МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

Институт информационных технологий, математики и механики
Кафедра дифференциальных уравнений, математического и численного анализа

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

«Численное решение задачи Коши для ОДУ»

Выполнил:
студент группы 381706-1
Денисов В.Л.

Проверил:
ст. преп. каф. ДУМЧА ИИТММ
Эгамов А.И.

Нижний Новгород
2020.

Содержание

1. Введение

Лабораторная работа направлена на изучение вопроса численного решения задачи Коши для ОДУ.

Дифференциальные уравнения – один из наиболее важных инструментов математического моделирования. Основа большинства физических законов сформулирована в их терминах. Законы Ньютона в механике и уравнения Максвелла в теории электромагнитного поля, законы Кирхгофа в теории электрических цепей и уравнение Шредингера в квантовой механике, а также многие другие дифференциальные уравнения или их системы составляют ядро математического аппарата физических исследований.

Аналитическое решение наиболее интересных дифференциальных уравнений, как правило, невозможно. В связи с этим возникает задача численного определения интегральных кривых исследуемых уравнений.

В рамках данной лабораторной работы предлагается к рассмотрению один из двух методов решения начальной задачи Коши: либо метод Рунге-Кутта 3-го или 4-го порядка точности, либо метод Адамса 3-го или 4-го порядка точности.

Мною будет рассмотрен метод Рунге-Кутта 4-го порядка точности, так как он имеет сравнимо хорошую точность, допускает введение переменного шага интегрирования в процессе работы, а также не требует вспомогательных методов для вычисления точек, по которым будут производиться вычисления.

В качестве задачи, для которой требуется найти решение, будет использован математический маятник с диссипацией, описываемый дифференциальным уравнением второго порядка вида $\ddot{x} + \delta\dot{x} + sinx = 0$. Кроме того, аналогичное уравнение возникает в ряде других задач, например, при рассмотрении контакта Джозефсона, при исследовании самоиндуцированной прозрачности в нелинейной оптике, при анализе изгиба упругой балки (так называемая аналогия Кирхгофа). И тот факт, что уравнение маятника охватывает множество различных задач, заставляет видеть в осцилляторе с нелинейностью типа синуса своего рода эталонную модель, заслуживающую подробного исследования.

Для разработки программного комплекса будет использован язык программирования C#, который предоставляет удобные инструменты для написания программ, а также позволяет применять его вместе с .NET Framework, предлагающим API-интерфейс WPF для создания настольных графических программ, имеющих неплохой дизайн и интерактивность.

2. Постановка задачи

В рамках лабораторной работы ставится задача реализации программного комплекса, который позволит находить численное решение задачи Коши для математического маятника с диссипацией, описываемого дифференциальным уравнением второго порядка вида $\ddot{x} + \delta\dot{x} + sinx = 0$.

Программный комплекс должен обладать следующими возможностями:

- Получение входных данных для построения фазовой траектории, а именно: параметр δ в уравнении математического маятника с диссипацией, начальные условия для построения фазовой траектории (то есть для нахождения решения задачи Коши), интервал интегрирования, начальное значение шага интегрирования, а также значение локальной погрешности.
- Вычисление значений, по которым будет построена фазовая траектория, методом Рунге-Кутта 4-го порядка точности с контролем локальной погрешности.
- Вывод на экран полученной фазовой траектории. При этом уже построенные фазовые траектории не должны удаляться, то есть должна быть поддержка построения фазового портрета.
- Сброс всех значений, которые использовались для построения фазовой траектории, и очистка фазовой плоскости без перезапуска программы.

Программный комплекс будет предоставлять:

1. Пользовательский интерфейс для работы с программой.
2. Необходимый набор функций, для построения фазовых траекторий.

3. Теоретическая часть

3.1 Метод Рунге-Кутта 4-го порядка

Методы Рунге-Кутта – большой класс численных методов решения задачи Коши для обыкновенных дифференциальных уравнений и их систем. Первые методы данного класса были предложены около 1900 года немецкими математиками К. Рунге и М. В. Куттой.

Наиболее часто используется и реализован в различных математических пакетах (Maple, MathCAD, Maxima) классический метод Рунге-Кутты, имеющий четвёртый порядок точности. Именно он и будет рассмотрен ниже.

В нашем случае имеем однородное дифференциальное уравнение 2-го порядка вида $\ddot{x} + \delta\dot{x} + sinx = 0$ с начальными условиями $x(a) = 0$ и $\dot{x}(a) = 0$ на интервале интегрирования $(a; b)$.

Сведем это уравнение к системе двух уравнений первого порядка с помощью замены переменных:

$$\begin{cases} \dot{x} = v = f \\ \dot{v} = -\delta v - sinx = g \end{cases}$$

Для решения этой системы воспользуемся методом Рунге-Кутта 4-го порядка, который представляет собой следующий набор действий. Для вычислений потребуются следующие рекуррентные формулы:

$$\begin{cases} x_{i+1} = x_i + \Delta x_i \\ v_{i+1} = v_i + \Delta v_i \end{cases}$$

где $\Delta x_i = \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ и $\Delta v_i = \frac{h}{6}(l_1 + 2l_2 + 2l_3 + l_4)$

Коэффициенты $k_{1,2,3,4}$ и $l_{1,2,3,4}$ вычисляются на каждом шаге по формулам:

$$k_1 = f(x, v) = v, \qquad l_1 = g(x, v) = -\delta v - sinx$$

$$k_2 = f\left(x + \frac{hk_1}{2}, v + \frac{hl_1}{2}\right) = v + \frac{hl_1}{2}, \qquad l_2 = g\left(x + \frac{hk_1}{2}, v + \frac{hl_1}{2}\right)$$

$$k_3 = f\left(x + \frac{hk_2}{2}, v + \frac{hl_2}{2}\right) = v + \frac{hl_2}{2}, \qquad l_3 = g\left(x + \frac{hk_2}{2}, v + \frac{hl_2}{2}\right)$$

$$k_4 = f(x + hk_3, v + hl_3) = v + hl_3, \qquad l_4 = g(x + hk_2, v + hl_3)$$

3.2 Контроль локальной погрешности

Одной и той же погрешности можно достичь, если ввести интегрирование с большим шагом, где решение изменяется плавно, и с малым шагом – в области резкого изменения решения.

Возникает вопрос, как определять величину шага?

Для этих целей чаще всего применяется алгоритм выбора шага с помощью удвоения и деления шага пополам.

Пусть S – оценка локальной погрешности метода на шаге h в точке $x_i + h$. При этом для нахождения этой оценки используется следующая формула:

$$S = \frac{\widetilde{x_{i+1}} - x_{i+1}}{2^p - 1}$$

где $\widetilde{x_{i+1}}$ – значение в следующей точке, полученное за 2 шага с длины $h/2$, а x_{i+1} – значение в следующей точке, полученное за 1 шаг с длины h.

Положим в качестве локальной погрешности значение ε, а также $M = 2^p$ – некоторую константу, где p – порядок аппроксимации метода. Тогда возможно несколько случаев:

1. $|S| > \varepsilon$ – тогда нужно выбрать новый шаг вдвое меньше прежнего ($h = h/2$), повторить расчет заново и найти новую оценку S. Так продолжается до тех пор, пока при какой-то величине шага оценка локальной погрешности не будет меньше или равной ε.
2. $\varepsilon/M < |S| \leq \varepsilon$ – шаг оставить неизменным и перейти к расчету следующей точки.
3. $|S| < \varepsilon/M$ – положить шаг вдвое больше прежнего $h = 2h$ и перейти к расчету следующей точки.

В случае метода Рунге-Кутта 4-го порядка значение $M = 32$, т.к. его порядок аппроксимации $p = 5$.

3. Руководство пользователя и пример работы с программой

При запуске программы перед пользователем появляется интерфейс управления.

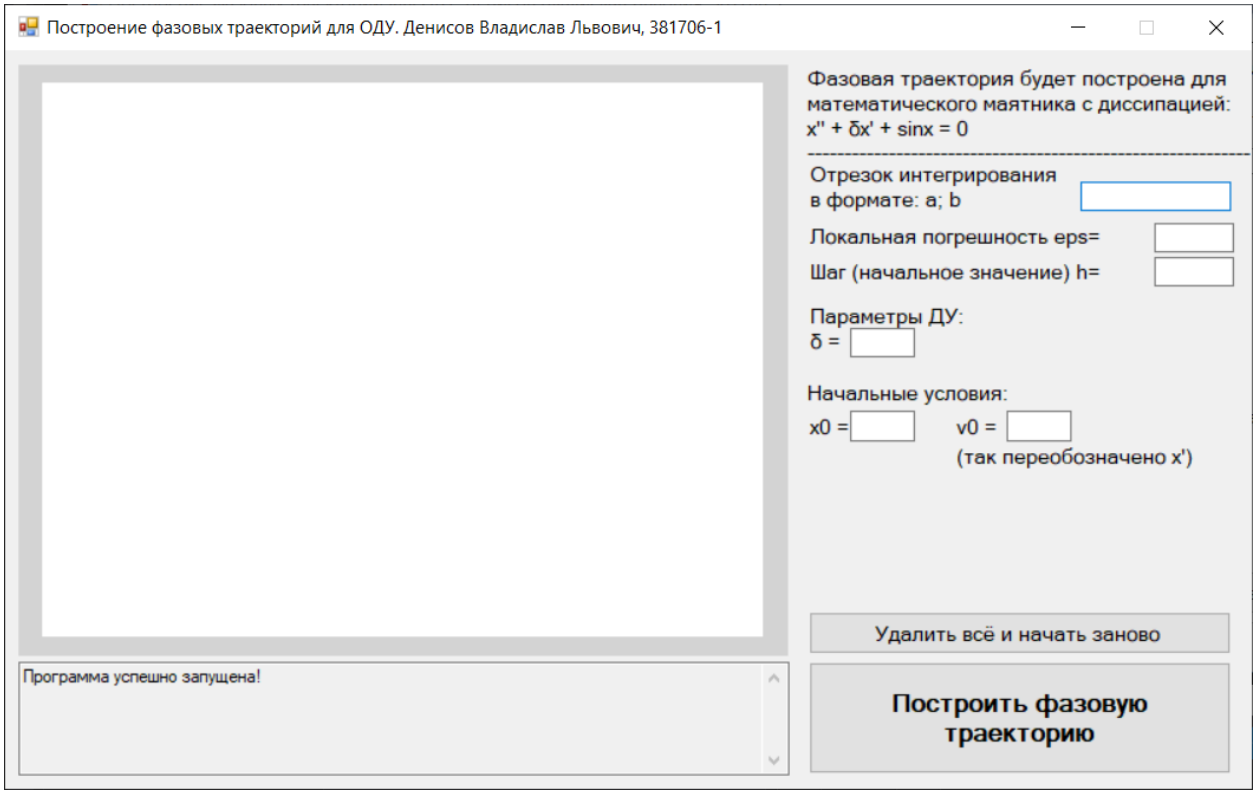

Рисунок 1 Первый запуск программы

Рассмотрим пример работы с программой.

Сначала заполним все поля, интересующими нас значениями. Это можно сделать путем ввода значения в соответствующее поле и нажатия на клавишу “Enter” на клавиатуре для подтверждения. Ввод начальных условий поддерживается кликом мыши по фазовой плоскости.

Все действия сопровождаются информационными сообщениями в специальном окне лога. Поэтому в случае ввода недопустимого значения программа сообщит об этом и даст возможность повторить действие.

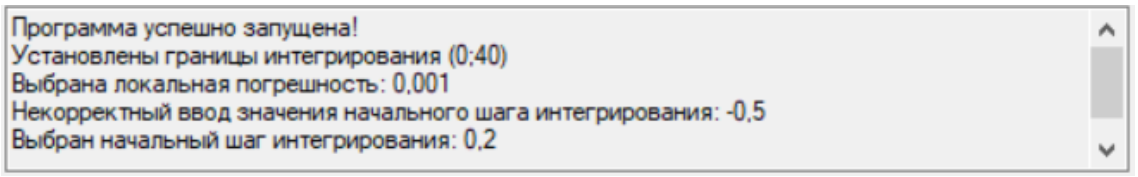

Рисунок 2 Ввод недопустимого значения

После заполнения всех полей следует нажать на кнопку «Построить фазовую траекторию». Результат можно увидеть на Рисунке 3.

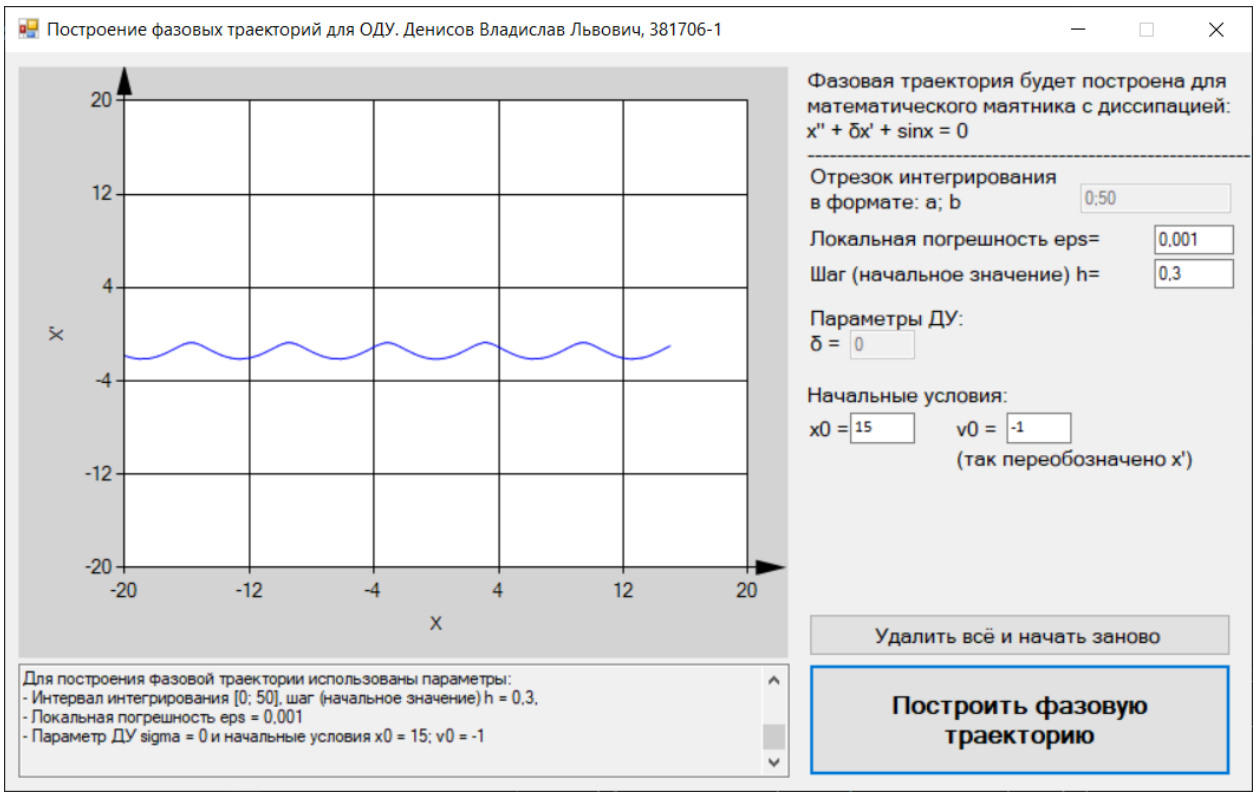

Рисунок 3 Построение фазовой траектории

Заметим, что после ввода отрезка интегрирования и параметра δ соответствующие поля нельзя редактировать. Такой подход обеспечивает построение корректного фазового портрета. Добавим еще несколько фазовых траекторий и убедимся в этом.

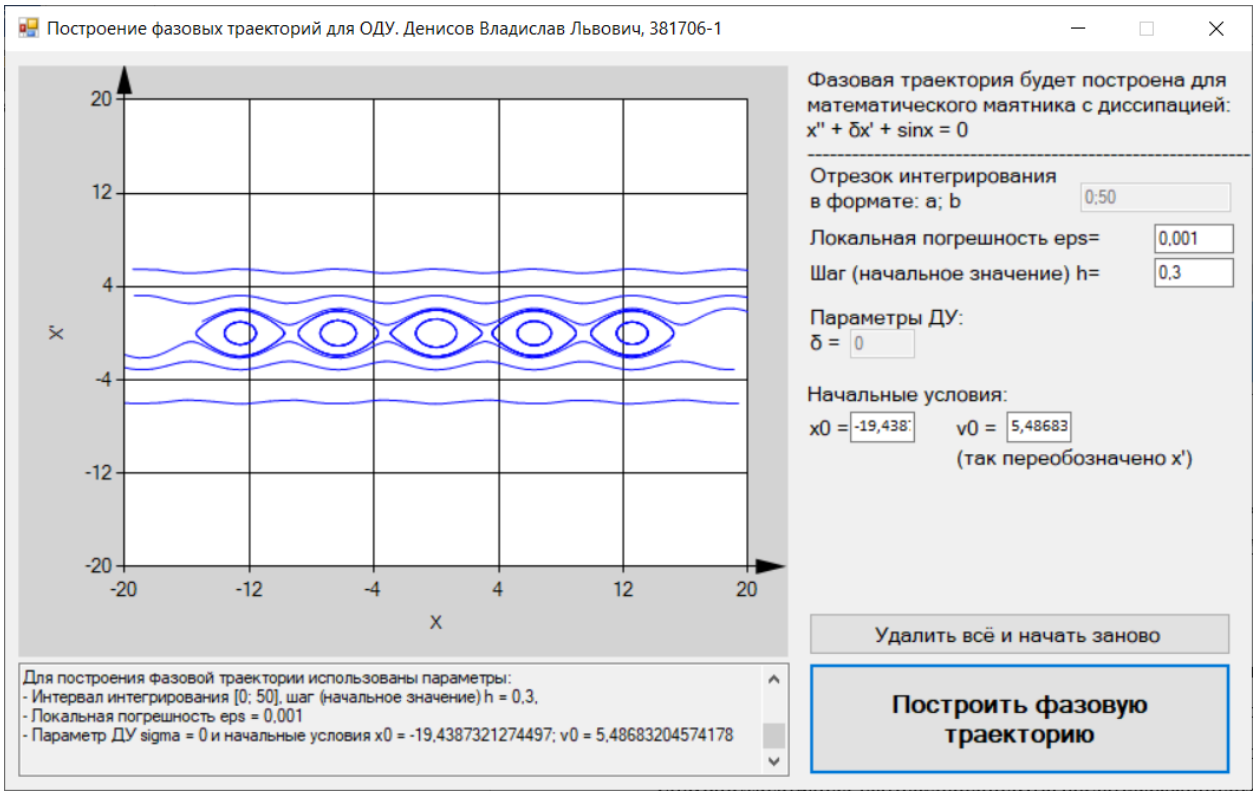

Рисунок 4 Построение фазового портрета

При желании можно построить фазовый портрет для линейного маятника с диссипацией при другом значении параметра δ и на другом интервале интегрирования. Для этого следует нажать на кнопку «Удалить всё и начать заново». Будет очищена фазовая плоскость и сброшены все параметры, которые были введены ранее.

Пример для других параметров на Рисунке 5.

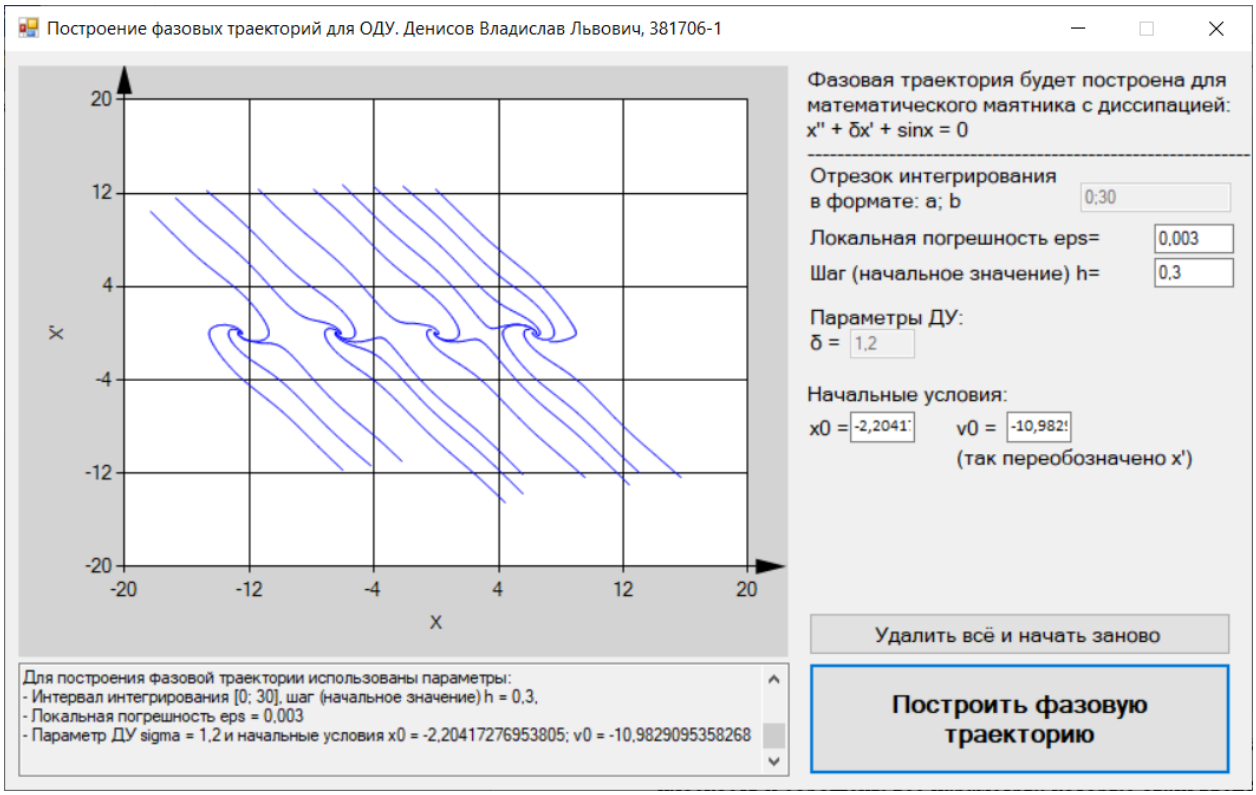

Рисунок 5 Построение фазового портрета (другой случай)

4. Заключение

В результате лабораторной работы разработан программный комплекс, который позволяет найти численное решение задачи Коши для дифференциального уравнения второго порядка, представляющего математический маятник с диссипацией.

Программа выполняет построение фазовой траектории с учетом начальных условий для решения задачи Коши, которые задаются пользователем вручную. Для построения фазового портрета автоматически фиксируются параметр δ в уравнении математического маятника с диссипацией, а также границы интервала интегрирования.

В результате работы программы на экране отображается одна (или несколько, если так захочет пользователь) фазовых траекторий.

Цели, поставленные в лабораторной работе, успешно достигнуты.

5. Литература

1. Мышенков В.И., Мышенков Е.В. Численные методы. Ч. 2. Численное решение обыкновенных дифференциальных уравнений: Учебное пособие для студентов специальности 073000. – М.:МГУЛ, 2005. – 109 с.: ил..
2. Баркалов К.А. Образовательный комплекс «Параллельные численные методы». Лекционные материалы. Н. Новгород, 2011.
3. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Метод_Рунге_—_Кутты

6. Приложение

В данном разделе находится листинг основных функций, которые используются в программе.

```
// Метод Рунге-Кутта 4-го порядка
private void RungeCutte()
{
    // Увеличиваем счетчик построенных фазовых траекторий.
    numSpline++;

    // Создаем новое техническое имя для графика, добавляем его объект на плоскость,
    устанавливаем цвет и тип – кривая.
    string nameSeries = numSpline.ToString();
    chart.Series.Add(nameSeries);
    chart.Series[nameSeries].Color = Color.Blue;
    chart.Series[nameSeries].ChartType = SeriesChartType.Spline;

    // Вычисляем число шагов для выполнения метода Р-К.
    numberSteps = (int)((intervalB - intervalA) / h);

    // В качестве первой точки берем начальные условия, добавляем точку к кривой,
    описывающей фазовую траекторию.
    double x = x0;
    double v = v0;
    chart.Series[nameSeries].Points.AddXY(x, v);

    // Значение шага сохраняем в локальную переменную, чтобы не испортить его для
    построения последующих траекторий.
    double localH = h;
    // Выполняем шаги метода Р-К.
    for (int i = 0; i <= numberSteps; ++i)
    {
        // Выполняем оценку локальной погрешности согласно теоретическим выкладкам.
        double[] oneStepWithH = calcOnOneStepWithH(x, v, localH);
        double[] twoStepWithHdiv2 = calcOnTwoStepWithH(x, v, localH / 2);
        double localEpsControlValue = (twoStepWithHdiv2[0] - oneStepWithH[0]) / 15;

        // Выполняем проверку условий для динамического изменения шага.
        if (System.Math.Abs(localEpsControlValue) > eps)
        {
                localH = localH / 2;
                while (System.Math.Abs(localEpsControlValue) > eps)
                {
                        oneStepWithH = calcOnOneStepWithH(x, v, localH);
```

```
                    twoStepWithHdiv2 = calcOnTwoStepWithH(x, v, localH / 2);
                    localEpsControlValue = (twoStepWithHdiv2[0] - oneStepWithH[0]) / 15;
                }
            }
            else if (System.Math.Abs(localEpsControlValue) < eps / 32)
            {
                localH = 2 * localH;
            }

            // Сохраняем очередную найденную точку фазовой траектории и добавляем ее к кривой.
            x = oneStepWithH[0];
            v = oneStepWithH[1];
            chart.Series[nameSeries].Points.AddXY(x, v);

            // Выполняем пересчет числа шагов, т.к. значение шага могло измениться.
            numberSteps = (int)((intervalB - intervalA) / localH);
        }
    }

    // Один шаг метода Р-К с величиной h
    private double[] calcOnOneStepWithH(double _x, double _v, double _h)
    {
            double[] result = new double[2];

            double k1, k2, k3, k4;
            double l1, l2, l3, l4;
            k1 = _v;
            l1 = func(_x, _v);

            k2 = _v + (_h / 2) * l1;
            l2 = func(_x + (_h / 2) * k1, _v + (_h / 2) * l1);

            k3 = _v + (h / 2) * l2;
            l3 = func(_x + (_h / 2) * k2, _v + (_h / 2) * l2);

            k4 = _v + _h * l3;
            l4 = func(_x + _h * k3, _v + _h * l3);

            result[0] = _x + _h / 6 * (k1 + 2 * k2 + 2 * k3 + k4);
            result[1] = _v + _h / 6 * (l1 + 2 * l2 + 2 * l3 + l4);

            return result;
    }

    // Два шага метода Р-К с величиной h
    private double[] calcOnTwoStepWithH(double _x, double _v, double _h)
    {
            double[] result = new double[2];
            result = calcOnOneStepWithH(_x, _v, _h);
            result = calcOnOneStepWithH(result[0], result[1], _h);
            return result;
    }


    // Вспомогательная функция для замены x’ = v
    private double func(double x, double v)
    {
            return -1 * (sigma * v + Math.Sin(x));
    }
```