на аппроксимации искомой функции *у(х)* в пределах каждого шага многочленом, который получен при помощи разложения функции *у(х)* в окрестности шага *h* каждой *i*-ой точки в ряд Тейлора:

*u( + h) = u() + hu′() + u′′( + h), ∈ [0,1].*

Пусть требуется найти приближенное решение задачи *u′(t) = f(t,u(t)), t ∈ (a,b], u(a)=*, на равномерной с шагом h сетке узлов.

Поскольку в силу уравнения справедливо равенство *u′() = f(,u()),* то соотношение (5) перепишется в виде

*= + hf(,) + h,*

*= ,*

*u′′( + h).*

В этой формуле слагаемое = O() является малой величиной, если *h* достаточно мало. Отбрасывая его, придем к методу Эйлера*: = + hf(,),* *i* = 0,1,...,N −1, = . Эти соотношения позволяют вычислить приближенное решение в точке сетки , зная приближенное решение в предыдущей точке . Такие численные методы называются одношаговыми.

Методы Рунге – Кутты при *q* = 3: Формулы имеют вид

yi+1 = yi + h (b1k1 + b2k2 + b3k3),

где

k1 = f(ti, yi),

k2 = f(ti + c2h, yi + a21hk1),

k3 = f(ti + c3h, yi + a31hk1 + a32hk2).

Аналогично тому, как это было сделано выше, получается следующая система из шести уравнений для определения восьми коэффициентов метода при m = 3:

c2 = a21,

c3 = a31 + a32,

b1 + b2 + b3 = 1,

2(c2b2 + c3b3) = 1,

,

.

Эта система имеет два семейства решений: двухпараметрическое со свободными параметрами c2 и c3, причем, c2 c3 и c2 2/3, и однопараметрическое со свободным параметром a32 (при c2 = c3 = 2/3). В качестве примера укажем коэффициенты при c2 = 1/2, c3 = 1: b1 = b3 = 1/6, b2 = 4/6, a21 = 1/2, a31 = −1, a32 = 2.

В задаче с моим вариантом требуется написать программу, используя метод Рунге – Кутты 3-го порядка точности с постоянным шагом h:

,

### Программное обеспечение

Программа была написана на языке С# в интегрированной среде разработки Visual Studio, так как программа содержит визуальные компоненты, при желании она может быть запущена на любом компьютере под управлением Windows и с установленной платформой .NET.

Для реализации программы было написано несколько классов и их методов, код которых можно увидеть в приложении А.

### Результаты тестирования

В ходе решения тестовой задачи мы задаем отрезок [a,b] и шаг h.

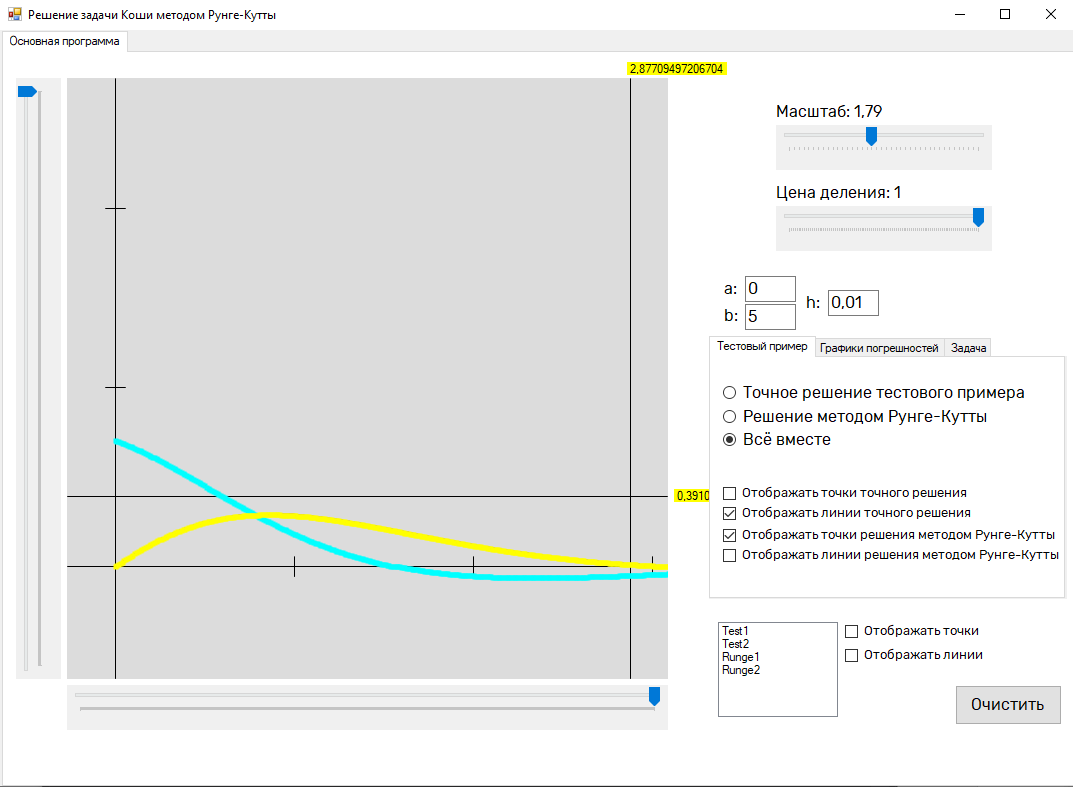


Рис. 1 – Решение тестовой задачи с шагом h=0.01

В ходе решения тестовой задачи мы задаем отрезок [a,b] и шаг h.

При шаге h=0.01 погрешность вычислений достаточно мала. Это видно по точкам, которые лежат на графиках точного решения задачи (рис. 1).

Однако если увеличить шаг сетки h в 5 (рис. 2) или в 10 (рис. 3) раз, то погрешность вычислений возрастает. Некоторые точки уже не лежат на графиках функций.

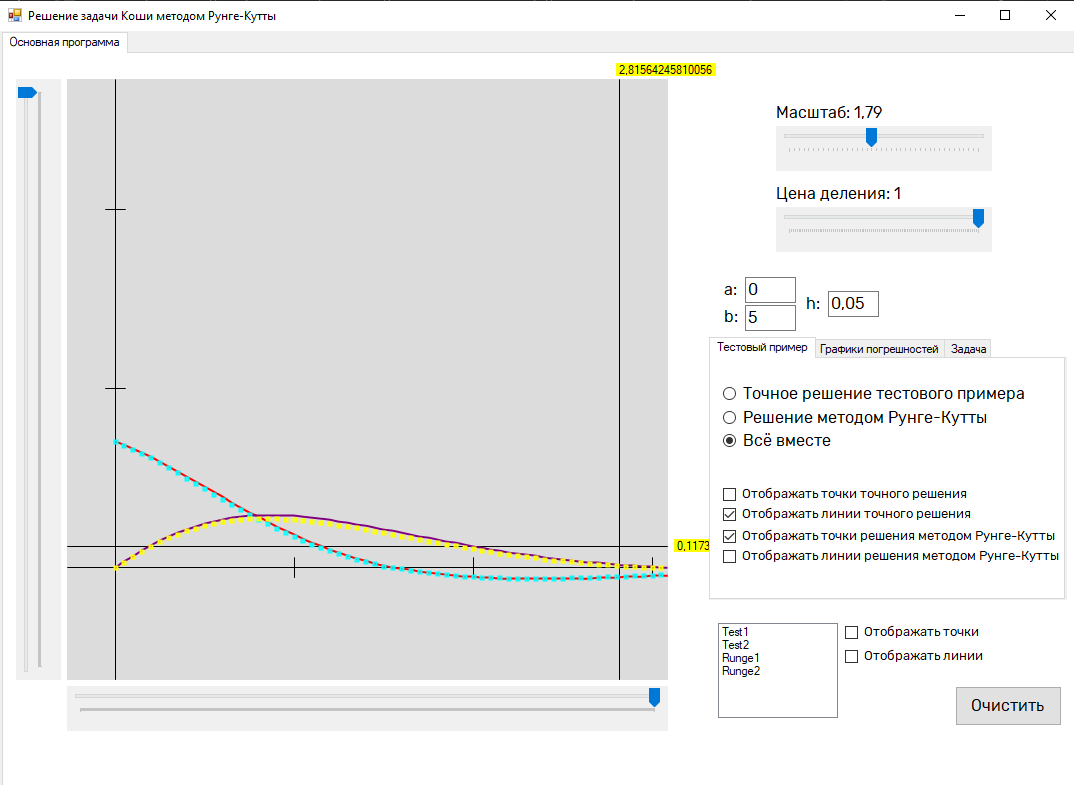


Рис. 2 – Решение тестовой задачи с шагом h=0.05

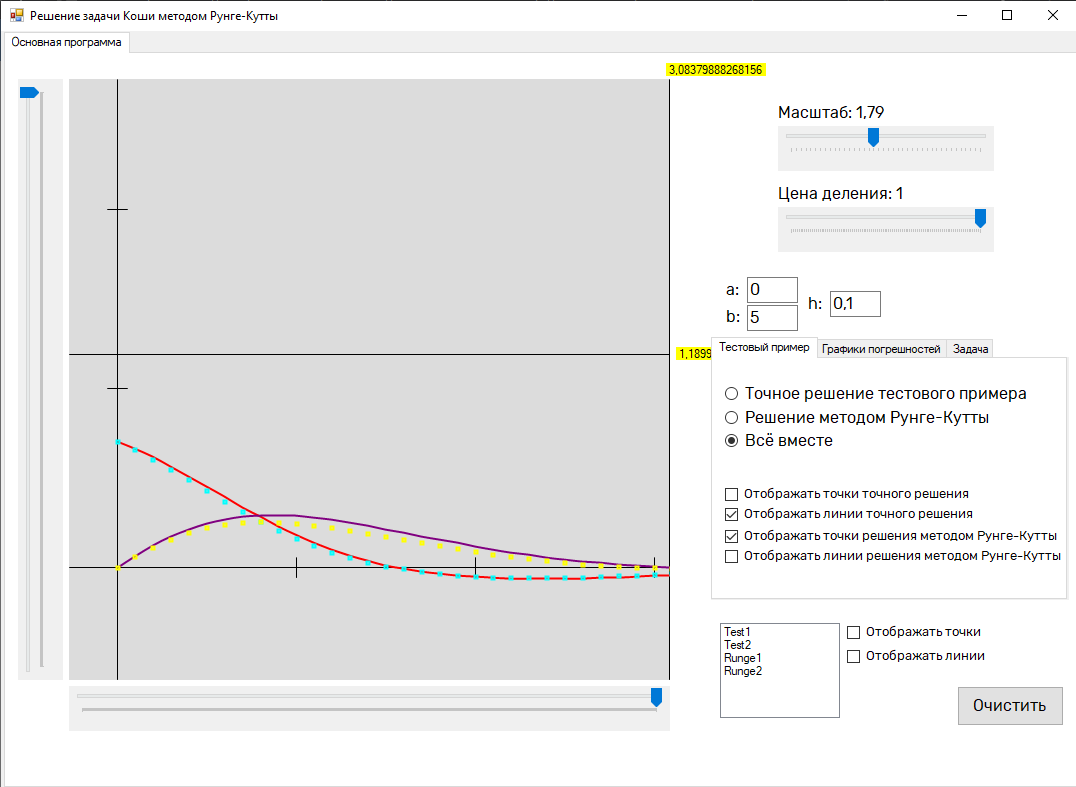


Рис. 3 – Решение тестовой задачи с шагом h=0.1

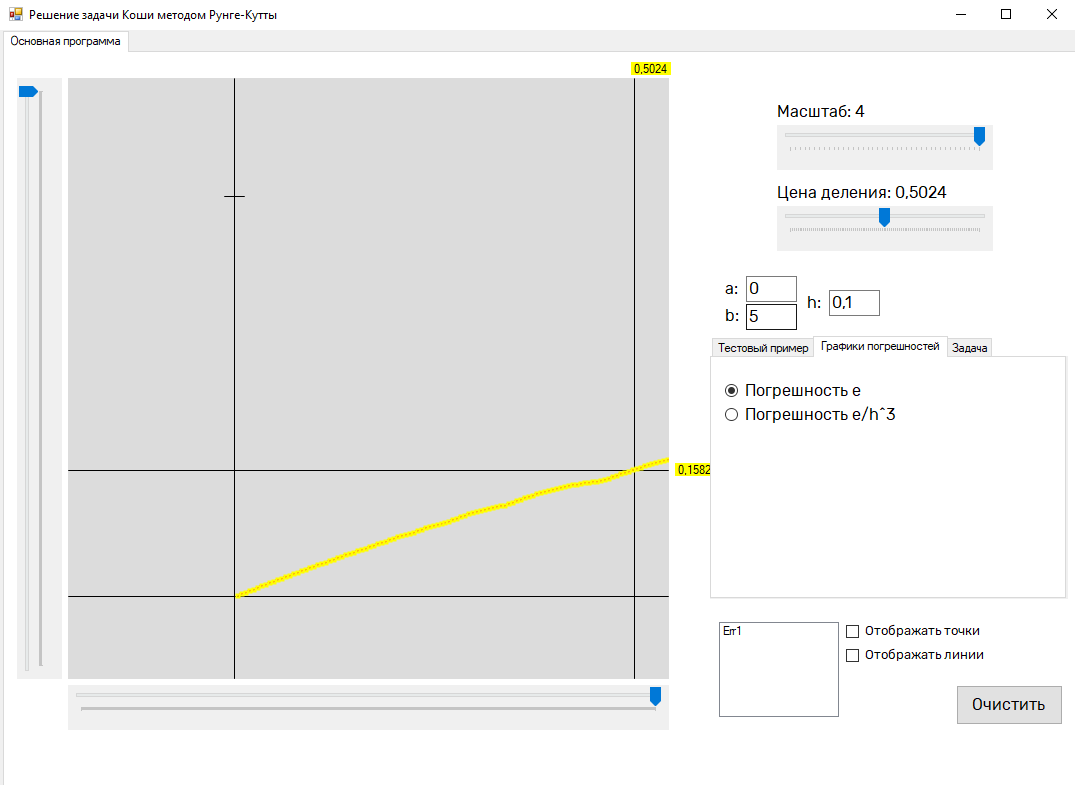


Рис.4 – График зависимости максимальной погрешности решения *e* от выбранного шага *h*

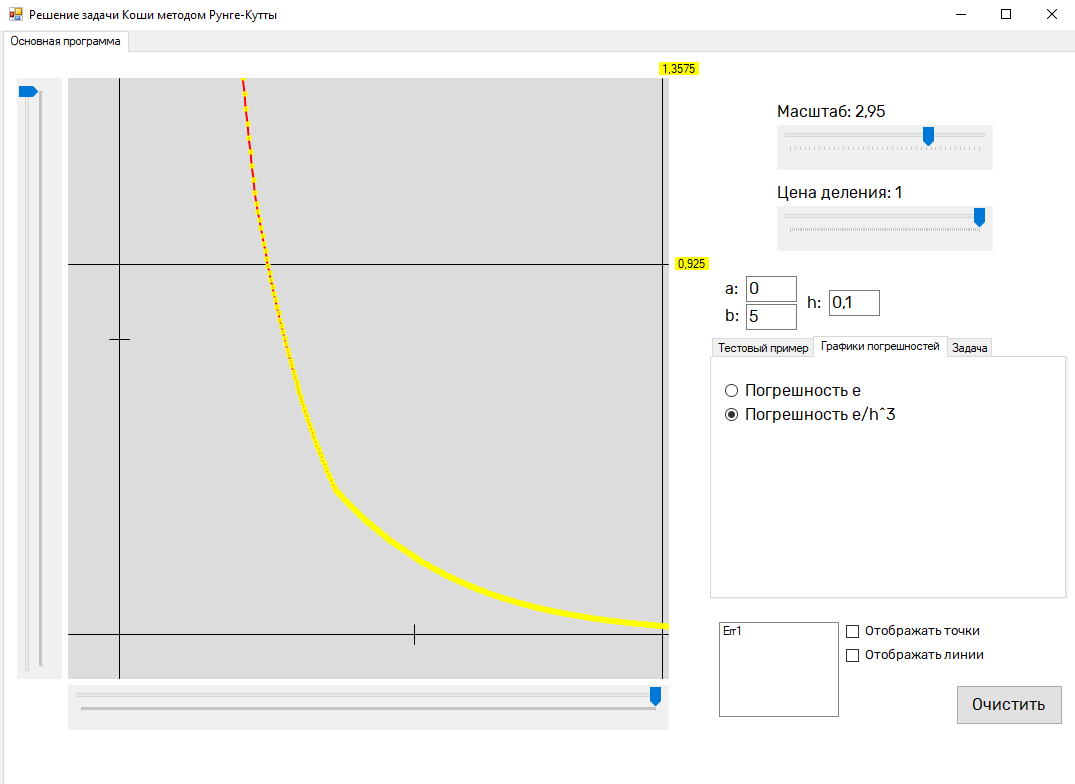
**

Рис.5 – График зависимости максимальной погрешности решения *e/* от выбранного шага *h*

Из графиков, приведенных на рисунках 5-6, можно сделать вывод о том, что порядок точности метода Рунге-Кутты, описанного в программе, соответствует заявленному порядку точности (третий порядок точности).

### Результаты решения задачи

При решении системы уравнений мы можем изменять начальное количество лейкоцитов (x0), клеток опухоли (y0) и коэффициент смертности (), остальные параметры заданы.

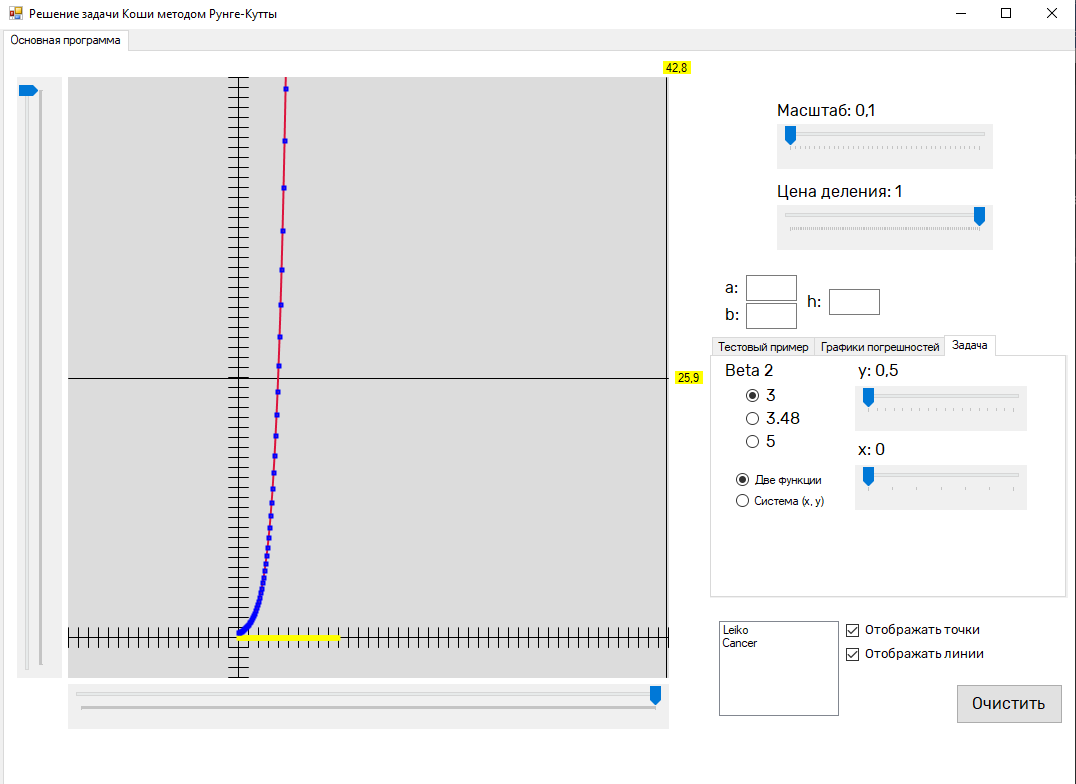


Рис.6 – График количества клеток опухоли (синие точки) и лейкоцитов (желтые точки) при начальных условиях: x0 = 0, y0 = 0.5, . По графику можно сделать вывод, что опухоль побеждает.

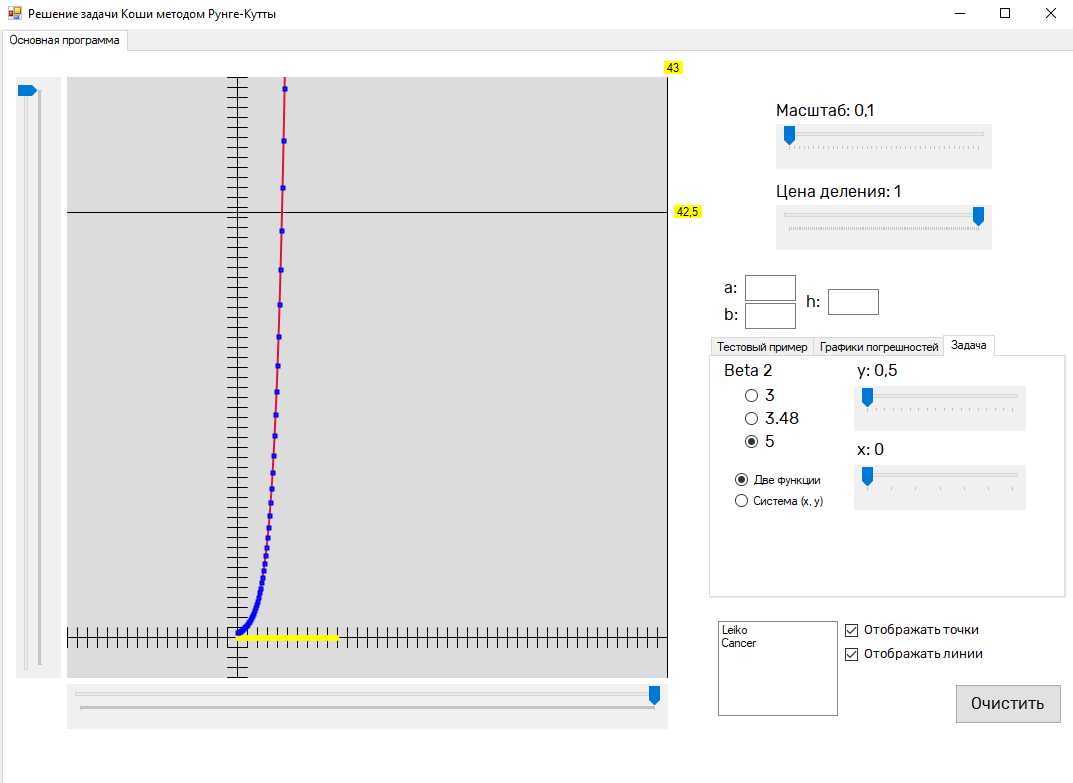
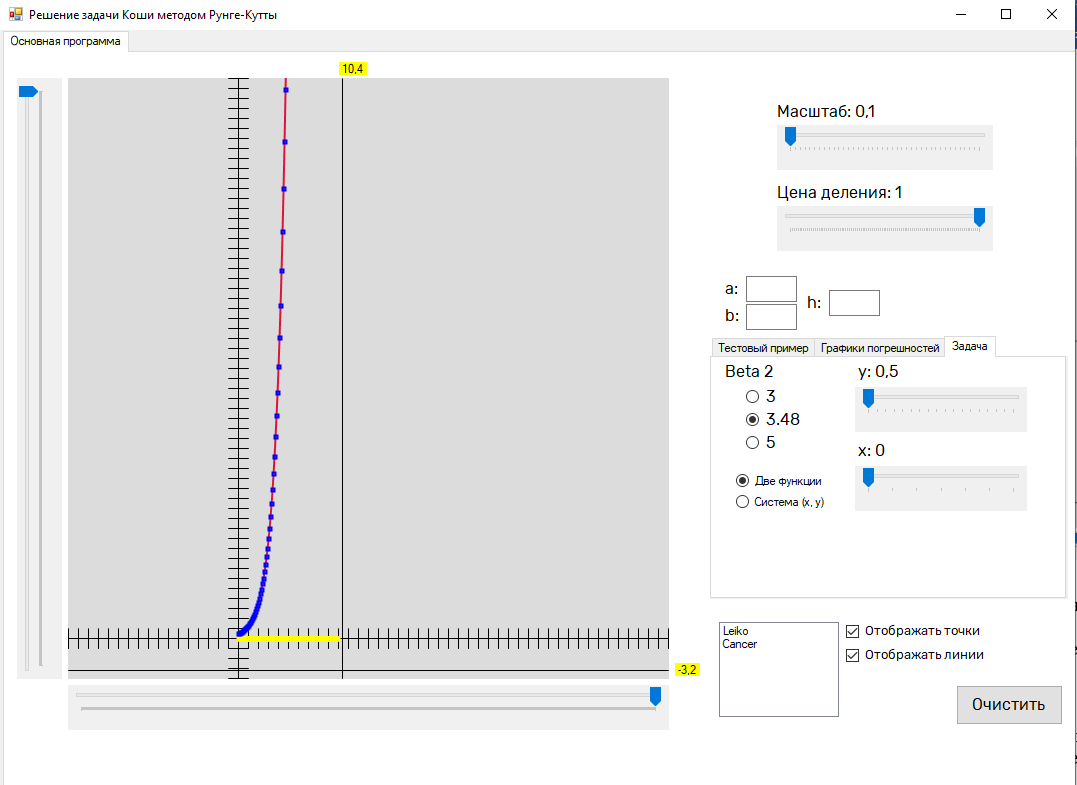
**

Рис.7 – График количества клеток опухоли (синие точки) и лейкоцитов (желтые точки) при другом начальном условии . По графикам можно сделать вывод, что опухоль побеждает.

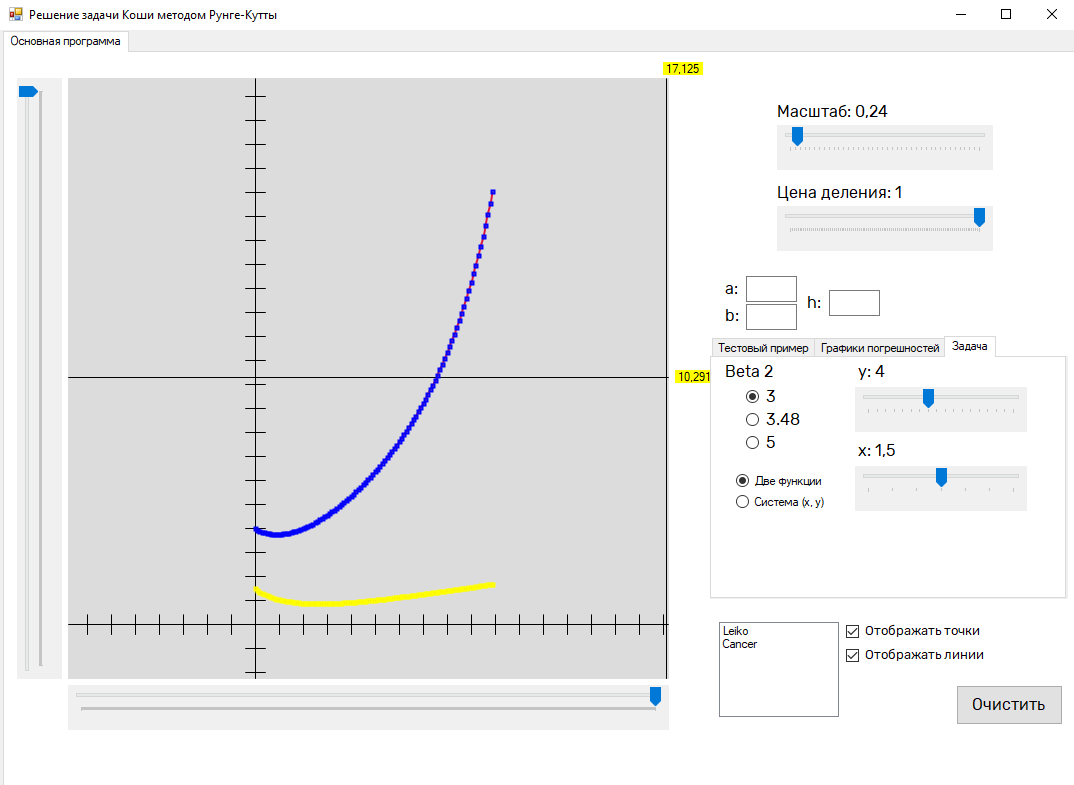


Рис.8 – График количества клеток опухоли (синие точки) и лейкоцитов (желтые точки) при начальных условиях: x0 = 1,5, y0 = 4, . По графику можно сделать вывод, что опухоль побеждает. При ситуация не меняется.

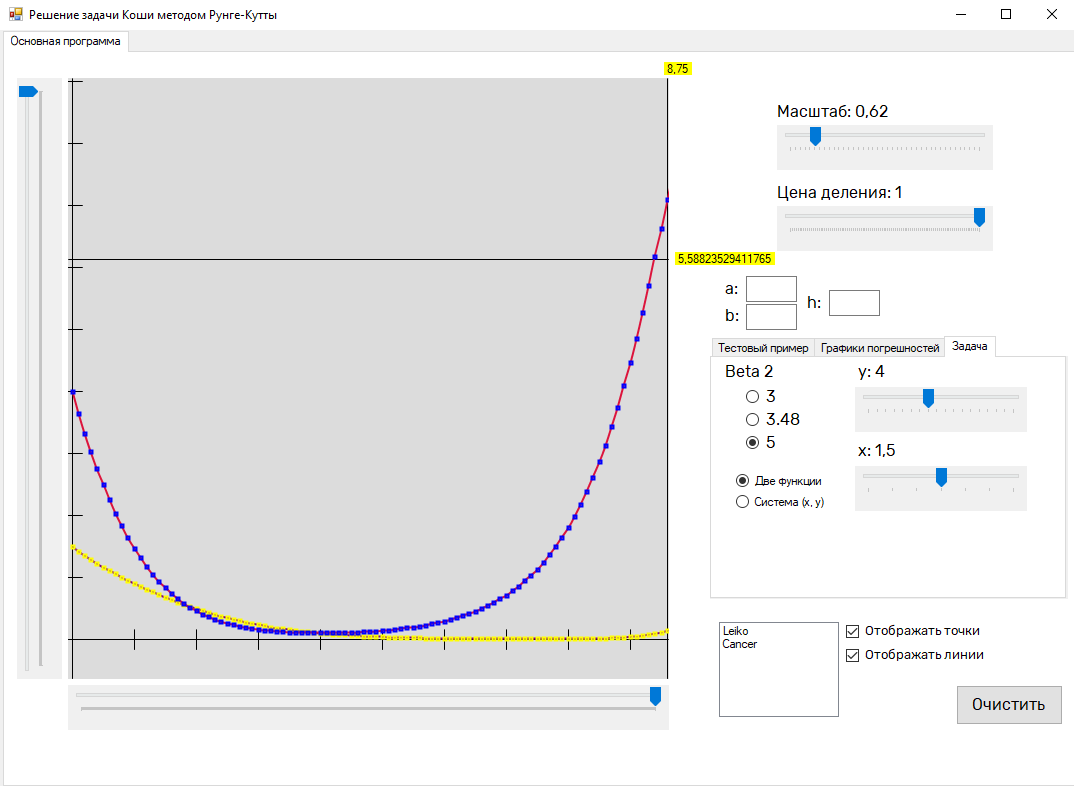


Рис.9 – График количества клеток опухоли (синие точки) и лейкоцитов (желтые точки) при начальных условиях: x0 = 1,5, y0 = 4, . Так как лейкоцитам не удаётся свести количество живых клеток опухоли в ноль, можно сделать вывод, что опухоль побеждает.

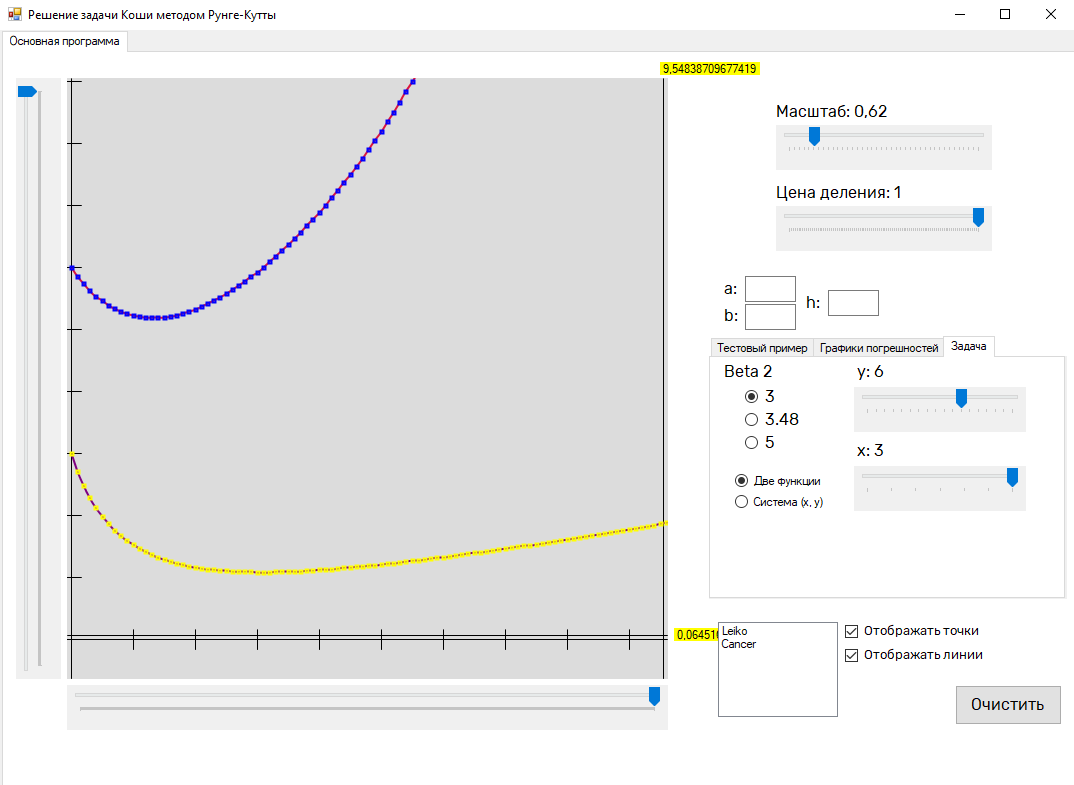


Рис.8 – График количества клеток опухоли (синие точки) и лейкоцитов (желтые точки) при начальных условиях: x0 = 3, y0 = 6, . По графику можно сделать вывод, что опухоль побеждает. При ситуация не меняется.

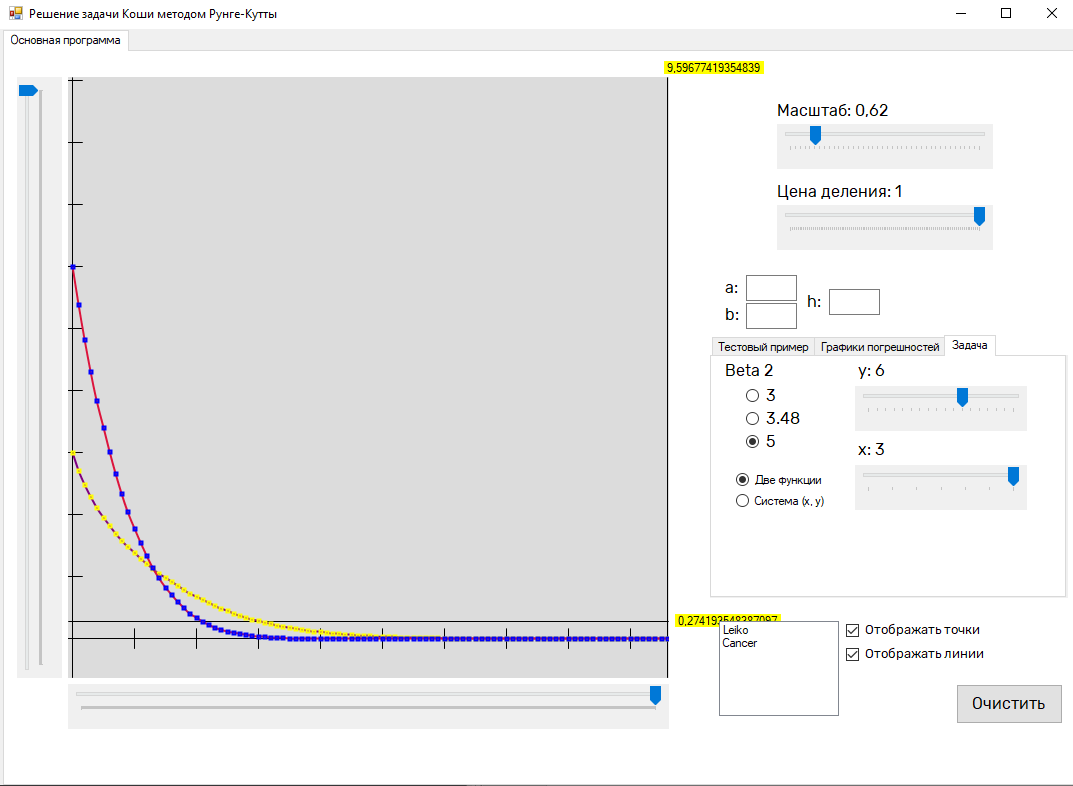


Рис.9 – График количества клеток опухоли (синие точки) и лейкоцитов (желтые точки) при начальных условиях: x0 = 3, y0 = 6, . Лейкоцитам удаётся свести количество живых опухолевых клеток в ноль, что говорит о выздоровлении организма.

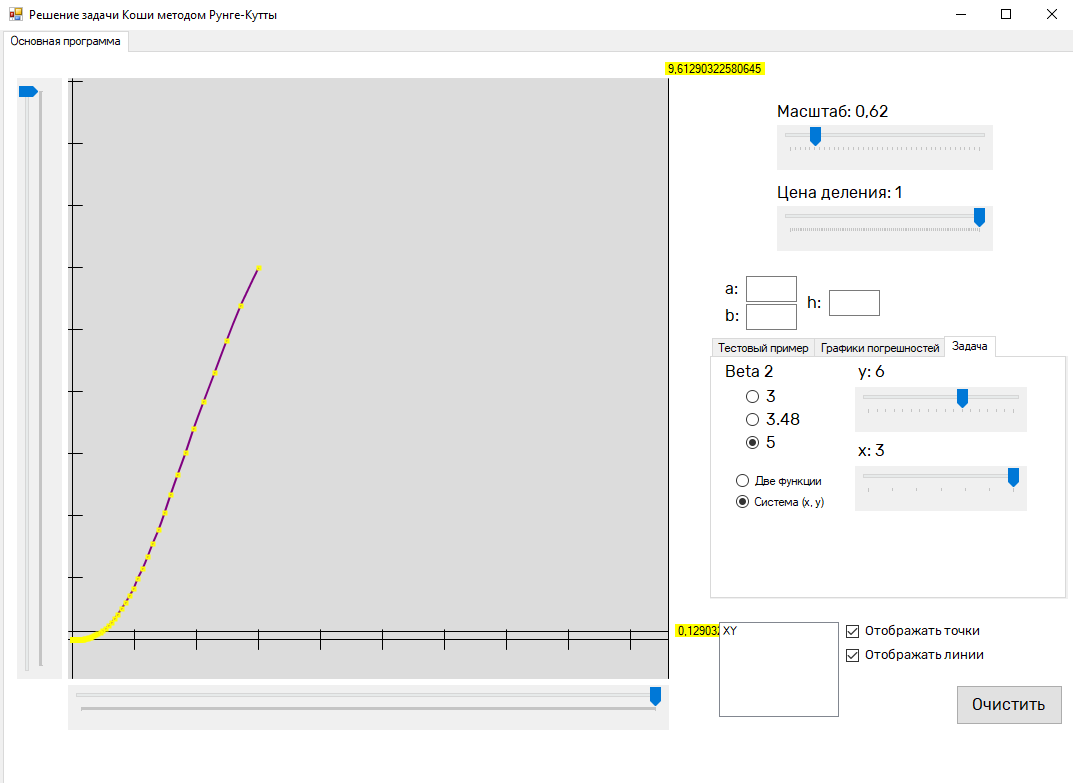


Рис.10 – Этот же график, но в координатах (x, y).

### Заключение

В ходе практической работы рассмотрено решение задачи Коши методом Рунге-Кутты 3 порядка точности, примененное для симуляции борьбы лейкоцитов и опухоли в организме. Метод Рунге-Кутты реализован в программе, проведены расчеты на тестовом примере.

### Список используемой литературы

1. elementy.ru [Электронный ресурс]. Режим доступа: https://elementy.ru/trefil/51/Teoriya\_ravnovesiya\_MakarturaUilsona (дата обращения 17.04.2019)
2. Хайрер Э., Нерсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. – М.: Мир, 1990.
3. Даутов, Р.З. Практикум по курсу численные методы. Решение задачи Коши для системы ОДУ / P.З. Даутов. – Изд-во Казанск. федер. ун-та, 2014 г. – 100 с.
4. Холодниок М., Клич А., Кубичек М., Марек М. Методы анализа нелинейных динамических моделей. — М.: Мир, 1991.
5. Самарский А.А., Гулин А.В. Численные методы.— М.:Наука,1989. 4. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. — М.: Наука, 1987.

### Приложение А

Код программы

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing.Drawing2D;

using System.Runtime.CompilerServices;

namespace practice

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

void RefreshForm()

{

scaleScrollValueLable.Text = Painter.scale.ToString();

divisionValueValueLable.Text = Painter.divisionValue.ToString();

Painter.UpdatePosition();

firstPagePaint.Invalidate();

}

void RefreshListBox()

{

listBox1.Items.Clear();

foreach (Curve i in Painter.curves)

{

listBox1.Items.Add(i.name);

}

}

private void firstPagePaint\_Paint(object sender, PaintEventArgs e)

{

DoubleBuffered = true;

Graphics graphics = firstPagePaint.CreateGraphics();

graphics.SmoothingMode = SmoothingMode.AntiAlias;

Pen blackPen = new Pen(Color.Black, 1);

Pen greenPen = new Pen(Color.Green, 2);

graphics.DrawLine(blackPen, new Point(Painter.cursorX, 0), new Point(Painter.cursorX, Painter.SCREEN\_HEIGHT));

graphics.DrawLine(blackPen, new Point(0, Painter.cursorY), new Point(Painter.SCREEN\_WIDTH, Painter.cursorY));

if(Painter.leftDownX <= 0)

graphics.DrawLine(blackPen, new Point(0, 0), new Point(0, Painter.SCREEN\_HEIGHT));

else if (Painter.leftDownX >= Painter.SCREEN\_WIDTH)

graphics.DrawLine(blackPen, new Point(Painter.SCREEN\_WIDTH, 0), new Point(Painter.SCREEN\_WIDTH, Painter.SCREEN\_HEIGHT));

else

graphics.DrawLine(blackPen, new Point(Painter.leftDownX, 0), new Point(Painter.leftDownX, Painter.SCREEN\_HEIGHT));

if (Painter.leftDownY <= 0)

graphics.DrawLine(blackPen, new Point(0, 0), new Point(Painter.SCREEN\_WIDTH, 0));

else if (Painter.leftDownY >= Painter.SCREEN\_WIDTH)

graphics.DrawLine(blackPen, new Point(0, Painter.SCREEN\_HEIGHT), new Point(Painter.SCREEN\_WIDTH, Painter.SCREEN\_HEIGHT));

else

graphics.DrawLine(blackPen, new Point(0, Painter.leftDownY), new Point(Painter.SCREEN\_WIDTH, Painter.leftDownY));

for (int i = -(int)(Painter.SCREEN\_WIDTH - Painter.SCREEN\_WIDTH % (Painter.SCREEN\_DIV\_VALUE \* Painter.scale)); i < Painter.SCREEN\_WIDTH; i += (int)(Painter.SCREEN\_DIV\_VALUE \* Painter.scale))

{

if (Painter.leftDownX <= 0)

graphics.DrawLine(blackPen, new Point(0 - 10, Painter.leftDownY - i), new Point(0 + 10, Painter.leftDownY - i));

else if (Painter.leftDownX >= Painter.SCREEN\_WIDTH)

graphics.DrawLine(blackPen, new Point(Painter.SCREEN\_WIDTH - 10, Painter.leftDownY - i), new Point(Painter.SCREEN\_WIDTH + 10, Painter.leftDownY - i));

else

graphics.DrawLine(blackPen, new Point(Painter.leftDownX - 10, Painter.leftDownY - i), new Point(Painter.leftDownX + 10, Painter.leftDownY - i));

if (Painter.leftDownY <= 0)

graphics.DrawLine(blackPen, new Point(Painter.leftDownX - i, 0 - 10), new Point(Painter.leftDownX - i, 0 + 10));

else if (Painter.leftDownY >= Painter.SCREEN\_WIDTH)

graphics.DrawLine(blackPen, new Point(Painter.leftDownX - i, Painter.SCREEN\_HEIGHT - 10), new Point(Painter.leftDownX - i, Painter.SCREEN\_HEIGHT + 10));

else

graphics.DrawLine(blackPen, new Point(Painter.leftDownX - i, Painter.leftDownY - 10), new Point(Painter.leftDownX - i, Painter.leftDownY + 10));

}

try

{

foreach (Curve i in Painter.curves)

{

if(i.drawCurve)

graphics.DrawCurve(new Pen(i.color, 2), i.ConvertToPoint());

if(i.drawDots)

foreach(Dot j in i.dots)

{

graphics.DrawRectangle(new Pen(i.dotsColor, 2), j.X - 1, j.Y - 1, 3, 3);

}

}

foreach (Dot i in Painter.dots)

{

graphics.DrawRectangle(greenPen, i.X - 1, i.Y - 1, 3, 3);

}

}

catch

{

}

}

private void tabPage1\_Click(object sender, EventArgs e)

{

}

private void scaleScroll\_Scroll(object sender, EventArgs e)

{

Painter.scale = scaleScroll.Value / 100.0;

RefreshForm();

}

private void divisionValueScroll\_Scroll(object sender, EventArgs e)

{

Painter.divisionValue = divisionValueScroll.Value / 10000.0;

RefreshForm();

}

private void scaleScrollValueLable\_Click(object sender, EventArgs e)

{

}

private void verticalScroll\_Scroll(object sender, EventArgs e)

{

Painter.leftDownY = Painter.SCREEN\_HEIGHT + verticalScroll.Value;

RefreshForm();

}

private void horizontalScroll\_Scroll(object sender, EventArgs e)

{

Painter.leftDownX = 0 - horizontalScroll.Value;

RefreshForm();

}

private void firstPagePaint\_MouseHover(object sender, EventArgs e)

{

}

private void firstPagePaint\_MouseMove(object sender, MouseEventArgs e)

{

Painter.UpdateCursorInfo(e);

cursorXLable.Location = new Point(58 + Painter.cursorX, cursorXLable.Location.Y);

cursorYLable.Location = new Point(cursorYLable.Location.X, 18 + Painter.cursorY);

cursorXLable.Text = ((double)(Painter.cursorX - Painter.leftDownX) / Painter.SCREEN\_DIV\_VALUE / Painter.scale \* Painter.divisionValue).ToString();

cursorYLable.Text = ((double)(Painter.leftDownY - Painter.cursorY) / Painter.SCREEN\_DIV\_VALUE / Painter.scale \* Painter.divisionValue).ToString();

if(Painter.mouseDown)

{

Painter.leftDownX = Painter.leftDownXOld + (Painter.cursorX - Painter.cursorXOld);

Painter.leftDownY = Painter.leftDownYOld + (Painter.cursorY - Painter.cursorYOld);

}

RefreshForm();

}

private void testTaskButton\_CheckedChanged(object sender, EventArgs e)

{

if(testTaskButton.Checked)

{

try

{

Painter.Clear();

double a = double.Parse(aTextBox.Text);

double b = double.Parse(bTextBox.Text);

double h = double.Parse(hTextBox.Text);

Painter.ImportFromMatrix(Counter.AccurateCount(Counter.TestAccurate, 0, a, b, h), "Test1" , Color.Red, Color.Green);

Painter.ImportFromMatrix(Counter.AccurateCount(Counter.TestAccurate, 1, a, b, h), "Test2" , Color.Purple, Color.Blue);

RefreshForm();

RefreshListBox();

}

catch

{

MessageBox.Show("Введены неверные значения!!!");

}

}

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

foreach (Curve i in Painter.curves)

{

if (i.name.Contains("Test"))

i.drawDots = checkBox1.Checked;

}

RefreshForm();

}

private void clearButton\_Click(object sender, EventArgs e)

{

Painter.Clear();

RefreshForm();

}

private void rungeKuttaTest\_CheckedChanged(object sender, EventArgs e)

{

if (rungeKuttaTest.Checked)

{

try

{

Painter.Clear();

double a = double.Parse(aTextBox.Text);

double b = double.Parse(bTextBox.Text);

double h = double.Parse(hTextBox.Text);

Painter.ImportFromMatrix(Counter.RungeKutta(Counter.TestODU, 0, a, b, h, a, Counter.GetFirstValues(Counter.TestAccurate, a, 2)),"Runge1", Color.Red, Color.Cyan);

Painter.ImportFromMatrix(Counter.RungeKutta(Counter.TestODU, 1, a, b, h, a, Counter.GetFirstValues(Counter.TestAccurate, a, 2)), "Runge2", Color.Red, Color.Yellow);

RefreshForm();

RefreshListBox();

}

catch

{

MessageBox.Show("Введены неверные значения!!!");

}

}

}

private void checkBox2\_CheckedChanged(object sender, EventArgs e)

{

foreach (Curve i in Painter.curves)

{

if(i.name.Contains("Test"))

i.drawCurve = checkBox2.Checked;

}

RefreshForm();

}

private void checkBox3\_CheckedChanged(object sender, EventArgs e)

{

foreach (Curve i in Painter.curves)

{

if (i.name.Contains("Runge"))

i.drawDots = checkBox3.Checked;

}

RefreshForm();

}

private void checkBox4\_CheckedChanged(object sender, EventArgs e)

{

foreach (Curve i in Painter.curves)

{

if (i.name.Contains("Runge"))

i.drawCurve = checkBox4.Checked;

}

RefreshForm();

}

private void togetherRadioButton\_CheckedChanged(object sender, EventArgs e)

{

if (togetherRadioButton.Checked)

{

try

{

Painter.Clear();

double a = double.Parse(aTextBox.Text);

double b = double.Parse(bTextBox.Text);

double h = double.Parse(hTextBox.Text);

Painter.ImportFromMatrix(Counter.AccurateCount(Counter.TestAccurate, 0, a, b, h), "Test1", Color.Red, Color.Green);

Painter.ImportFromMatrix(Counter.AccurateCount(Counter.TestAccurate, 1, a, b, h), "Test2", Color.Purple, Color.Blue);

Painter.ImportFromMatrix(Counter.RungeKutta(Counter.TestODU, 0, a, b, h, a, Counter.GetFirstValues(Counter.TestAccurate, a, 2)), "Runge1", Color.Red, Color.Cyan);

Painter.ImportFromMatrix(Counter.RungeKutta(Counter.TestODU, 1, a, b, h, a, Counter.GetFirstValues(Counter.TestAccurate, a, 2)), "Runge2", Color.Red, Color.Yellow);

RefreshForm();

RefreshListBox();

}

catch

{

MessageBox.Show("Введены неверные значения!!!");

}

}

}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton2.Checked)

{

try

{

Painter.Clear();

double a = double.Parse(aTextBox.Text);

double b = double.Parse(bTextBox.Text);

double h = double.Parse(hTextBox.Text);

Painter.ImportFromMatrix(Counter.CountErr(Counter.RungeKutta, Counter.TestODU, Counter.TestAccurate, a, b, a, Counter.GetFirstValues(Counter.TestAccurate, a, 2), 0), "Err1", Color.Red, Color.Yellow);

RefreshForm();

RefreshListBox();

}

catch

{

MessageBox.Show("Введены неверные значения!!!");

}

}

}

private void firstPagePaint\_MouseDown(object sender, MouseEventArgs e)

{

Painter.mouseDown = true;

Painter.cursorXOld = Painter.cursorX;

Painter.cursorYOld = Painter.cursorY;

Painter.leftDownXOld = Painter.leftDownX;

Painter.leftDownYOld = Painter.leftDownY;

}

private void firstPagePaint\_MouseUp(object sender, MouseEventArgs e)

{

Painter.mouseDown = false;

Painter.cursorXOld = 0;

Painter.cursorYOld = 0;

Painter.leftDownXOld = 0;

Painter.leftDownYOld = 0;

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton1.Checked)

{

try

{

Painter.Clear();

double a = double.Parse(aTextBox.Text);

double b = double.Parse(bTextBox.Text);

double h = double.Parse(hTextBox.Text);

Painter.ImportFromMatrix(Counter.CountErr(Counter.RungeKutta, Counter.TestODU, Counter.TestAccurate, a, b, a, Counter.GetFirstValues(Counter.TestAccurate, a, 2), 1), "Err1", Color.Red, Color.Yellow);

RefreshForm();

RefreshListBox();

}

catch

{

MessageBox.Show("Введены неверные значения!!!");

}

}

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

foreach(Curve i in Painter.curves)

{

if(i.name.Contains(listBox1.SelectedItem.ToString()))

{

showDots.Checked = i.drawDots;

showCurve.Checked = i.drawCurve;

}

}

}

private void showDots\_CheckedChanged(object sender, EventArgs e)

{

if (listBox1.SelectedItem != null)

{

foreach (Curve i in Painter.curves)

{

if (i.name.Contains(listBox1.SelectedItem.ToString()))

{

i.drawDots = showDots.Checked;

RefreshForm();

}

}

}

}

private void showCurve\_CheckedChanged(object sender, EventArgs e)

{

if (listBox1.SelectedItem != null)

{

foreach (Curve i in Painter.curves)

{

if (i.name.Contains(listBox1.SelectedItem.ToString()))

{

i.drawCurve = showCurve.Checked;

RefreshForm();

}

}

}

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

yLable.Text = "y: " + (trackBar1.Value / 10.0).ToString();

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

private void radioButton3\_CheckedChanged(object sender, EventArgs e)

{

if(radioButton3.Checked)

{

Counter.beta2 = 3;

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

}

private void radioButton4\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton4.Checked)

{

Counter.beta2 = 3.48;

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

}

private void radioButton5\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton5.Checked)

{

Counter.beta2 = 5;

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

}

private void TaskGraphic (double x0, double y0)

{

Painter.Clear();

if (twoFuncButton.Checked)

{

Painter.ImportFromMatrix(Counter.RungeKutta(Counter.TaskODU, 0, 0, 10, 0.1, 0, new double[2] { x0, y0 }), "Leiko", Color.Purple, Color.Yellow);

Painter.ImportFromMatrix(Counter.RungeKutta(Counter.TaskODU, 1, 0, 10, 0.1, 0, new double[2] { x0, y0 }), "Cancer", Color.Crimson, Color.Blue);

}

else if (xySystemButton.Checked)

{

Painter.ImportFromMatrix(Counter.ConvertToXY(Counter.RungeKutta(Counter.TaskODU, 0, 0, 10, 0.1, 0, new double[2] { x0, y0 }), Counter.RungeKutta(Counter.TaskODU, 1, 0, 10, 0.1, 0, new double[2] { x0, y0 })), "XY", Color.Purple, Color.Yellow);

}

RefreshListBox();

RefreshForm();

}

private void trackBar2\_Scroll(object sender, EventArgs e)

{

xLable.Text = "x: " + (trackBar2.Value / 10.0).ToString();

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

private void twoFuncButton\_CheckedChanged(object sender, EventArgs e)

{

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

private void xySystemButton\_CheckedChanged(object sender, EventArgs e)

{

TaskGraphic(trackBar2.Value / 10.0, trackBar1.Value / 10.0);

}

}

}

**Counter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace practice

{

public static class Counter

{

public static double lambda1 = 1;

public static double lambda2 = 1;

public static double beta1 = 1;

public static double beta2 = 3;

public static double c = 3;

public delegate double FuncDelegate(int funcInd, double x, double[] y);

public delegate List<double[,]> RungeDelegate(FuncDelegate funcDelegate, double a, double b, double h, double t0, double[] y0);

public static double TestAccurate(int funcInd, double x, double[] y)

{

double res = 0;

switch (funcInd)

{

case 0:

res = Math.Cos(x) / Math.Sqrt(1 + Math.Pow(Math.E, 2 \* x));

break;

case 1:

res = Math.Sin(x) / Math.Sqrt(1 + Math.Pow(Math.E, 2 \* x));

break;

}

return res;

}

public static double[,] AccurateCount(FuncDelegate funcDelegate, int funcInd, double a, double b, double h)

{

int N = (int)((b - a) / h);

double[,] result = new double[N, 2];

int k = 0;

for (double i = a; k < N; i += h)

{

result[k, 0] = i;

result[k, 1] = funcDelegate(funcInd, i, new double[1]);//TestAccurate(i, firstFunc);

k++;

}

return result;

}

public static double[] GetFirstValues(FuncDelegate funcDelegate, double t0, int numOfValues)

{

double[] res = new double[numOfValues];

for(int i = 0; i < numOfValues; i++)

{

res[i] = funcDelegate(i, t0, new double[1]);

}

return res;

}

public static double TestODU(int funcInd, double t, double[] y)

{

switch (funcInd)

{

case 0:

return -Math.Sin(t) / Math.Sqrt(1 + Math.Pow(Math.E, 2 \* t)) + y[0] \* (Math.Pow(y[0], 2) + Math.Pow(y[1], 2) - 1);

case 1:

return Math.Cos(t) / Math.Sqrt(1 + Math.Pow(Math.E, 2 \* t)) + y[1] \* (Math.Pow(y[0], 2) + Math.Pow(y[1], 2) - 1);

}

return 0;

}

public static double TaskODU(int funcInd, double t, double[] yPrev)

{

double x = yPrev[0];

double y = yPrev[1];

switch (funcInd)

{

case 0:

return (-lambda1 + beta1 \* Math.Pow(y \* y, 1.0 / 3) \* (1 - x / c) / (1 + x)) \* x;

case 1:

double ans = lambda2 \* y - beta2 \* x \* Math.Pow(y \* y, 1.0 / 3) / (1 + x);

return ans;

}

return 0;

}

public static double[,] RungeKutta(FuncDelegate funcDelegate, int funcInd, double a, double b, double h, double t0, double[] y0)

{

int N = (int)((b - a) / h);

List<double[,]> yshtr = new List<double[,]>();

for (int i = 0; i < y0.Length; i++)

{

yshtr.Add(new double[N, 2]);

yshtr[i][0, 0] = t0;

yshtr[i][0, 1] = y0[i];

}

for (int i = 1; i < N; i += 1)

{

double[] yPrev = new double[y0.Length];

for(int j = 0; j < y0.Length; j++)

{

yPrev[j] = yshtr[j][i - 1, 1];

}

double[] k1 = new double[y0.Length];

double[] k2 = new double[y0.Length];

double[] k3 = new double[y0.Length];

for(int j = 0; j < y0.Length; j++)

{

yshtr[j][i, 0] = a + i \* h;

k1[j] = funcDelegate(j, yshtr[j][i, 0], yPrev);

}

for(int j = 0; j < y0.Length; j++)

{

k2[j] = funcDelegate(j, yshtr[j][i, 0] + h / 2, Addict(/\*h \* k1 / 2\*/ Multyplicate(h / 2, k1), yPrev));

}

for (int j = 0; j < y0.Length; j++)

{

k3[j] = funcDelegate(j, yshtr[j][i, 0] + 3 \* h / 4, Addict(/\*h \* k1 / 2\*/ Multyplicate(3 \* h / 4, k2), yPrev));

}

for (int j = 0; j < y0.Length; j++)

{

yshtr[j][i, 1] = yshtr[j][i - 1, 1] + h \* (2 \* k1[j] + 3 \* k2[j] + 4 \* k3[j]) / 9;

}

}

return yshtr[funcInd];

}

public static List<double[,]> RungeKutta(FuncDelegate funcDelegate, double a, double b, double h, double t0, double[] y0)

{

int N = (int)((b - a) / h);

List<double[,]> yshtr = new List<double[,]>();

for (int i = 0; i < y0.Length; i++)

{

yshtr.Add(new double[N, 2]);

yshtr[i][0, 0] = t0;

yshtr[i][0, 1] = y0[i];

}

for (int i = 1; i < N; i += 1)

{

double[] yPrev = new double[y0.Length];

for (int j = 0; j < y0.Length; j++)

{

yPrev[j] = yshtr[j][i - 1, 1];

}

for (int j = 0; j < y0.Length; j++)

{

yshtr[j][i, 0] = a + i \* h;

double k1 = funcDelegate(j, yshtr[j][i, 0], yPrev);

double k2 = funcDelegate(j, yshtr[j][i, 0] + h / 2, Addict(h \* k1 / 2, yPrev));

double k3 = funcDelegate(j, yshtr[j][i, 0] + 3 \* h / 4, Addict(3 \* h \* k2 / 4, yPrev));

yshtr[j][i, 1] = yshtr[j][i - 1, 1] + h \* (2 \* k1 + 3 \* k2 + 4 \* k3) / 9;

}

}

return yshtr;

}

public static double[,] CountErr(RungeDelegate rungeDelegate, FuncDelegate funcDelegate, FuncDelegate trueFunc, double a, double b, double t0, double[] y0, int errInd)

{

int k = 0;

double[,] res = new double[ (int)(2 / 0.005), 2];

double[,] res2 = new double[(int)(2 / 0.005), 2];

for (double i = 0.005; i <= 2; i += 0.005)

{

double maxErr = 0;

List<double[,]> list = rungeDelegate(funcDelegate, a, b, i, t0, y0);

int m = 0;

foreach (double[,] j in list)

{

for(int p = 0; p < j.GetLength(0); p++)

{

if(Math.Abs(j[p, 1] - trueFunc(m, j[p, 0], y0)) > maxErr)

{

maxErr = Math.Abs(j[p, 1] - trueFunc(m, j[p, 0], y0));

}

}

m++;

}

res[k, 0] = i;

res2[k, 0] = i;

res[k, 1] = maxErr;

res2[k, 1] = maxErr / Math.Pow(i, 3);

k++;

}

switch(errInd)

{

case 0:

return res;

case 1:

return res2;

default:

return new double[1,1];

}

}

public static int GetNumOfEls(double b)

{

int i = 0;

double h = 0.0005;

while (h <= b)

{

h += h \* 2;

i++;

}

return i;

}

public static double[] Addict(double num, double[] mass)

{

double[] res = new double[mass.Length];

for(int i = 0; i < mass.Length; i++)

{

res[i] = mass[i] + num;

}

return res;

}

public static double[] Addict(double[] mass1, double[] mass2)

{

double[] res = new double[mass1.Length];

for (int i = 0; i < mass1.Length; i++)

{

res[i] = mass1[i] + mass2[i];

}

return res;

}

public static double[] Multyplicate(double num, double[] mass)

{

double[] res = new double[mass.Length];

for (int i = 0; i < mass.Length; i++)

{

res[i] = mass[i] \* num;

}

return res;

}

public static double[,] ConvertToXY(double[,] x, double[,] y)

{

double[,] result = new double[x.GetLength(0), 2];

for(int i = 0; i < x.GetLength(0); i++)

{

result[i, 0] = x[i, 1];

result[i, 1] = y[i, 1];

}

return result;

}

}

}

**Curve.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace practice

{

class Curve

{

public List<Dot> dots = new List<Dot>();

public string name = "";

public bool enabled = true;

public bool drawDots = true;

public bool drawCurve = true;

public Color color = Color.Red;

public Color dotsColor = Color.Green;

public Curve(List<Dot> \_dots, string \_name)

{

dots = \_dots;

name = \_name;

}

public Curve(List<Dot> \_dots, string \_name, Color \_curveColor, Color \_dotColor)

{

dots = \_dots;

name = \_name;

color = \_curveColor;

dotsColor = \_dotColor;

}

public Point[] ConvertToPoint()

{

Point[] res = new Point[dots.Count];

int k = 0;

foreach(Dot i in dots)

{

res[k] = i.ConvertToPoint();

k++;

}

return res;

}

}

}

**Dot.cs**

class Dot

{

public int X, Y;

public double doubX = 0;

public double doubY = 0;

public Dot(int \_X, int \_Y)

{

X = \_X;

Y = \_Y;

}

public Dot(double \_X, double \_Y)

{

doubX = \_X;

doubY = \_Y;

X = Painter.leftDownX + (int)(\_X \* Painter.SCREEN\_DIV\_VALUE \* Painter.scale / Painter.divisionValue);

Y = Painter.leftDownY - (int)(\_Y \* Painter.SCREEN\_DIV\_VALUE \* Painter.scale / Painter.divisionValue);

}

public void UpdatePosition()

{

X = Painter.leftDownX + (int)(doubX \* Painter.SCREEN\_DIV\_VALUE \* Painter.scale / Painter.divisionValue);

Y = Painter.leftDownY - (int)(doubY \* Painter.SCREEN\_DIV\_VALUE \* Painter.scale / Painter.divisionValue);

}

public Point ConvertToPoint()

{

return new Point(X, Y);

}

}

**Painter.cs**

static class Painter

{

public const int SCREEN\_WIDTH = 600;

public const int SCREEN\_HEIGHT = 600;

public const int SCREEN\_DIV\_VALUE = 100;

public static int leftDownX = 0;

public static int leftDownXOld = 0;

public static int leftDownY = SCREEN\_HEIGHT;

public static int leftDownYOld = SCREEN\_HEIGHT;

public static Pen blackPen = new Pen(Color.Black, 1);

public static Pen greenPen = new Pen(Color.Green, 2);

public static Pen redPen = new Pen(Color.Red, 2);

public static int cursorX = 0;

public static int cursorY = 0;

public static int cursorXOld = 0;

public static int cursorYOld = 0;

public static bool mouseDown = false;

public static double scale = 0.1;

public static double divisionValue = 1;

public static List<Dot> dots = new List<Dot>();

public static List<Curve> curves = new List<Curve>();

public static void UpdatePosition()

{

foreach (Dot i in dots)

{

i.UpdatePosition();

}

foreach (Curve i in curves)

{

foreach (Dot j in i.dots)

{

j.UpdatePosition();

}

}

}

public static void Clear()

{

dots.Clear();

curves.Clear();

}

public static void UpdateCursorInfo(MouseEventArgs eventArgs)

{

cursorX = eventArgs.X;

cursorY = eventArgs.Y;

}

public static void ImportFromMatrix(double[,] matrix, string funcName)

{

List<Dot> newCurveDots = new List<Dot>();

for (int i = 0; i < matrix.GetUpperBound(0) + 1; i++)

{

newCurveDots.Add(new Dot(matrix[i, 0], matrix[i, 1]));

}

curves.Add(new Curve(newCurveDots, funcName));

}

public static void ImportFromMatrix(double[,] matrix, string funcName, Color curveColor, Color dotsColor)

{

List<Dot> newCurveDots = new List<Dot>();

for (int i = 0; i < matrix.GetUpperBound(0) + 1; i++)

{

newCurveDots.Add(new Dot(matrix[i, 0], matrix[i, 1]));

}

curves.Add(new Curve(newCurveDots, funcName, curveColor, dotsColor));

}

}