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

ЛАБОРАТОРНАЯ РАБОТА

Численное решение задачи Коши для ОДУ

Выполнил:
Антипин Александр Сергеевич 381706-2
___________ Подпись

Проверил:
Ассистент кафедры ДУМЧА:
Морозов Кирилл Евгеньевич

___________ Подпись

Нижний Новгород

2020

Оглавление

Введение

Дифференциальное уравнение представляет собой математическое уравнение, которое относится к функции и её производным. В прикладной математике функции как правило представляют собой физические величины, а производные скорости их изменения. Дифференциальные уравнения в свою очередь определяют отношения между ними. Поэтому дифференциальные уравнения играют важную роль в различных дисциплинах, включая инженерию, физику, химию, экономику и биологию.

В чистой математике дифференциальные уравнения изучаются с разных точек зрения, большинство из которых касается множества решений функций, удовлетворяющих начальному условию. Только самые простые дифференциальные уравнения могут быть решены с помощью явных формул, однако некоторые свойства решений дифференциального уравнения могут быть определены и без нахождения их точного вида.

Если точное решение не может быть найдено, оно может быть получено численно путём приближения с использованием компьютеров. Теория динамических систем подчёркивает качественный анализ систем, описываемых дифференциальными уравнениями, в то время как многие численные методы были разработаны для определения решений с определённой степенью точности.

Теория

Задача Коши (исходная задача) - задача, заключающаяся в нахождении конкретной функции, удовлетворяющей заданному дифференциальному уравнению и начальному условию. В случае уравнения первой степени начальным условием будет точка, через которую должен пройти график искомой функции. В случае уравнения второй степени начальная задача дополнительно будет содержать значение первой производной в данной точке и аналогично в случае уравнений более высокого уровня.

Теорема существования и единственности утверждает, что если функция $f(x,y)$ дифференцируема по y в окрестности точки плоскости (x_0, y_0) и её производная также дифференцируема в этой точке, то задача Коши однозначно разрешима в окрестности данной точки. Если функция дифференцируема на всем отрезке $[a, b]$, то решение задачи Коши также существует на всем этом отрезке. При численном решении задач чаще всего требуется найти решение для значений $x > x_0$, т.е. первый параметр задачи Коши будет x_0. Одним из методов численного решения дифференциальных уравнений является метод **Рунге-Кутты**.

Метод Рунге-Кутты

Метод Рунге-Кутты, который, вообще говоря, является более точным и имеет большее практическое значение чем метод Эйлера, хотя идея метода схожа. Приращение функции приближённо выражается в виде линейной комбинации дифференциалов, вычисленных в текущем узле и в окрестных точках узла.

Рассмотрим уравнение второго порядка, разрешённое относительно второй производной:

$$y'' = f(x, y, y')$$

На отрезке [a, b] с начальным условием:

$$y(x_0) = y_0, y'(x_0) = y'_0$$

Это уравнение легко свести к системе уравнений первого порядка с помощью замены переменных. Тогда и уравнение сводится к системе первого порядка.

$$y' = z \qquad y'' = z'$$

$$\begin{cases} y' = f_1(x, y, z) \\ z' = f_2(x, y, z) \end{cases} \qquad y(x_0) = y_0,\ z(x_0) = z_0$$

Напишем формулы для решения системы двух уравнений методом Рунге-Кутты:

k1 = z[i]
q1 = f(x[i], y[i], z[i])

k2 = z[i] + q1h/2
q2 = f(x[i] + h/2, y[i] + k1h/2, z[i] + q1h/2)

k3 = z[i] + q2h/2
q3 = f(x[i] + h/2, y[i] + k2h/2, z[i] + q2h/2)

k4 = z[i] + q2h
q4 = f(x[i] + h, y[i] + k3h, z[i] + q3h)

y[i+1] = y[i] + h(k1 + 2k2 + 2k3 + k4)/6
z[i+1] = z[i] + h(q1 + 2q2 + 2*q3 + q4)/6

Где h является приращением аргумента. Приведённые формулы *k1*, *q1*, *k2*, *q2*, *k3*, *q3*, *k4*, *q4* последовательно вычисляются на каждом шаге, после чего вычисляются *y[i+1]* и *z[i+1]*.

Выбор модели

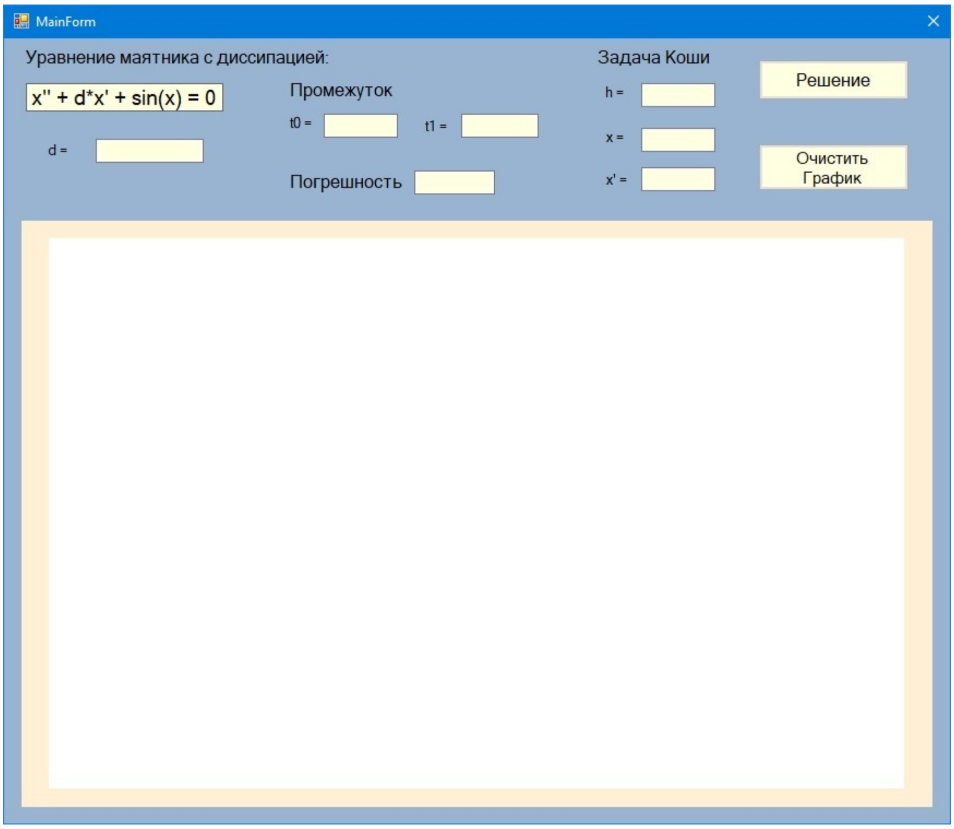
В данной лабораторной работе я решил рассмотреть уравнение маятника с диссипацией: $x'' + dx' + \sin(x) = 0$.

Выбор языка

Для создания программы использовался язык высокого уровня C++ с фреймворком Windows Forms для обеспечения визуального представления.

Программная реализация

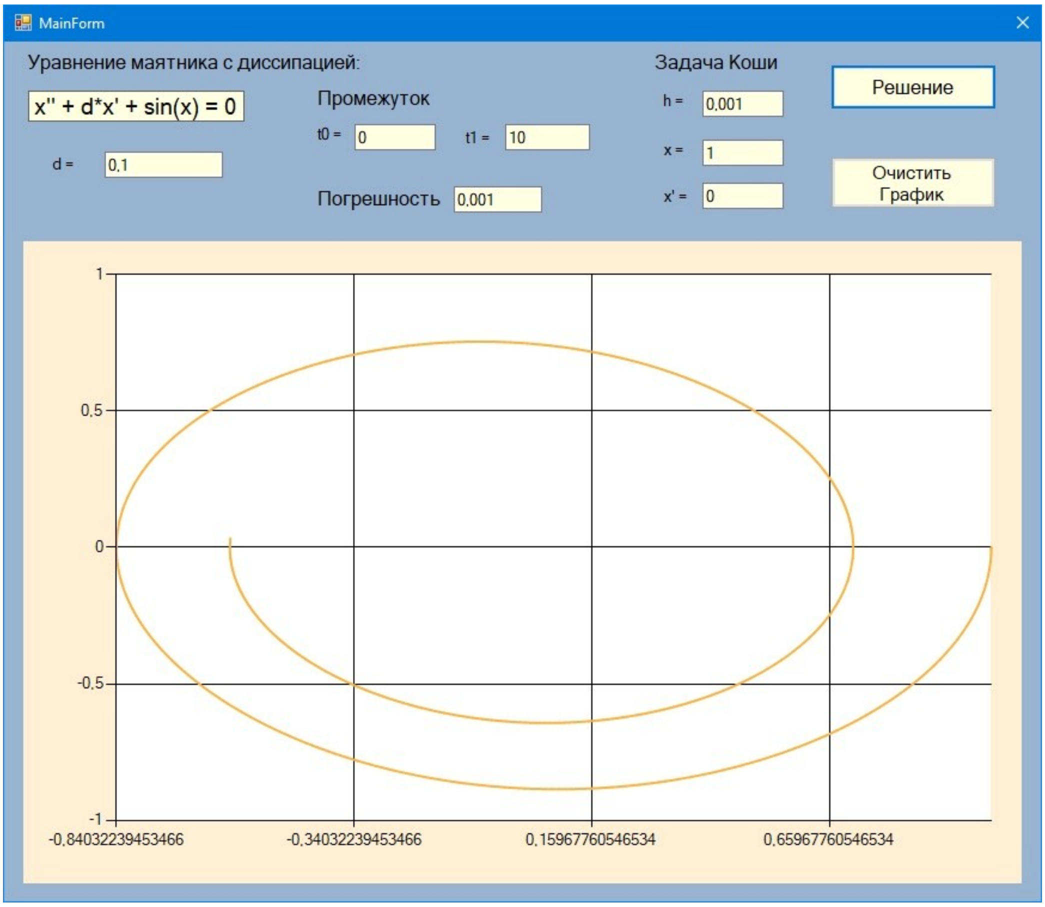
На следующем рисунке приведён интерфейс программы, где:



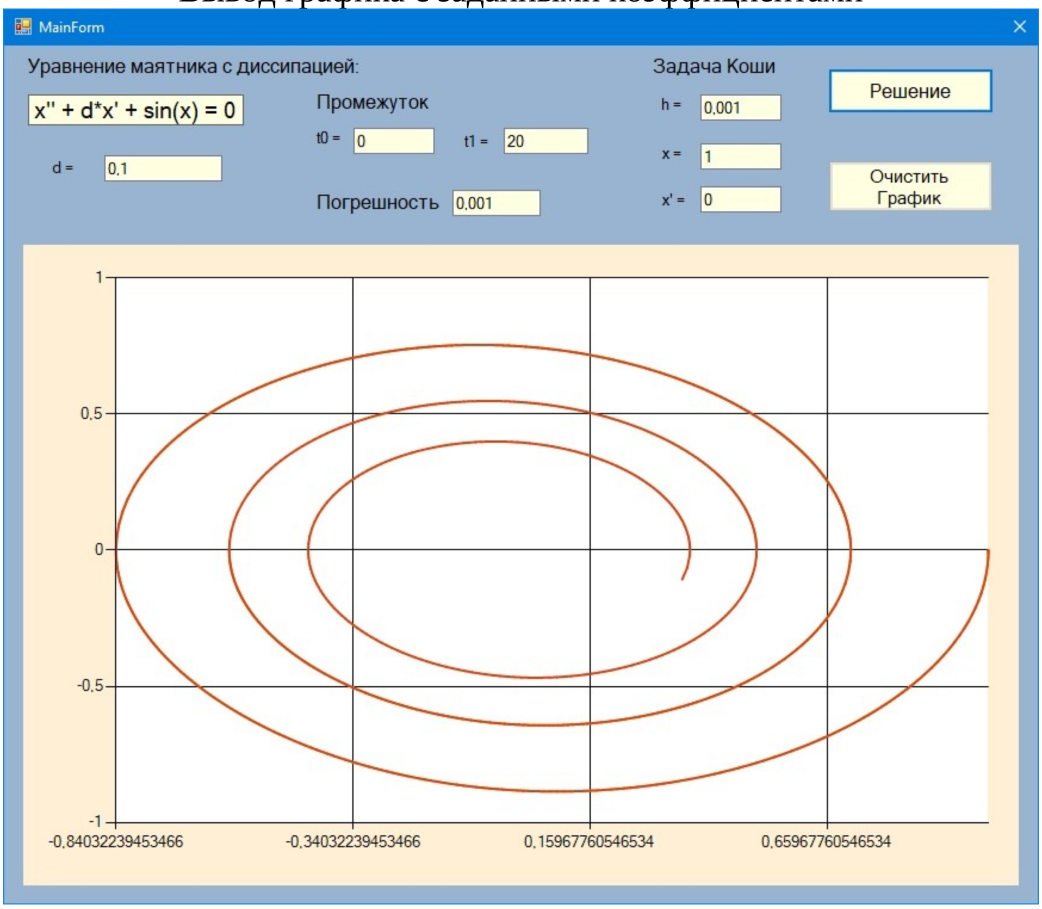
слева сверху приведён общий вид уравнения маятника с диссипацией. Под ним можно записать коэффициент уравнения, которое будет решаться в данный момент. Слева можно ввести промежуток на котором будут производиться вычисления, а под ним есть поле для ввода погрешности вычислений. Справа представлены поля для ввода задачи Коши, а также поле задания шага свободной переменной с которым будут происходить вычисления (шаг может быть автоматически изменен в зависимости от выбранной погрешности).

Также в программе предусмотрена функция продолжения фазовой траектории по нажатию клавиши R.

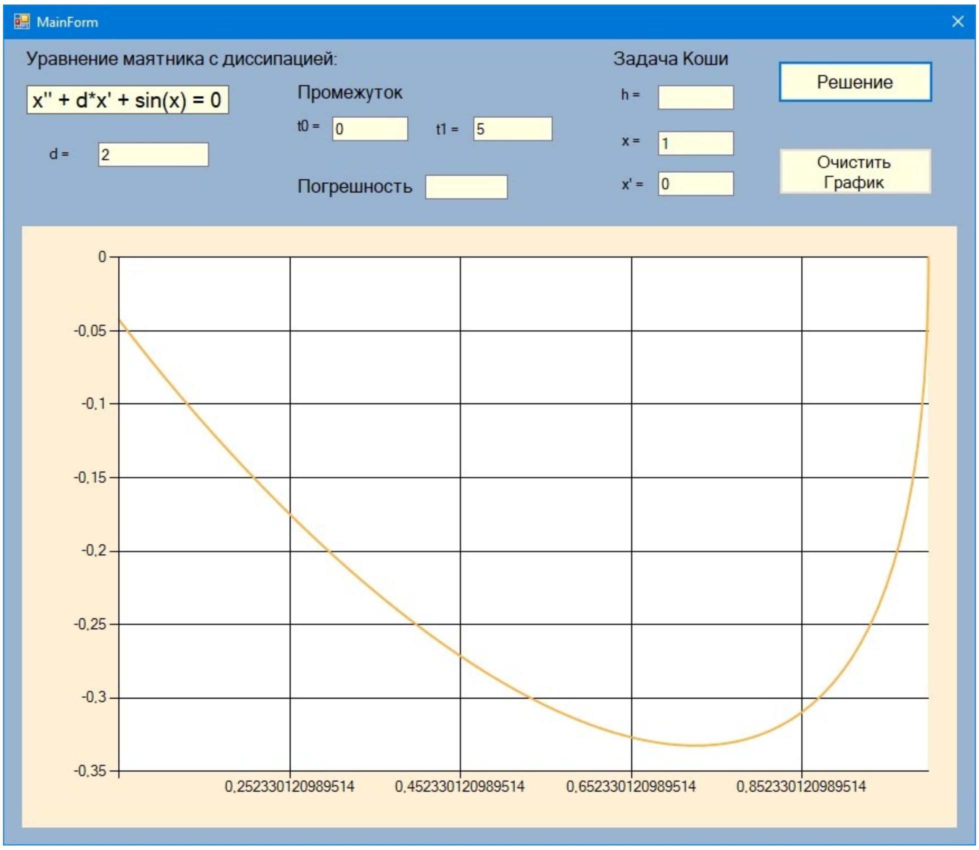
Далее представлены несколько скриншотов вывода графиков в зависимости от заданных начальных условий.



Вывод графика с заданными коэффициентами



Продолжение фазовой траектории по нажатию R



Погрешность и шаг можно не задавать, у них есть значение по умолчанию

Вывод

В ходе работы было сделано несколько выводов: метод Рунге-Кутты не сложен в реализации, и имеет высокую точность. Ошибка на конечно интервале интегрирования имеет порядок O(h4), где h – расстояние между узлами, в которых вычисляется искомая функция. Если известен класс функций, которые могут стоять в правой части задачи Коши, то алгоритм можно эффективно модифицировать, чтобы избавиться от лишних пересылок данных. Еще к достоинствам этого метода можно добавить то, что он является явными, т.е. значения yi+1 находится по ранее найденным значениям y1,y2, … yi.

Список использованной литературы

- А.А.Самарский. Введение в численные методы М.: Наука, 1982.
- https://ru.wikipedia.org/wiki/Дифференциальное_уравнение
- https://portal.tpu.ru/SHARED/l/LOPATKIN/Students/DG/9-Problems_1-3.pdf
- https://algowiki-project.org/ru/Участник:Anlesnichiy Решение_задачи_Коши_для_системы_ОДУ_методом_Рунге-Кутты_4_порядка

Листинг основных функций

```
// Уравнение маятника с диссипацией
double func(double x, double y, double z) {
   return -coef1 * z - sin(y);
}

// Рунге-Кутт
void Runge_Kutt(std::vector<double>& x, std::vector<double>& y, std::vector<double>& z,
   double t0, double t1, double h, double eps) {
   size_t i = 0;
   while (t0 < t1) {
      // вычисление коэффициентов
      double k1 = h * z[i];
      double q1 = h * func(x[i], y[i], z[i]);

      double k2 = h * (z[i] + q1 / 2);
      double q2 = h * func(x[i] + h / 2, y[i] + k1 / 2, z[i] + q1 / 2);

      double k3 = h * (z[i] + q2 / 2);
      double q3 = h * func(x[i] + h / 2, y[i] + k2 / 2, z[i] + q2 / 2);

      double k4 = h * (z[i] + q2);
      double q4 = h * func(x[i] + h, y[i] + k3, z[i] + q3);

      double yt = y[i] + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
      double zt = z[i] + (q1 + 2 * q2 + 2 * q3 + q4) / 6;

      if (eps > 0)
      {
         h /= 2; // уменьшаем шаг в 2 раза и проходим раза по промежутку
         double ux_temp = y[i];
         double uy_temp = z[i];
         for (int j = 0; j < 2; j++)
         {
            double uk1 = h * z[i];
            double ul1 = h * func(x[i], y[i], z[i]);

            double uk2 = h * (z[i] + q1 / 2);
            double ul2 = h * func(x[i] + h / 2, y[i] + k1 / 2, z[i] + q1 / 2);

            double uk3 = h * (z[i] + q2 / 2);
```

```
            double ul3 = h * func(x[i] + h / 2, y[i] + k2 / 2, z[i] + q2 / 2);

            double uk4 = h * (z[i] + q2);
            double ul4 = h * func(x[i] + h, y[i] + k3, z[i] + q3);

            ux_temp += (k1 + 2 * k2 + 2 * k3 + k4) / 6;
            uy_temp += (q1 + 2 * q2 + 2 * q3 + q4) / 6;
        }

        double S = abs((yt - ux_temp) / 15); // высчитываем погрешность
        if (S > eps) {
            continue;
            //если погрешность не устраивает, уменьшаем шаг, считаем заново
            //добавления точек при этом не происходит и i не увеличивается
        }
              else if (S < eps / 32) { //если вычисления слишком точные, то
увеличиваем шаг
            h *= 2; // шаг можно увеличить
        }
            h *= 2; // т.к. мы перед этим уменьшали в 2 раза, т.е. возвращаем в
исходное
    }

    x.push_back(x[i] + h);
    y.push_back(yt);
    z.push_back(zt);
    ++i;
    t0 += h;
  }
}


private: System::Void button1_Click(System::Object^ sender, System::EventArgs^
e) {
          // Проверка на заполнение необходимых текстовых полей
          if (this->textBox1->Text == "" || this->textBox5->Text == "" ||
                  this->textBox6->Text == "" || this->textBox8->Text == "" ||
                  this->textBox9->Text == "") {
                  MessageBox::Show("Введены не все данные");
                  return;
          }
          // считывание данных
          double coef1 = Convert::ToDouble(this->textBox1->Text);
          double x0 = Convert::ToDouble(this->textBox5->Text);
```

```
double t1 = Convert::ToDouble(this->textBox6->Text);
if (this->textBox7->Text != "")
	h = Convert::ToDouble(this->textBox7->Text);
double y0 = Convert::ToDouble(this->textBox8->Text);
double z0 = Convert::ToDouble(this->textBox9->Text);
if (this->textBox2->Text != "")
	pogr = Convert::ToDouble(this->textBox2->Text);
if (x0 > t1) {
	MessageBox::Show("Неправильный интервал");
	return;
}
t0 = x0;
tf = t1;
_x.Add(x0);
_y.Add(y0);
_z.Add(z0);
coef.Add(coef1);

size_t count = (t1 - x0) / h + 1;
std::vector<double> x(1), y(1), z(1);
x[0] = x0;
y[0] = y0;
z[0] = z0;
init(coef1);
// применение метода Рунге-Кутты
Runge_Kutt(x, y, z, x[0], t1, h, pogr);

// Отрисовка графика
++countC;
System::String^ str("0" + countC);
System::Windows::Forms::DataVisualization::Charting::Series^ ser1
= chart1->Series->Add(str);
ser1->BorderWidth = 2;
ser1->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;
ser1->IsVisibleInLegend = false;
ret.Add(ser1);
for (size_t i = 0; i < x.size(); ++i) {
	chart1->Series[str]->Points->AddXY(y[i], z[i]);
}
x.clear();
y.clear();
z.clear();
}
```