Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет**

**Выполнил**:

Cтудент группы 381703-2

Колегов И.А.

**Проверил**:

Доцент кафедры ПМ

Панкратова E.B

Нижний Новгород

2019

Оглавление

[1. Методы численного интегрирования 3](#_Toc28122073)

[1.1 Вычисление значения определенного интеграла 3](#_Toc28122074)

[1.2 Численные методы Рунге-Кутты 4](#_Toc28122075)

[1.3 Интегрирование с переменным шагом 4](#_Toc28122076)

[2. Методы визуализации 7](#_Toc28122077)

[2.1 Визуализация в реальном времени 7](#_Toc28122078)

[2.2 Визуализация посчитанных заранее результатов 8](#_Toc28122079)

[3. Модель ФитцХью — Нагумо 11](#_Toc28122080)

[Приложение 13](#_Toc28122081)

# 1. Методы численного интегрирования

## 1.1 Вычисление значения определенного интеграла

Существует несколько способов вычисления определенного интеграла:

Самый простой – метод прямоугольников. У него есть 3 основные разновидности: метод левых прямоугольник, правых прямоугольников и средних прямоугольников. Рассмотрим метод средних прямоугольников поподробнее.

Данная формула вычисляет приближенное значение интеграла, на отрезке . Если мы построим равномерную сетку с узлами в точках и шагом , то сможем определить формулу средних прямоугольников как:

Исследуя аппроксимацию на каждом участке через формулу Тейлора, мы получим, что погрешность является величиной порядка .

Второй по известности способ вычисления интеграла – метод трапеции.

На равномерной сетке мы получим следующую формулу

При исследовании данного метода, мы приходим к выводу, что погрешность так же ведется себя как .

Для нахождения первообразной функции, в теории, мы можем построить сетку, и проинтегрировать функцию на каждом из полученных участков. Но точность часто оказывается недостаточной. Так же мы не всегда можем выразить исходную функцию.

## 1.2 Численные методы Рунге-Кутты

Методы Рунге-Кутты – широкий класс итерационных методов, для численного решения задачи Коши для обыкновенных дифференциальных уравнений, а так же систем таких уравнений.

Самый простой из методов данного класса – метод Эйлера

Здесь – обозначает численное решение, – точное решение. Данный метод имеет первый прядок. Следовательно, ошибка на конкретном шаге имеет порядок , но суммарная ошибка будет иметь порядок . Данная точность слишком мала для исследования поставленной задачи, поэтому был выбран явный метод Рунге-Кутты четвертого порядка.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Ошибка на конкретном шаге данного метода составляет величину, сопоставимую с . Данная точность является вполне приемлемой для решения большого класса задач.

Метод может быть без труда экстраполирован на систему обыкновенных дифференциальных уравнений.

## 1.3 Интегрирование с переменным шагом

Глобальной погрешностью называется разница между точным решением задачи и численным решением. Локальной погрешностью на определенном шаге называют разницу точного решения, проходящего через точку предыдущего шага, и численного решения на данном шаге. Зная локальную погрешность, становится возможным контролировать шаг.

Таким образом, если погрешность становится меньше какого-то наперед заданного , мы можем увеличить шаг, для ускорения процесса. Напротив, если погрешность становится больше чем , то шаг необходимо уменьшить.

Существует несколько способов оценить локальную погрешность. Например, мы можем взять метод более высокого порядка, и сосчитать значение на интересующей нас итерации. Либо мы можем сосчитать значение в точке, при помощи двойного счета.

Обозначим – значение, посчитанное в точке . – значение, посчитанное в той же точке, при помощи половинного шага. Тогда оценка локальной погрешности задается формулой:

Где p – порядок метода. Эта погрешность должна находиться в пределах:

Если границы нарушаются – мы меняем шаг.

Проверка порядка метода, на примере программы, для численного решения системы ФитцХью-Нагумо.

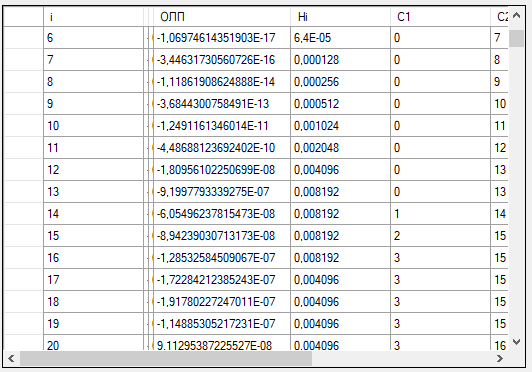


Рис. 1, данные о численном решении

Между 6 и 7 итерацией шаг увеличился в 2 раза

Следовательно, метод имеет 4 порядок.

Переменный шаг – удобный способ ускорения алгоритма численного интегрирования, без потери точности.

# 2. Методы визуализации

## 2.1 Визуализация в реальном времени

Для удобного дальнейшего исследования системы ФитцХью-Нагумо было разработано приложение с пользовательским интерфейсом, в котором можно настроить параметры системы, а так же увидеть график полученного решения.

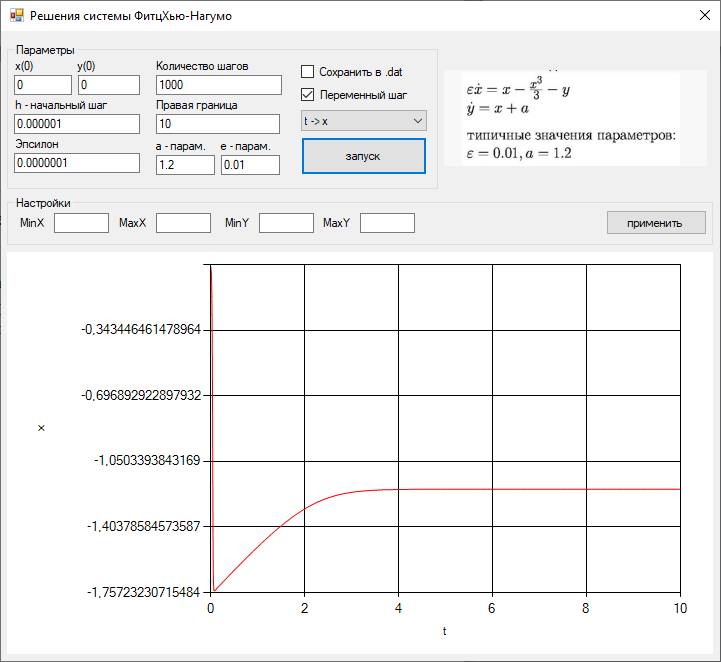


Рис. 2, пользовательский интерфейс

Программа имеет выпадающее меню, сверху от кнопки «запуск», которое позволяет выбрать какое из решений будет отображено в графике (y(t), x(t) или y(x)).

На первой стадии разработки было принято решение отправлять данные на отрисовку сразу по мере счета. Такое решение оказалось совершенно не работоспособным, так как при большинстве параметров программа переставала отвечать на запросы пользователя на продолжительное время.

Во имя избегания такого сценария, было решено вынести сами вычисления в отдельный поток, но оставить добавление данных на форму сразу, после их вычисления.

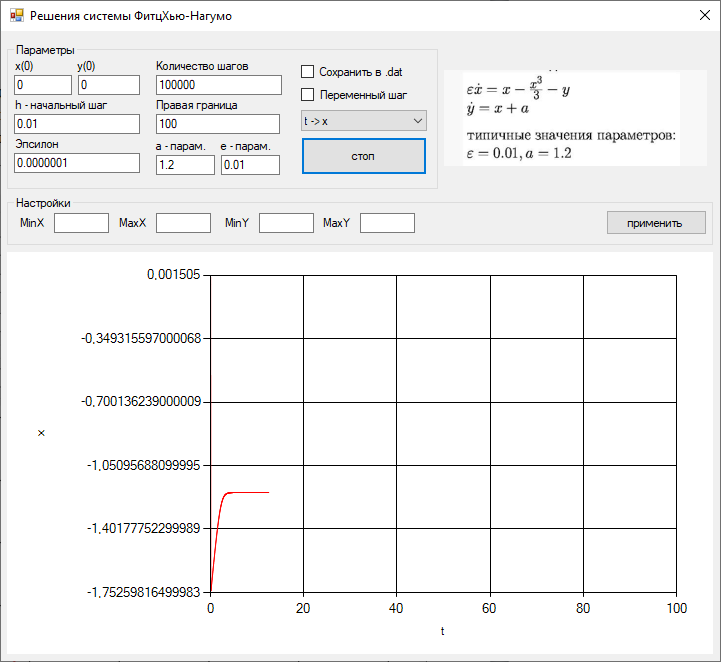
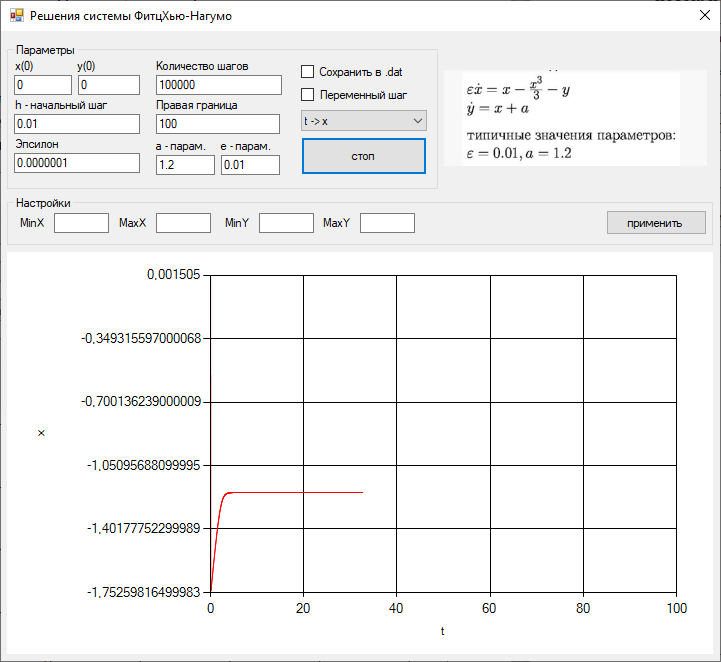


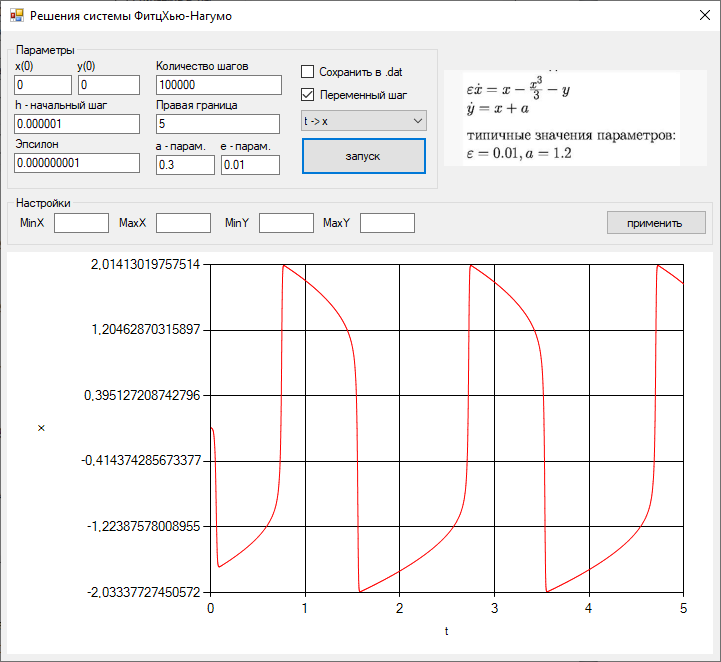
Рис. 3, вывод данных в реальном времени, изображения сделаны с разницей в 5 секунд

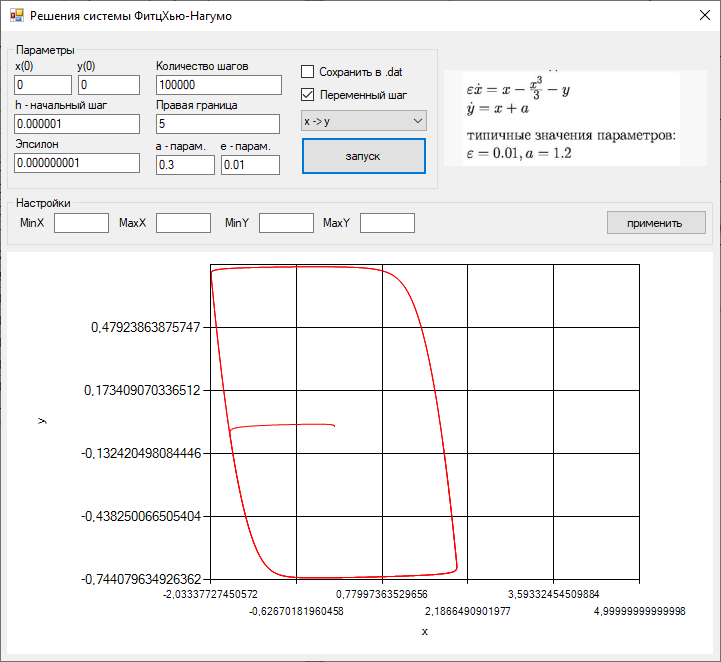
Программа теперь всегда реагирует на управление пользователя, но счет, все еще занимает весьма большой объем времени. Использование переменного шага помогает сократить время расчета, но лишь при определенных параметрах.

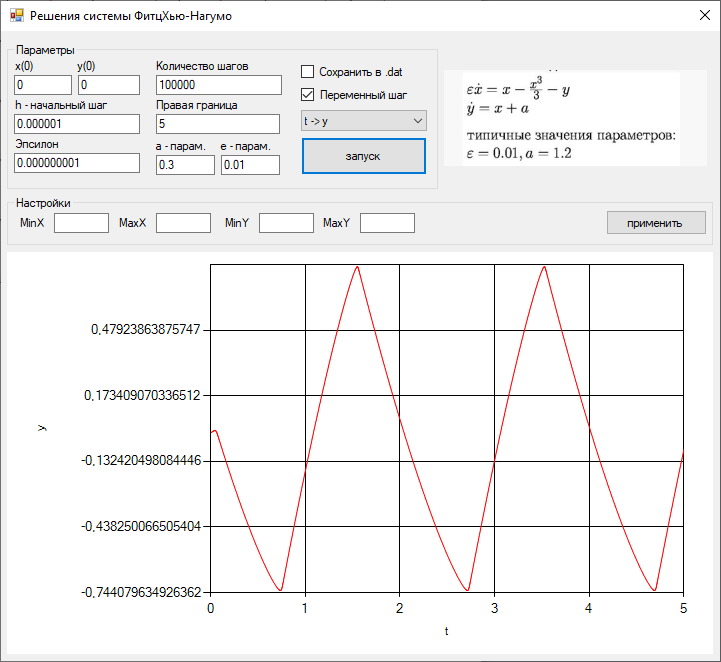
## 2.2 Визуализация посчитанных заранее результатов

Для оптимизации работы программы, был произведен расчет, без вывода данных на экран. Программа завершила метод в разы быстрее, чем при выводе данных в реальном времени. Было принято решение отказаться от вывода данных в форму во время вычисления в угоду производительности. Данные сохраняются в файл, который потом может быть визуализирован для дальнейшего исследования.

Несколько примеров визуализации. На первом рисунке представлен график зависимости y(t), на втором рисунке график x(t), на третьем графике представлена фазовая траектория или график y(x)







Поскольку разработанное приложение позволяет выводить результаты расчета в файл, мы можем построить визуализацию при помощи сторонних приложений.

Например, приложение Graph, в которое поточечно были загружены результаты счета.

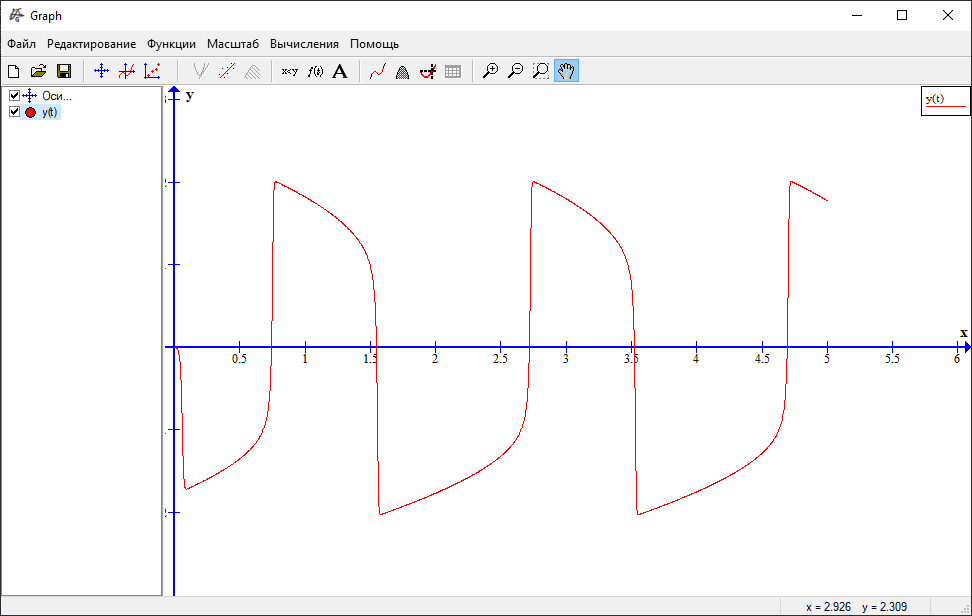


Рис. 7, построение численного решения в программе Graph

# 3. Модель ФитцХью — Нагумо

Модель ФитцХью — Нагумо описывает прототип возбудимой системы. В данном разделе хотелось бы исследовать поведение модели при различных параметрах.

При параметрах системы , мы наблюдем следующее поведение системы

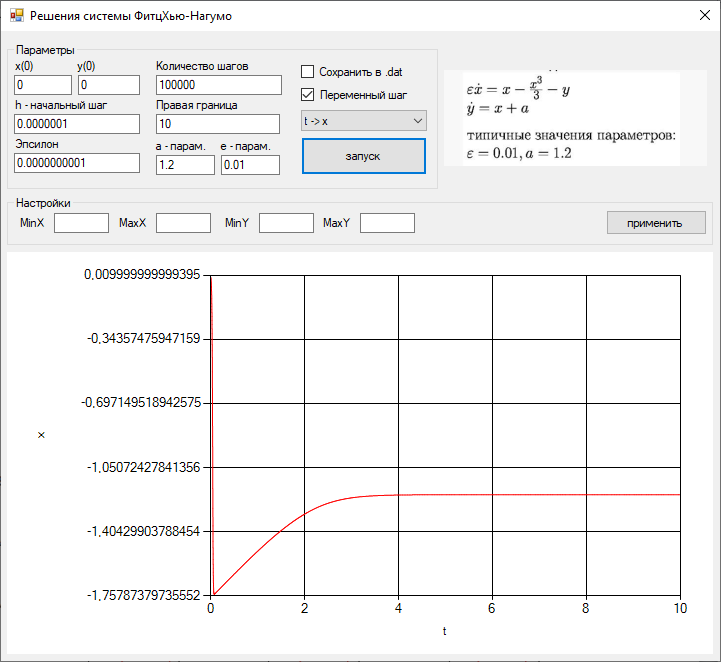


Рис. 8, a=1.2,ε = 0.01

При увеличении система все больше напоминает затухающий осциллятор

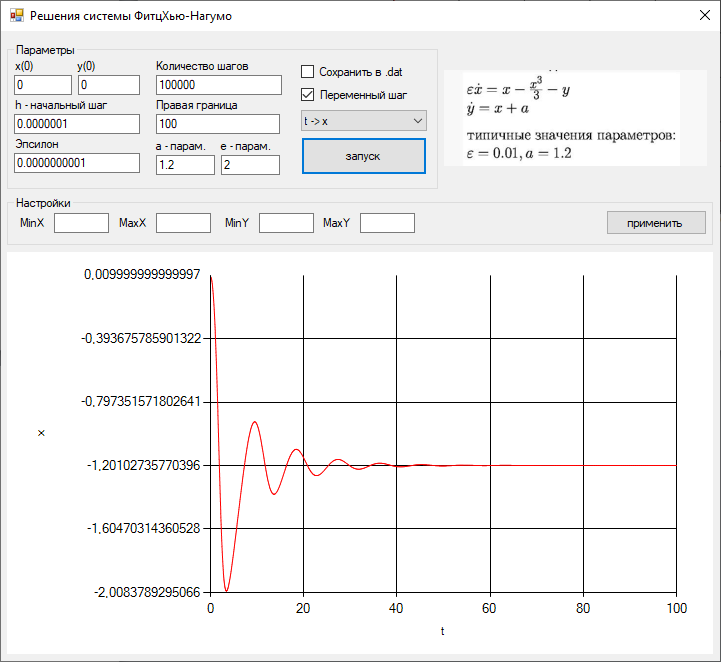


Рис. 9, a=1.2,ε = 2

Это так же прослеживается на фазовой траектории, которая закручивается к точке равновесия, тип которой является фокусом.

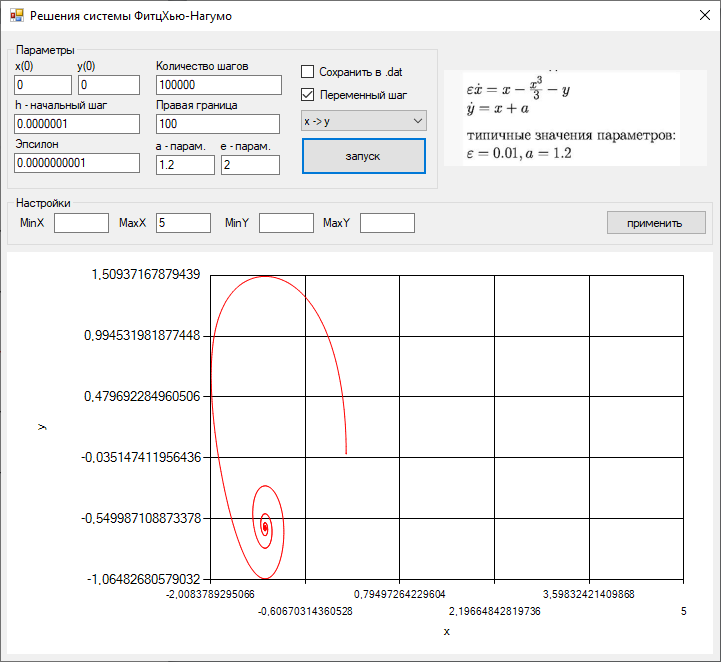


Рис. 10, a=1.2,ε = 2 (фазовая траектория)

При уменьшении же система наоборот начинает проявлять периодичность решения.

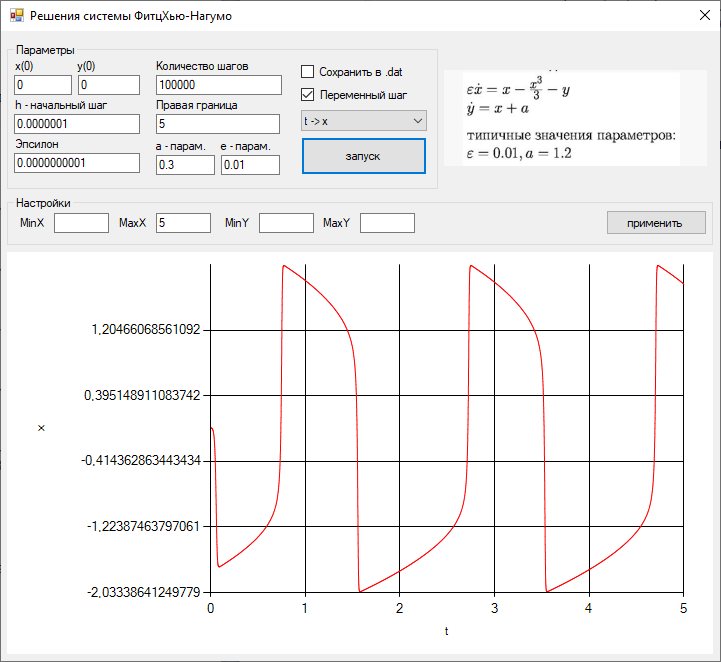


Рис. 11, a=0.3,ε = 0.01

# Приложение

Класс, реализующий метод Рунге-Кутта четвертого порядка

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace core

{

public class MDot

{

public double X { get; set; }

public double U1 { get; set; }

public double U2 { get; set; }

public double Y { get; set; }

}

public class SecondMethod

{

public delegate double Func(double x, double u1, double u2);

public Func Function1 { get; private set; }

public Func Function2 { get; private set; }

MDot point;

double step;

public int C1 { private set; get; } = 0;

public int C2 { private set; get; } = 0;

bool control;

double eps;

public double Step { get => step; }

/// <summary>

/// Создает объект для работы с методом РК(4)

/// </summary>

/// <param name="f">Функция двух пременных</param>

/// <param name="x0">Точка x0</param>

/// <param name="u0">Точка u0</param>

/// <param name="s">Первоначальный шаг</param>

public SecondMethod(Func f1, Func f2, double x0, double u10, double u20, double s, double e, bool ctrl)

{

Function1 = f1;

Function2 = f2;

point = new MDot()

{

X = x0,

U1 = u10,

U2 = u20

};

step = s;

eps = e;

control = ctrl;

}

public MDot nextStep(out double upV, out double len)

{

MDot next = new MDot();

double h = step;

next.X = point.X + h;

double k1 = Function1.Invoke(point.X, point.U1, point.U2);

double l1 = Function2.Invoke(point.X, point.U1, point.U2);

double k2 = Function1.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k1, point.U2 + (h / 2) \* l1);

double l2 = Function2.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k1, point.U2 + (h / 2) \* l1);

double k3 = Function1.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k2, point.U2 + (h / 2) \* l2);

double l3 = Function2.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k2, point.U2 + (h / 2) \* l2);

double k4 = Function1.Invoke(point.X + h, point.U1 + h \* k3, point.U2 + h \* l3);

double l4 = Function2.Invoke(point.X + h, point.U1 + h \* k3, point.U2 + h \* l3);

next.U1 = point.U1 + (h / 6) \* (k1 + 2 \* k2 + 2 \* k3 + k4);

next.U2 = point.U2 + (h / 6) \* (l1 + 2 \* l2 + 2 \* l3 + l4);

MDot half = new MDot();

h = h / 2;

half.X = point.X + h;

k1 = Function1.Invoke(point.X, point.U1, point.U2);

l1 = Function2.Invoke(point.X, point.U1, point.U2);

k2 = Function1.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k1, point.U2 + (h / 2) \* l1);

l2 = Function2.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k1, point.U2 + (h / 2) \* l1);

k3 = Function1.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k2, point.U2 + (h / 2) \* l2);

l3 = Function2.Invoke(point.X + (h / 2), point.U1 + (h / 2) \* k2, point.U2 + (h / 2) \* l2);

k4 = Function1.Invoke(point.X + h, point.U1 + h \* k3, point.U2 + h \* l3);

l4 = Function2.Invoke(point.X + h, point.U1 + h \* k3, point.U2 + h \* l3);

half.U1 = point.U1 + (h / 6) \* (k1 + 2 \* k2 + 2 \* k3 + k4);

half.U2 = point.U2 + (h / 6) \* (l1 + 2 \* l2 + 2 \* l3 + l4);

MDot mes = new MDot();

mes.X = half.X + h;

k1 = Function1.Invoke(half.X, half.U1, half.U2);

l1 = Function2.Invoke(half.X, half.U1, half.U2);

k2 = Function1.Invoke(half.X + (h / 2), half.U1 + (h / 2) \* k1, half.U2 + (h / 2) \* l1);

l2 = Function2.Invoke(half.X + (h / 2), half.U1 + (h / 2) \* k1, half.U2 + (h / 2) \* l1);

k3 = Function1.Invoke(half.X + (h / 2), half.U1 + (h / 2) \* k2, half.U2 + (h / 2) \* l2);

l3 = Function2.Invoke(half.X + (h / 2), half.U1 + (h / 2) \* k2, half.U2 + (h / 2) \* l2);

k4 = Function1.Invoke(half.X + h, half.U1 + h \* k3, half.U2 + h \* l3);

l4 = Function2.Invoke(half.X + h, half.U1 + h \* k3, half.U2 + h \* l3);

mes.U1 = half.U1 + (h / 6) \* (k1 + 2 \* k2 + 2 \* k3 + k4);

mes.U2 = half.U2 + (h / 6) \* (l1 + 2 \* l2 + 2 \* l3 + l4);

upV = mes.U2;

len = 0;

if (control)

{

double s = (mes.U2 - next.U2) / (15);

if (Math.Abs(s) >= eps)

{

C1++;

step /= 2;

return nextStep(out upV, out len);

}

if (Math.Abs(s) <= eps / 15)

{

if (step <= 1e+100)

{

C2++;

step \*= 2;

}

}

double e = 16 \* s;

len = e;

next.U2 = next.U2 + e;

}

point = next;

return point;

}

}

}

## 