**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Теория разностных схем.

**Отчет по лабораторной работе № 1**

**Тема:** «Решение начальных и краевых задач для обыкновенных дифференциальных уравнений»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа МКН-315 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Халиков А.Р |  |  |  |
| Принял | Белевцов Н.С. |  |  |  |

**Уфа 2022**

**Цель работы:** получить навык численного решения задач для обыкновенных дифференциальных уравнений с использованием различных методов на примере задачи Коши для системы обыкновенных дифференциальных уравнений первого порядка и начально-краевой задачи для обыкновенного дифференциального уравнения второго порядка.

**Теоретический материал**

Задача Коши для системы ОДУ

Рассматривается задача Коши для системы уравнений движения материальной точки в потенциальном поле U(x):

Рассматриваемые численные методы решения

1. Метод Эйлера с постоянным шагом.
2. Явная двухшаговая схема Адамса.
3. Метод Рунге-Кутта 4-го порядка.

Метод Эйлера.

Рассмотрим задачу Коши для ОДУ первого порядка

Обозначим за h шаг сетки и грубо аппроксимируем приращение функции:

Подставляя исходное уравнение в правую часть и заменяя переменную и функцию сеточными функциями, получим схему для метода Эйлера:

Метод Адамса

Аппроксимируем приращение функции более точно:

Экстраполируем функцию линейно по уже известным двум значениям в точках и и проинтегрируем. После преобразования получим явную схему Адамса второго порядка:

Так как схема для расчета нового значения требует два предыдущих, дополним начальное условие значением, посчитанным, например, методом Эйлера:

Метод Рунге-Кутты 4 порядка

Данный метод строит четырехчленную схему на основе разложения функции погрешности в ряд Тейлора и приравнивания первых четырех ее производных к нулю.

Наиболее употребительная схема:

**Задания на лабораторную работу**

***I. Задача Коши для системы уравнений движения***

**Приведение системы к безразмерному виду:**

В первых трех заданиях решается следующая задача Коши:

Где

.

Введем замену переменных:

где . При такой замене система принимает вид:

***Задача 1 (1 балл).***

1. Написать вычислительную программу на языке программирования C++ решения задачи (1) методом Эйлера с постоянным шагом.
2. Исследовать зависимость решения при больших временах от величины шага временной сетки. Построить графики решений для различных значений шага.
3. Выполнить сравнение полученных решений с численным решением в каком-либо математическом пакете, полученным с помощью метода высокого порядка точности (например, Рунге-Кутта 4–5). Построить графики разности решений.
4. Проверить применимость правила Рунге и с его помощью повысить точность решения.

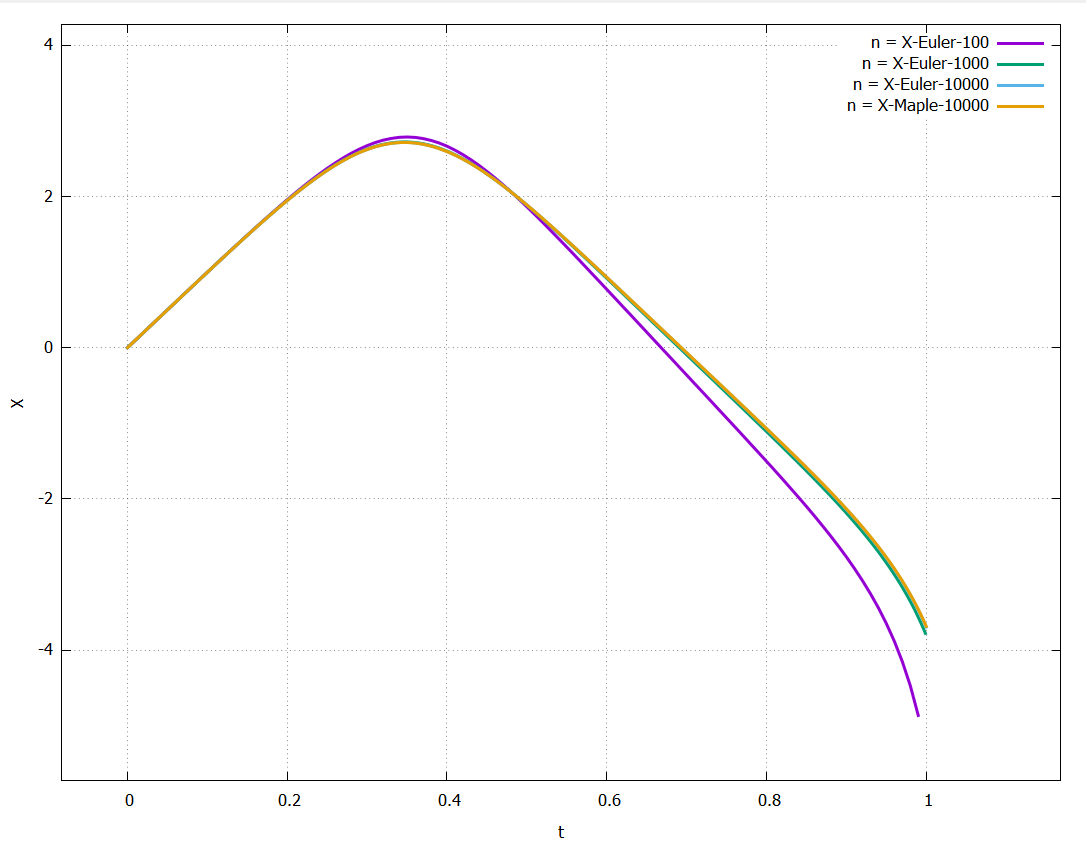


Рисунок 1 – графики (x(t)), построенные с помощью метода Эйлера с разными шагами и Maple.

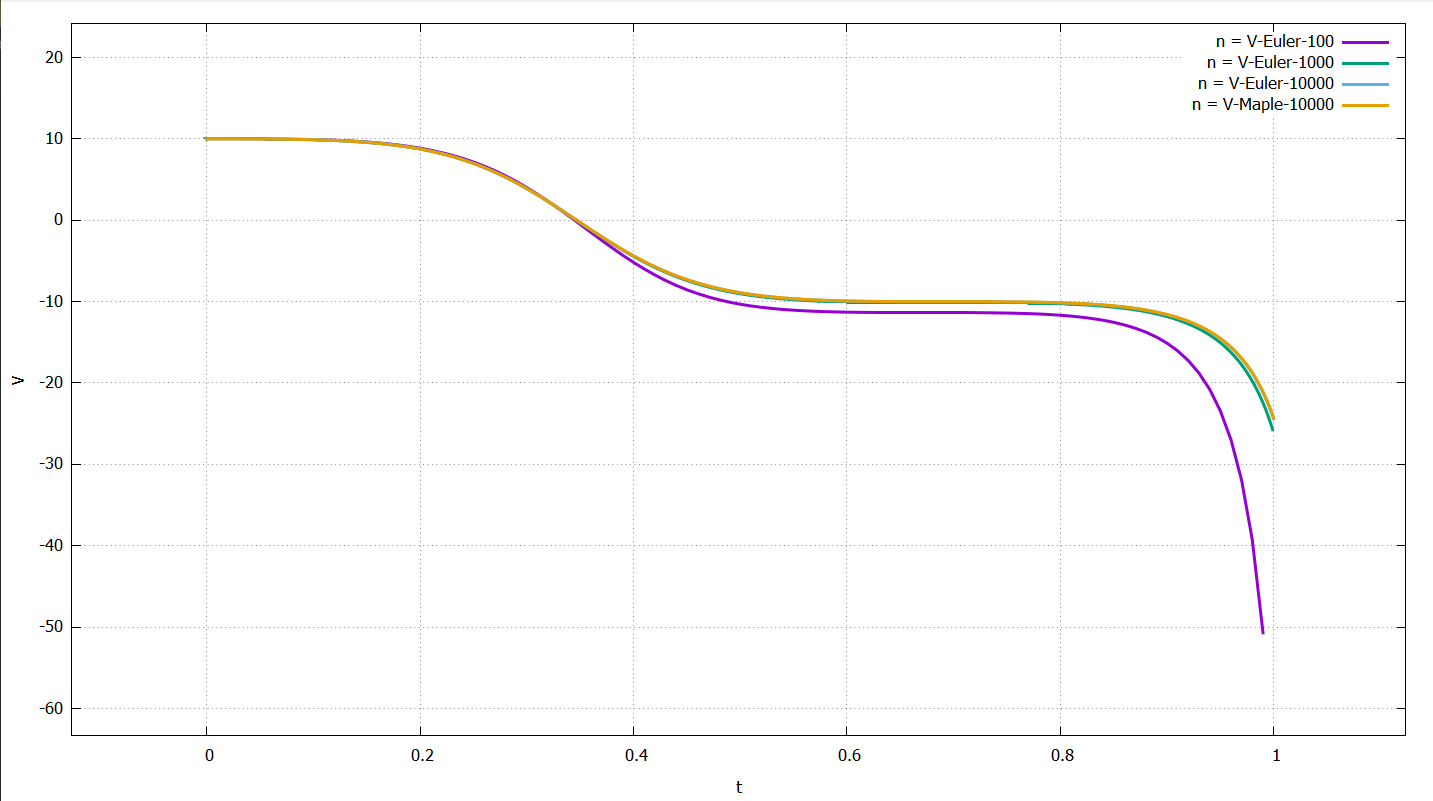
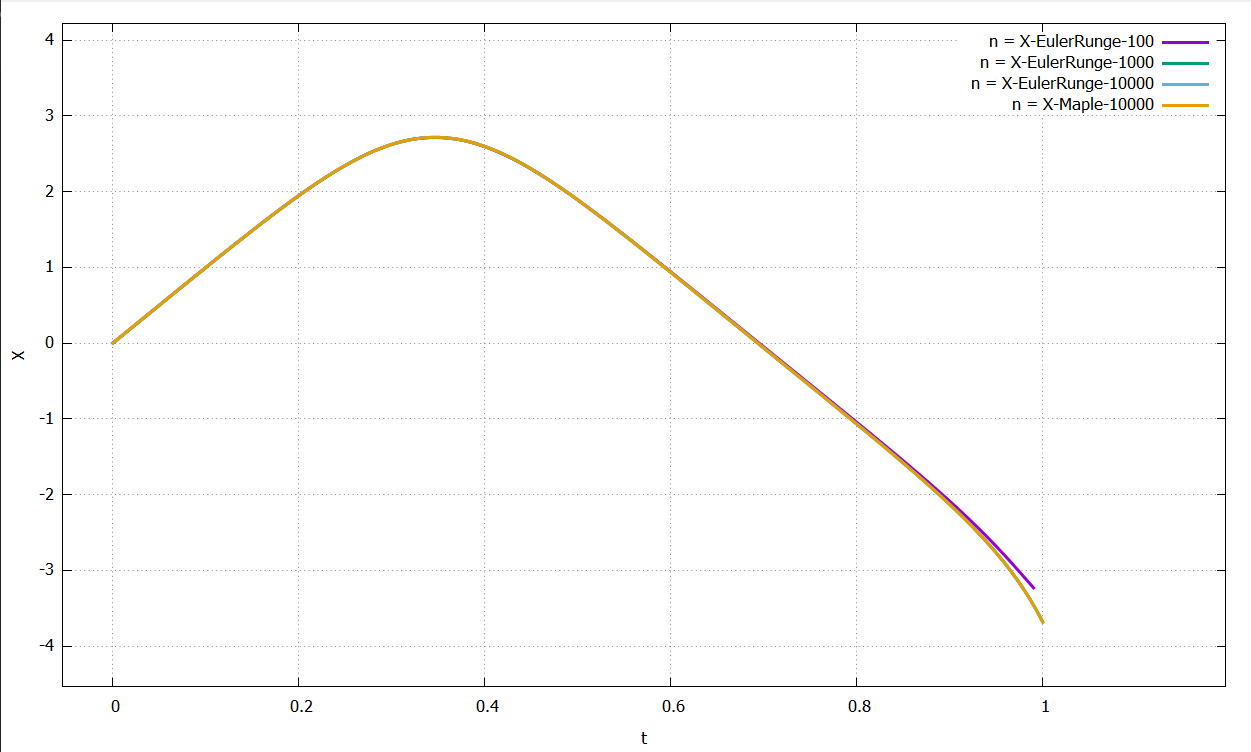


Рисунок 2– графики (v(t)), построенные с помощью метода Эйлера с разными шагами и Maple.

По графикам (Рисунок 1, Рисунок 2) видно, что с увеличением размерности график решения, построенный методом Эйлера, приближается к «точному» решению при N = 10000.

Рисунок 3 - графики (x(t)), построенные с помощью метода Эйлера, с повышенной точностью, с разными шагами и Maple.

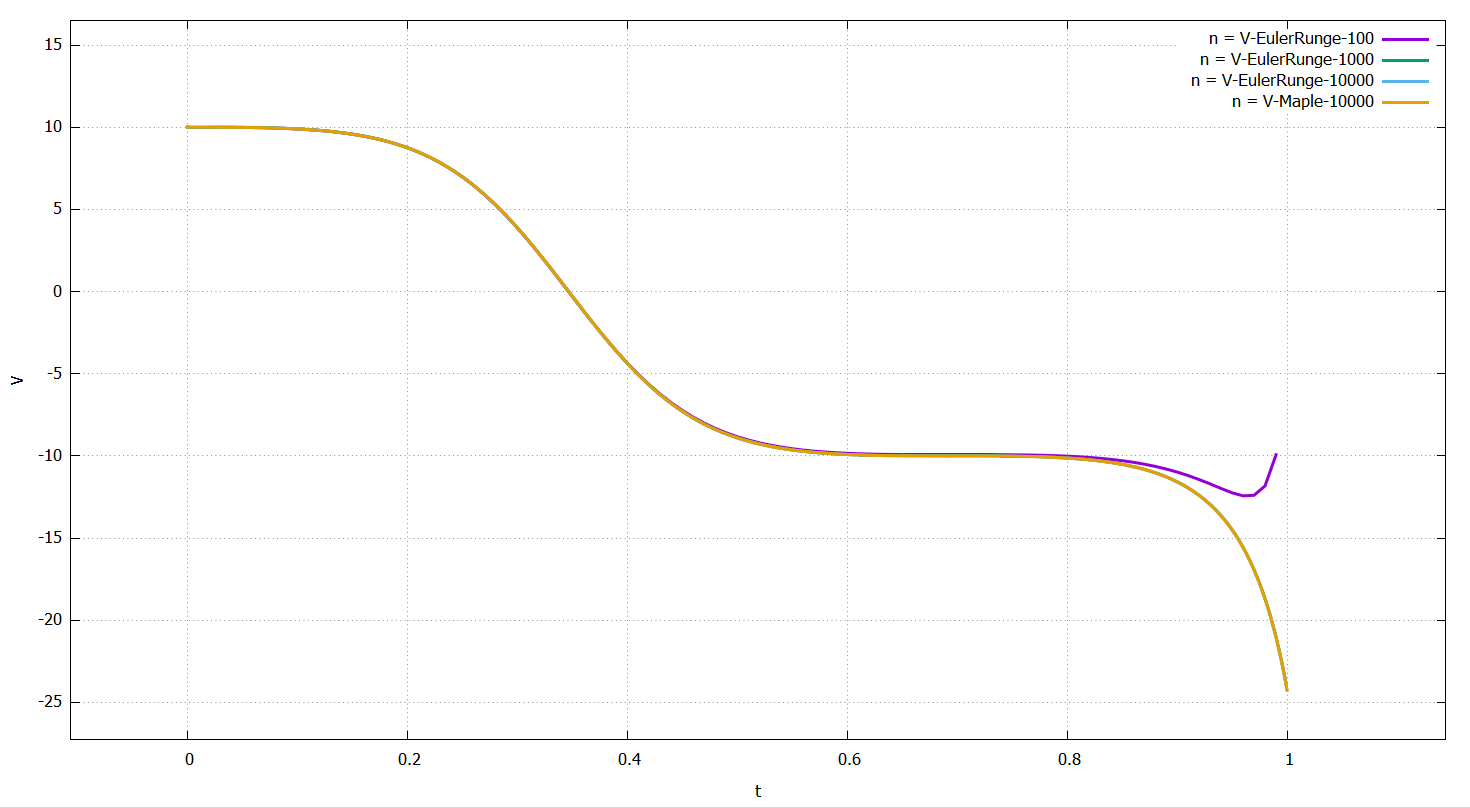


Рисунок 4 - графики (v(t)), построенные с помощью метода Эйлера, с повышенной точностью, с разными шагами и Maple.

По графикам (Рисунок 5, Рисунок 4) видно, что с увеличением размерности, график решения, построенного методом Эйлера, с повышенной точность правилом Рунге, приближается к «точному» решению, уже при N = 1000.

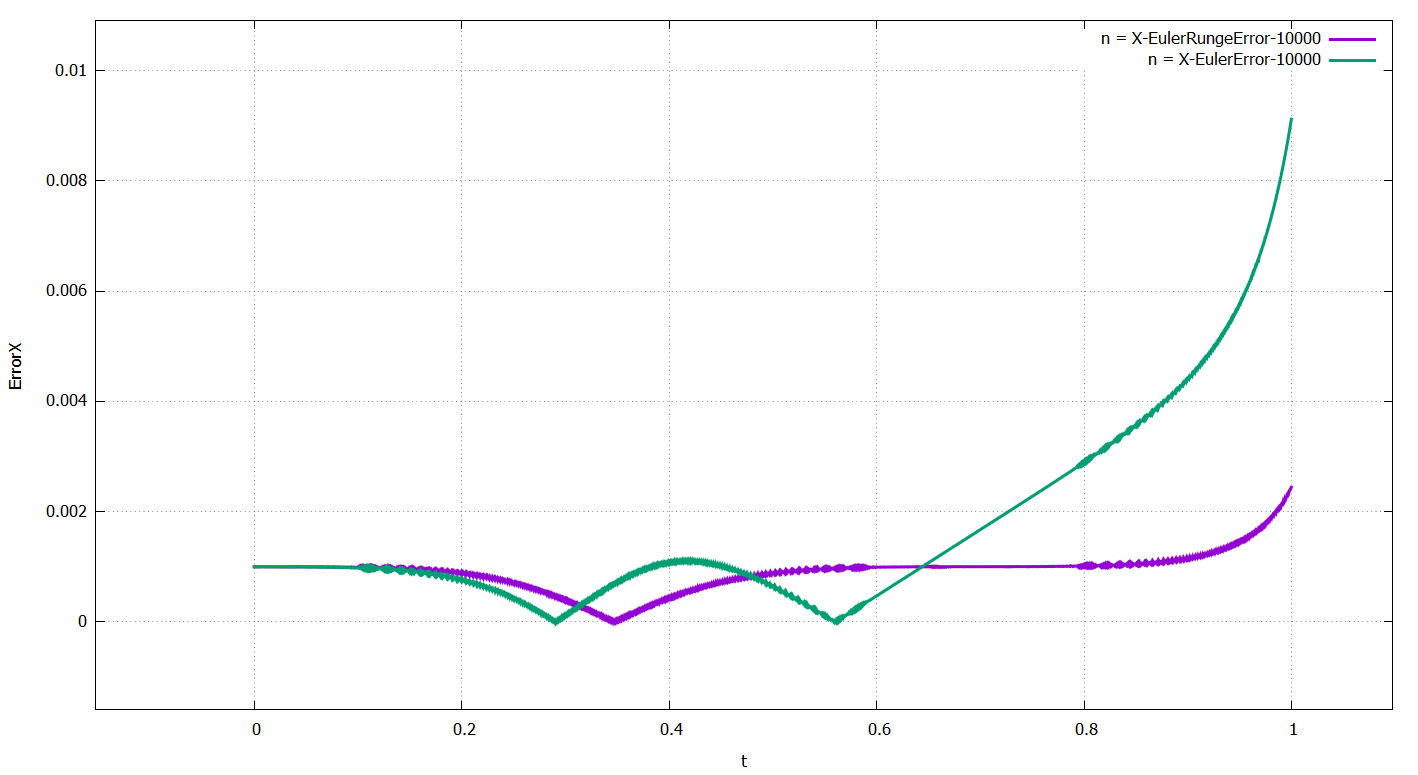


Рисунок 6– графики ошибок функции x(t), построенные методом Эйлера, и методом, повышающим точность Эйлера.

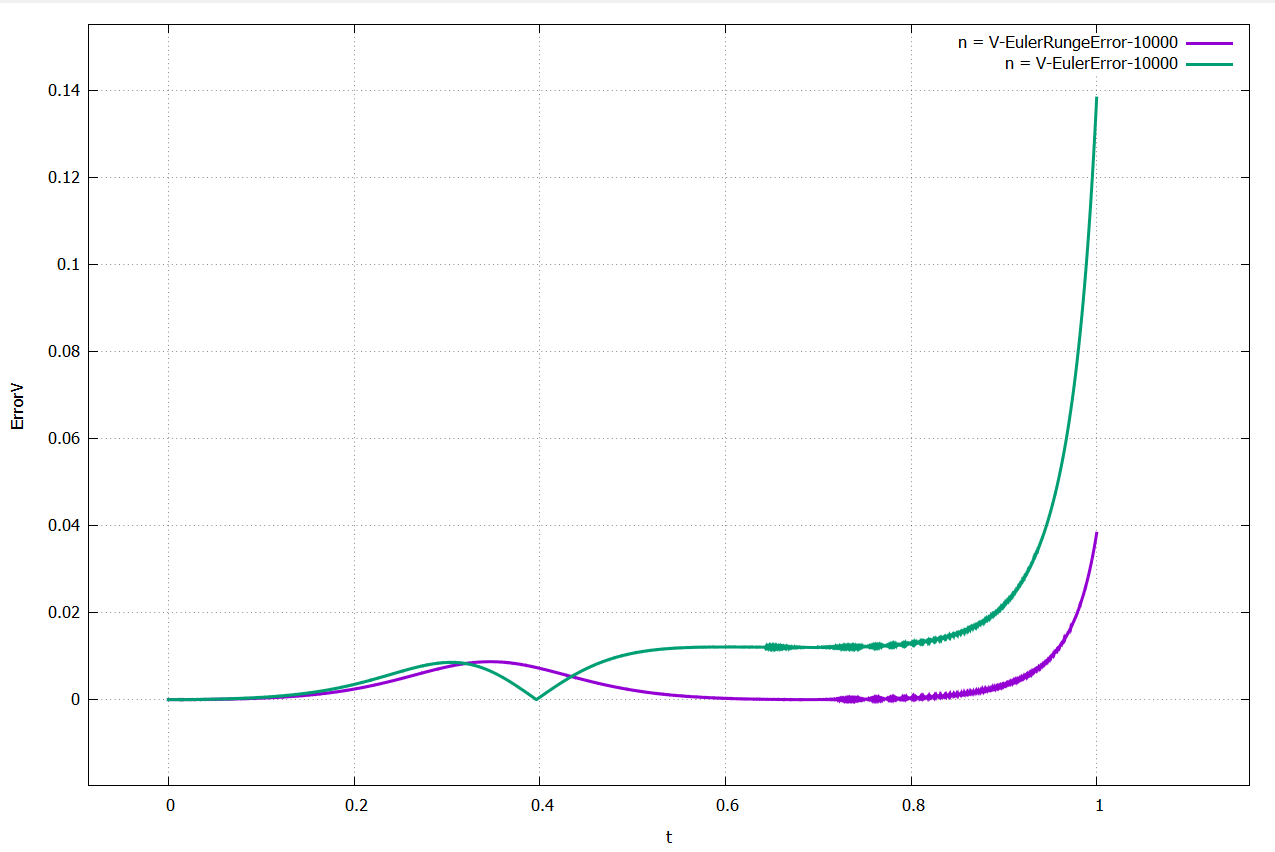


Рисунок 7 – графики ошибок функции v(t), построенные методом Эйлера, и методом, повышающим точность Эйлера.

Как видно из (Рисунок 6, Рисунок 7), на одной и той же сетке, с помощью правила Рунге получилось повысить точность решения.

***Задача 2 (2 балла).***

1. Написать вычислительную программу на языке программирования C++ решения задачи (1) по явной двухшаговой схеме Адамса с постоянным шагом.
2. Исследовать зависимость решения при больших временах от величины шага временной сетки. Построить графики решений для различных значений шага.
3. Выполнить сравнение полученных решений с решением по методу Эйлера (задача 1) и численным решением в каком-либо математическом пакете, полученным с помощью метода высокого порядка точности (например, Рунге-Кутта 4–5). Построить графики разности решений.

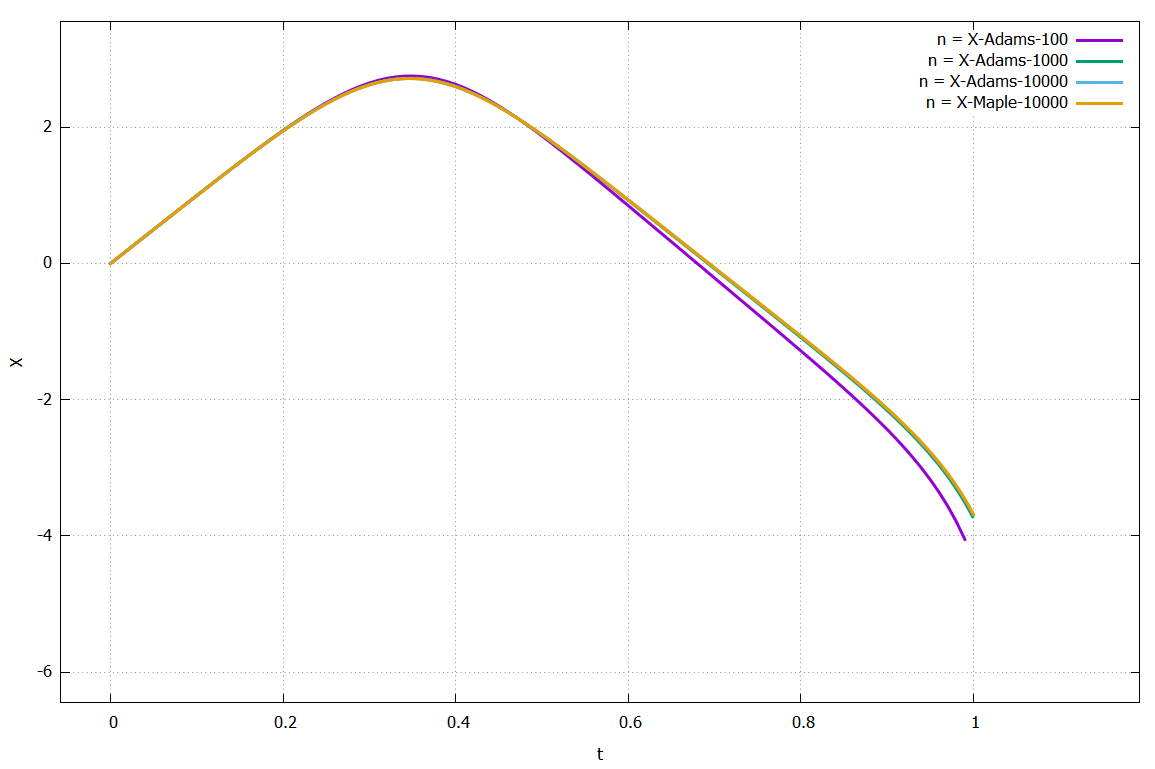


Рисунок 8 - графики (x(t)), построенные с помощью метода Адамса с разными шагами и Maple.

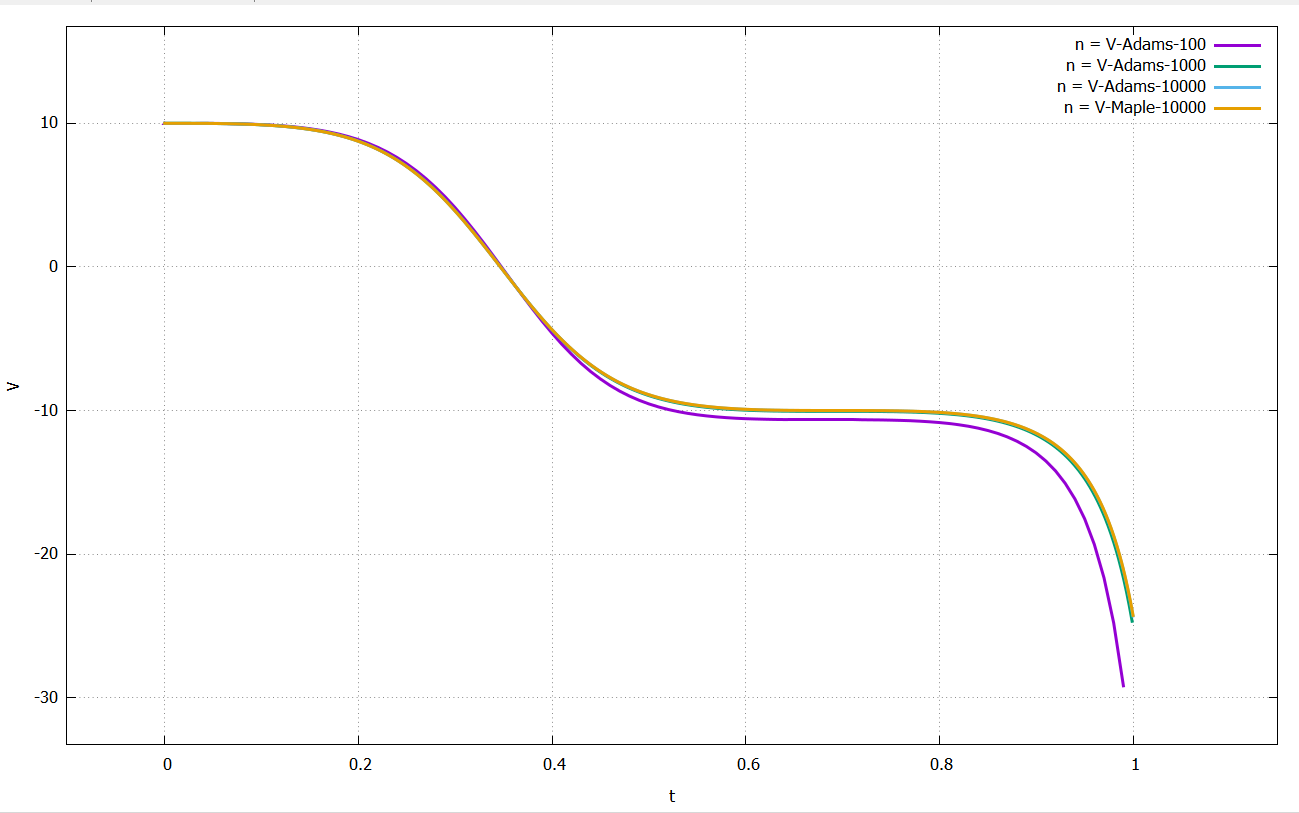


Рисунок 9 - графики (v(t)), построенные с помощью метода Адамса с разными шагами и Maple.

По графикам (Рисунок 8, Рисунок 9) видно, что с увеличением размерности, график решения, построенный методом Адамса приближается к «точному» решению уже при N = 1000. Что гораздо лучше, чем в методе Эйлера, но что сопоставимо с методом Эйлера, для которого повысили точность с помощью правила Рунге.

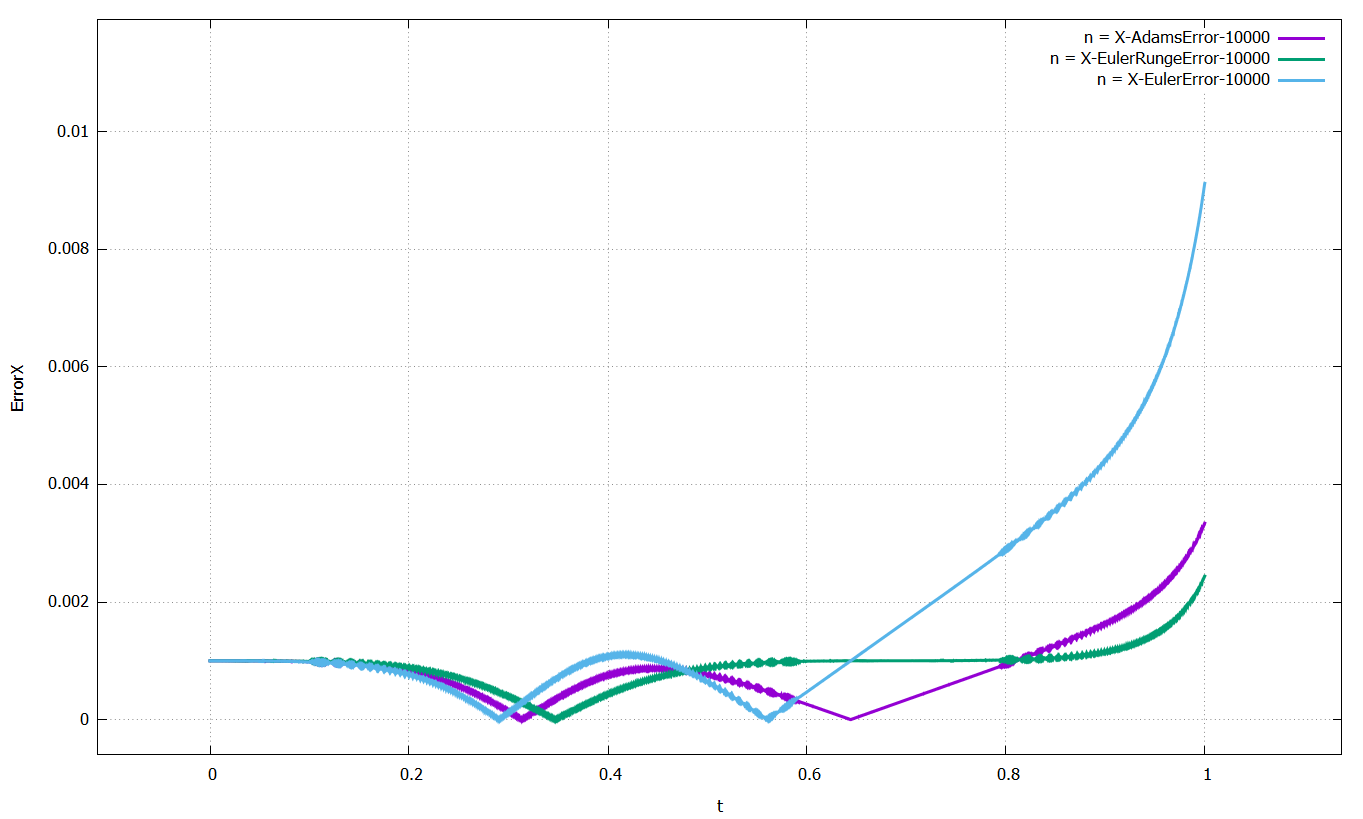


Рисунок 10 - графики ошибок функции x(t), построенные методами Адамса, Эйлера и Эйлера с применением правила Рунге.

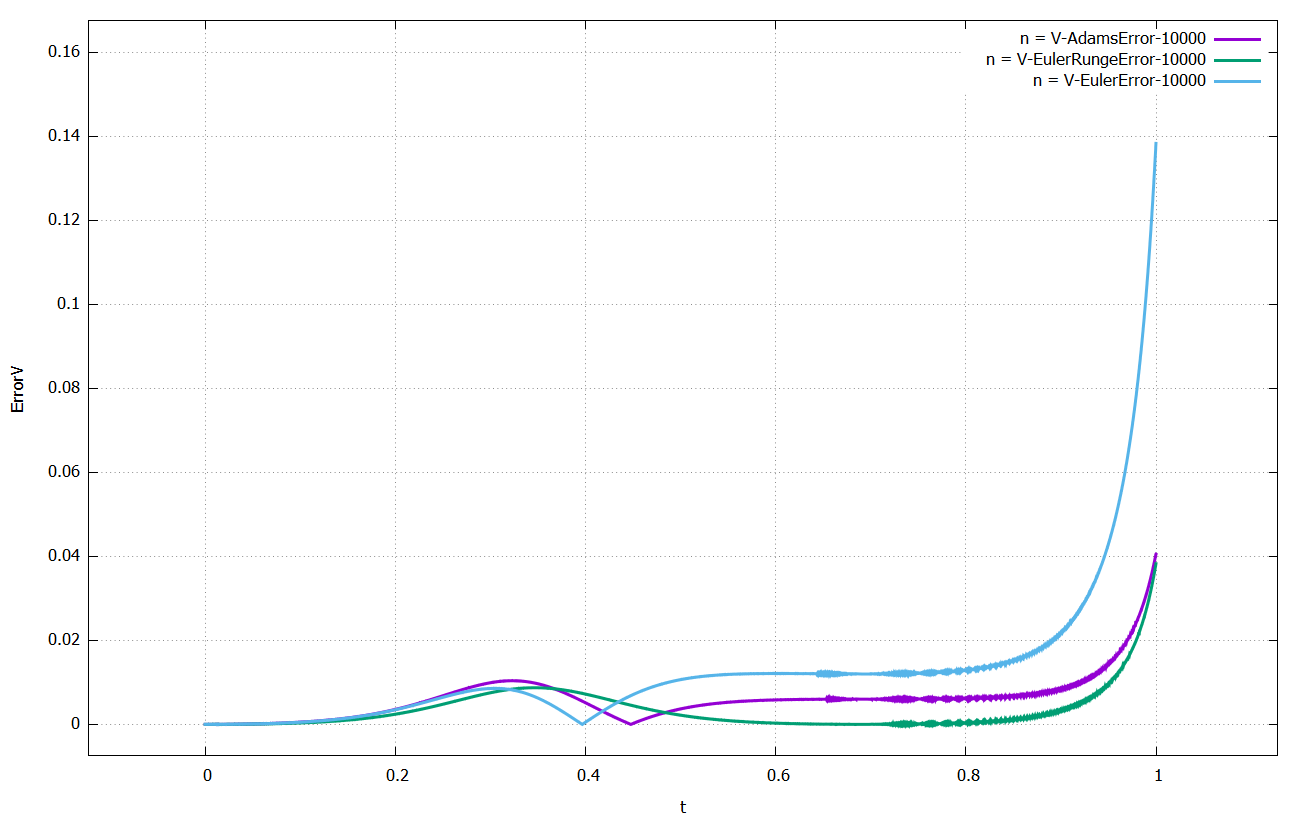


Рисунок 11 - графики ошибок функции v(t), построенные методами Адамса, Эйлера и Эйлера с применением правила Рунге.

Как видно из (Рисунок 10, Рисунок 11) точность метода Эйлера меньше, чем точность метода Адамса. Это происходит из-за того, что метод Адамса имеет второй порядок точности, когда метод Эйлера только первый. Метод Адамса почти не имеет отличий в точности с методом Эйлера, точность которого повысили с помощью правила Рунге.

***Задача 3 (2 балла).***

1. Написать вычислительную программу на языке программирования C++ решения задачи (1) методом Рунге-Кутта 4-го порядка.
2. Исследовать зависимость решения при больших временах от величины шага временной сетки. Построить графики решений для различных значений шага.
3. Выполнить сравнение полученных решений с численным решением в каком-либо математическом пакете, полученным с помощью метода высокого порядка точности (например, Рунге-Кутта 4–5). Построить графики разности решений.

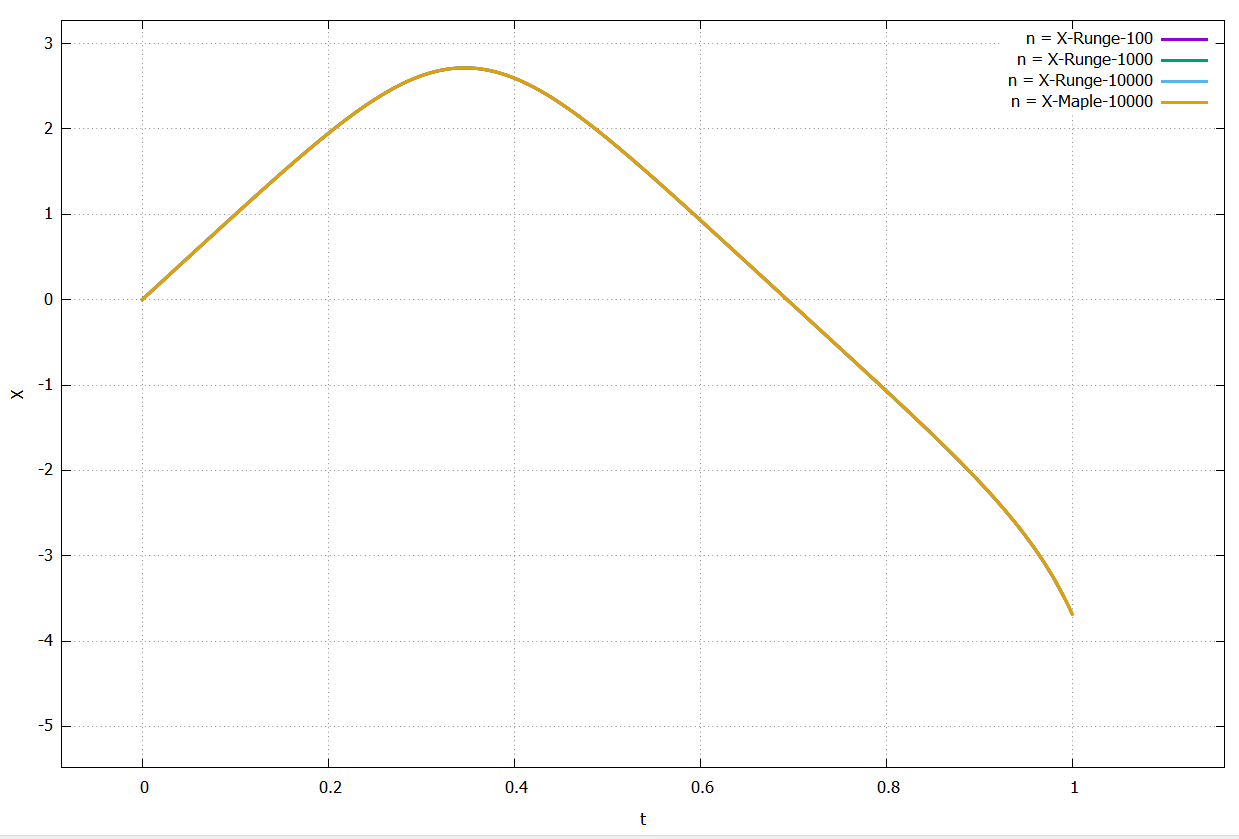


Рисунок 12 - графики (x(t)), построенные с помощью метода Рунге-Кутта 4-го порядка с разными шагами и Maple.

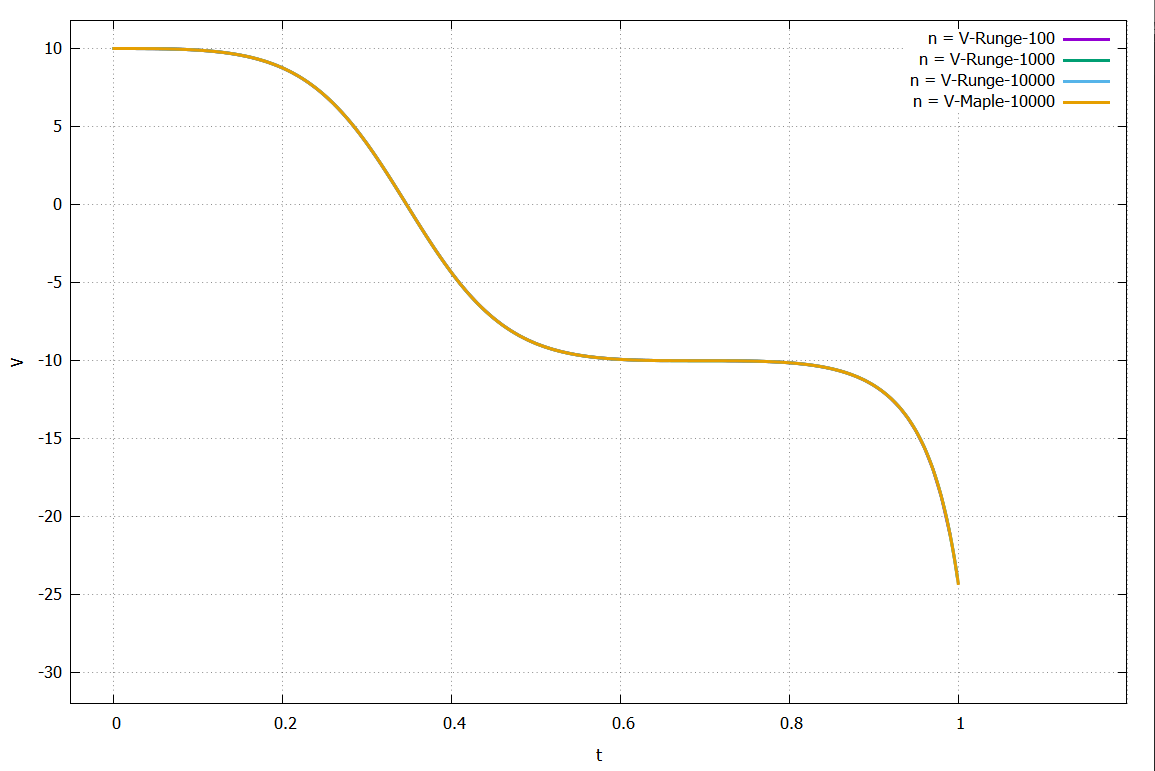


Рисунок 13- графики (v(t)), построенные с помощью метода Рунге-Кутта 4-го порядка с разными шагами и Maple.

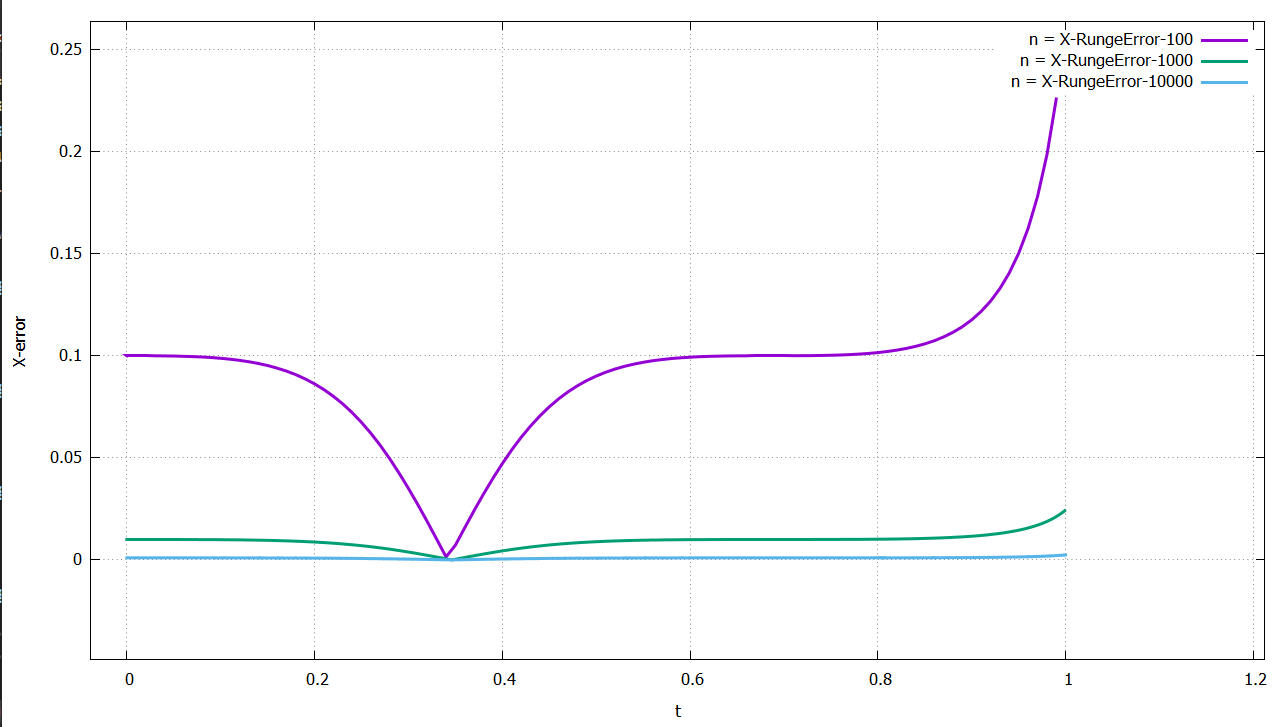


Рисунок 14 - графики ошибок функции x(t), построенные методами Рунге-Кутта 4-го порядка.

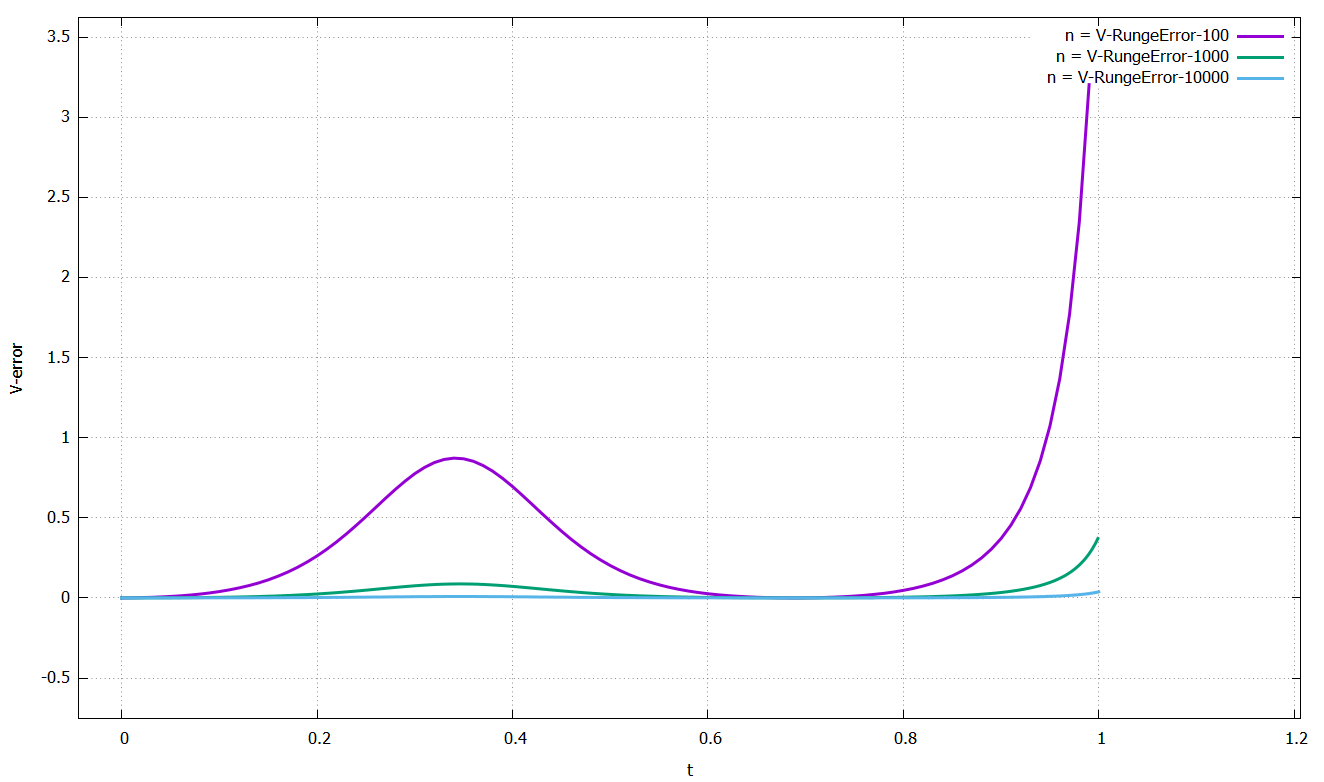


Рисунок 15- графики ошибок функции v(t), построенные методами Рунге-Кутта 4-го порядка.

По графикам видно, что начиная уже с N=1000 разница в решении методом Рунге-Кутта 4-5 порядка незначительна.

**II. Краевая задача для обыкновенного дифференциального уравнения второго прядка**

Решается следующая краевая задача для неоднородного ОДУ второго порядка:

Имеем параметры **(9 вариант)**:

***Задача 4 (3 балла).***

1. Написать вычислительную программу на языке программирования C++ решения задачи (2) конечно-разностным методом с решением получающейся СЛАУ методом прогонки.
2. Исследовать зависимость решения от величины шага сетки. Построить графики решений для различных значений шага.
3. Выполнить сравнение полученных решений с численным решением в каком-либо математическом пакете.
4. Построить графики разности решений.

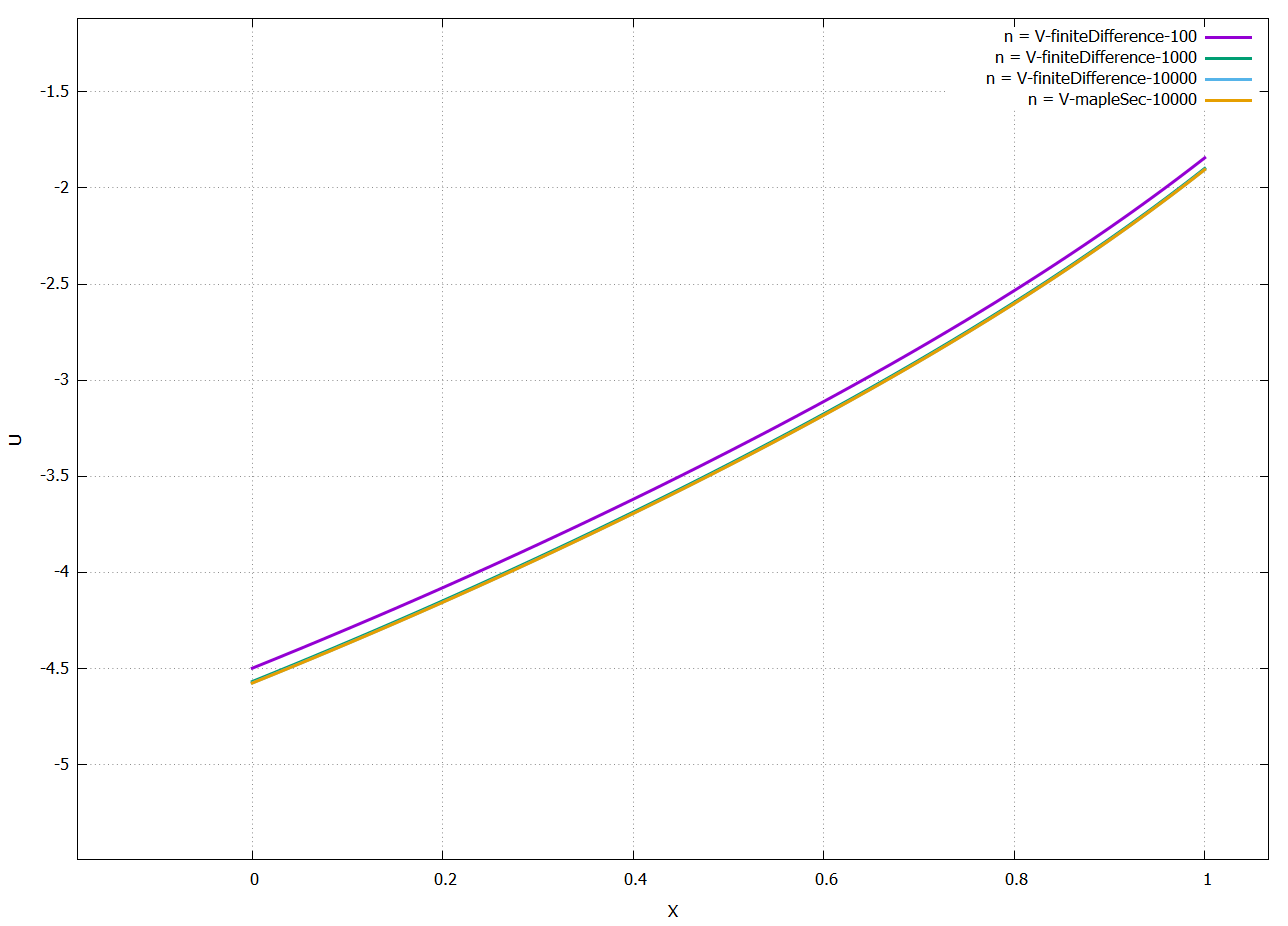


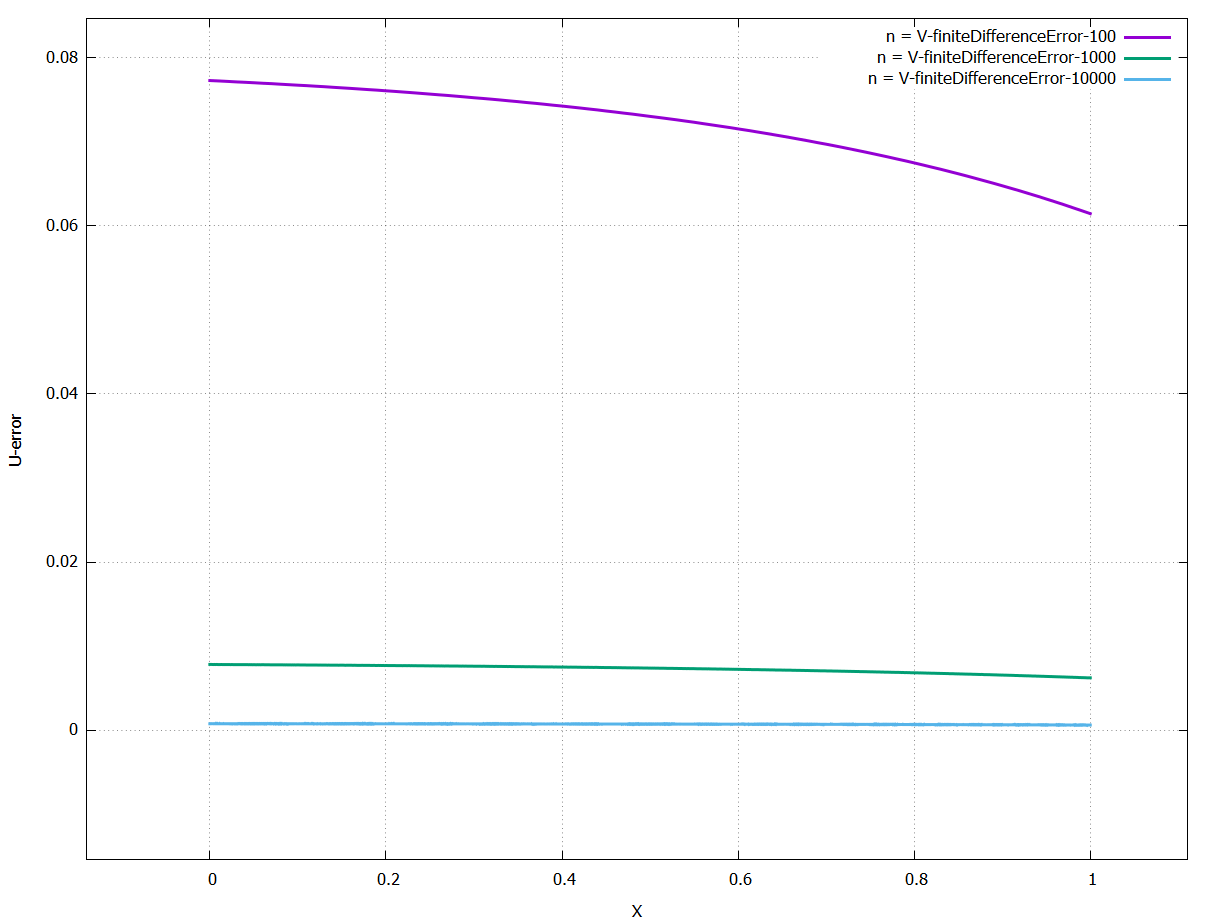
Рисунок 16- графики (u(x)), построенные с помощью метода конечных разностей с разными шагами и Maple.

Рисунок 17 - Ошибки метода конечных разностей на 100, 1000 и 10000 шагах.

По графикам (Рисунок 16, Рисунок 17) видно, что с увеличением размерности, графики решений, построенные методом конечных разностей, приближается к «точному» решению.

***Задача 5 (2 балла).***

1. Написать вычислительную программу на языке программирования C++ решения задачи (2) методом стрельбы (пристрелки). Решение соответствующей задачи Коши выполнить методом Рунге-Кутта 4-го порядка (использовать результаты задачи 3).
2. Выполнить сравнение полученного решения с решением, полученным в задаче 4.

**Вывод**

В результате проделанной лабораторной работы был изучен теоретический материал необходимый для решения начальных и краевых задач для обыкновенных дифференциальных уравнений.

Для каждой поставленной задачи написана вычислительная программа на языке программирования С++, выполняющая необходимые построения и расчеты.

Для системы ОДУ было выявлено, что метод Эйлера уступает в точности методу Адамса и методу Рунге-Кутта 4-го порядка. Метод Адамса является более точным, нежели метод Эйлера, но его точность фактически совпадает с методом Эйлера, к которому применили правило Рунге. Самым точным оказался метод Рунге-Кутта-4. Такой же метод по умолчанию используется в математическом пакете Maple для решения систем ДУ.

Для ОДУ второго порядка были построены решения конечно-разностным методом. С помощью метода прогонки была решена система уравнений и получено решение при различных Было проведено сравнение и исследование зависимости для численного приближенного решения к решению, выдаваемое математическим пакетом Maple.

**Приложение**

**lab1.cpp**

#include "EulerMethod.h"

#include "AdamsMethod.h"

#include "RungeKuttaMethod.h"

double newF2(double);

int main() {

system("chcp 1251");

cout.precision(10);

//cout << "Тест русского языка" << endl;

//double newX0 = 78, newV0 = 0, A = 1e-5, B = 1e+10;

//double newX0 = 78, newV0 = 0, A = 1, B = 1;

cout << "Start partial calculations" << endl;

//1.1

EulerMethod Euler;

cout << "Euler = " << Euler.ValuesOfX(1, 1e+4) << endl;

//double value = 1. / 2;

//cout << Euler.ValuesOfX(value, 1e+4) - 1.88807609975728 << endl;

////1.4

////do not work

////have tested for x = 0.5

//cout << (Euler.ValuesOfX(value, 1e+4) + (Euler.ValuesOfX(value, 1e+3) - Euler.ValuesOfX(value, 1e+4))) - 1.88807609975728 << endl;

cout << "End partial calculations" << endl;

int numberOfValues = 1e+3;

double valueXwhenTequalTo1 = -3.68617024223302;

vector<int> stepOfPointTimeGrid(numberOfValues);

vector<double> valuesFunction(numberOfValues);

vector<double> accuracySolution(numberOfValues);

//1.2

for (int i = 1; i < numberOfValues; i++){

stepOfPointTimeGrid[i] = i;

valuesFunction[i] = Euler.ValuesOfX(1, stepOfPointTimeGrid[i]);

}

Euler.makeFileForGraph(stepOfPointTimeGrid, valuesFunction, "1.2.txt");

//Euler.drawGraph("1.2.txt", "Euler values");

//1.3

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

accuracySolution[i] = valuesFunction[i] - valueXwhenTequalTo1;

Euler.makeFileForGraph(stepOfPointTimeGrid, accuracySolution, "1.3.txt");

//Euler.drawGraph("1.3.txt", "Accuracy Euler solution");

//2.1

AdamsMethod Adams;

cout << "Adams = " << Adams.ValuesOfX(1, 1e+4) << endl;

//2.2

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

valuesFunction[i] = Euler.ValuesOfX(1, stepOfPointTimeGrid[i]);

Adams.makeFileForGraph(stepOfPointTimeGrid, valuesFunction, "2.2.txt");

//Adams.drawGraph("2.2.txt", "Adams values");

//2.3

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

accuracySolution[i] = valuesFunction[i] - valueXwhenTequalTo1;

Adams.makeFileForGraph(stepOfPointTimeGrid, accuracySolution, "2.3.txt");

//Adams.drawGraph("2.3.txt", "Accuracy Adams solution");

//3.1

RungeKuttaMethod RungeKutta;

cout << "RungeKutta = " << RungeKutta.ValuesOfX(1, 1e+4) << endl;

//3.2

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

valuesFunction[i] = RungeKutta.ValuesOfX(1, stepOfPointTimeGrid[i]);

RungeKutta.makeFileForGraph(stepOfPointTimeGrid, valuesFunction, "3.2.txt");

//RungeKutta.drawGraph("3.2.txt", "RungeKutta values");

//3.3

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

accuracySolution[i] = valuesFunction[i] - valueXwhenTequalTo1;

RungeKutta.makeFileForGraph(stepOfPointTimeGrid, accuracySolution, "3.3.txt");

//RungeKutta.drawGraph("3.3.txt", "Accuracy RungeKutta solution");

}

double newF2(double x) {

return (1 / (x \* x) + 60 / (x \* x \* x \* x)) / 1.9;

}

**AdamsMethod.h**

#pragma once

#include "Calculation.h"

class AdamsMethod :

public Calculation {

public:

AdamsMethod(double x0, double v0, double A, double B, double (\*f)(double), int numberOfPoint) : Calculation(x0, v0, A, B, \*f, numberOfPoint) {

}

AdamsMethod(double x0, double v0, double A, double B, double (\*f)(double)) : Calculation(x0, v0, A, B, \*f) {

}

AdamsMethod() : Calculation() {

}

double ValuesOfX(double t) {

double newT = A \* t;

Calculate(newT);

return B \* x[this->numberOfPoint - 1];

}

double ValuesOfX(double t, int numberOfPoint) {

this->numberOfPoint = numberOfPoint;

this->x.resize(numberOfPoint);

this->v.resize(numberOfPoint);

double newT = A \* t;

Calculate(newT);

return B \* x[this->numberOfPoint - 1];

}

~AdamsMethod() {

}

private:

void Calculate(double t) {

double step = t / (numberOfPoint - 1);

//cout << "step = " << step << endl;

x[0] = x0;

v[0] = v0;

x[1] = x[0] + v[0] \* step;

v[1] = v[0] + f(x[0]) \* step;

for (int i = 2; i < numberOfPoint; i++) {

x[i] = x[i - 1] + step / 2 \* (3 \* v[i - 1] - v[i - 2]);

v[i] = v[i - 1] + step / 2 \* (3 \* f(x[i - 1]) - f(x[i - 2]));

}

}

};

**EulerMethod.h**

#pragma once

#include "Calculation.h"

class EulerMethod :

public Calculation {

public:

EulerMethod(double x0, double v0, double A, double B, double (\*f)(double), int numberOfPoint) : Calculation(x0, v0, A, B, \*f, numberOfPoint) {

}

EulerMethod(double x0, double v0, double A, double B, double (\*f)(double)) : Calculation(x0, v0, A, B, \*f) {

}

EulerMethod() : Calculation() {

}

double ValuesOfX(double t) {

double newT = A \* t;

Calculate(newT);

return B \* x[this->numberOfPoint - 1];

}

double ValuesOfX(double t, int numberOfPoint) {

this->numberOfPoint = numberOfPoint;

this->x.resize(numberOfPoint);

this->v.resize(numberOfPoint);

double newT = A \* t;

Calculate(newT);

return B \* x[this->numberOfPoint - 1];

}

~EulerMethod() {

}

private:

void Calculate(double t) {

double step = t / (numberOfPoint - 1);

//cout << "step = " << step << endl;

x[0] = x0;

v[0] = v0;

for (int i = 1; i < numberOfPoint; i++) {

x[i] = x[i - 1] + v[i - 1] \* step;

v[i] = v[i - 1] + f(x[i - 1]) \* step;

}

}

};

**RungeKutMethod.h**

#pragma once

#include "Calculation.h"

class RungeKuttaMethod :

public Calculation {

public:

RungeKuttaMethod(double x0, double v0, double A, double B, double (\*f2)(double), int numberOfPoint) : Calculation(x0, v0, A, B, \*f2, numberOfPoint) {

}

RungeKuttaMethod(double x0, double v0, double A, double B, double (\*f2)(double)) : Calculation(x0, v0, A, B, \*f2) {

}

RungeKuttaMethod() : Calculation() {

}

double ValuesOfX(double t) {

double newT = A \* t;

Calculate(newT);

return B \* x[this->numberOfPoint - 1];

}

double ValuesOfX(double t, int numberOfPoint) {

this->numberOfPoint = numberOfPoint;

this->x.resize(numberOfPoint);

this->v.resize(numberOfPoint);

double newT = A \* t;

Calculate(newT);

return B \* x[this->numberOfPoint - 1];

}

~RungeKuttaMethod() {

}

private:

void Calculate(double t) {

double step = t / (numberOfPoint - 1);

//cout << "step = " << step << endl;

v[0] = v0;

x[0] = x0;

for (int i = 1; i < numberOfPoint; i++) {

double kv0 = f(x[i - 1]);

double kx0 = v[i - 1];

double kv1 = f(x[i - 1] + step / 2 \* kx0);

double kx1 = v[i - 1] + step / 2 \* kv0;

double kv2 = f(x[i - 1] + step / 2 \* kx1);

double kx2 = v[i - 1] + step / 2 \* kv1;

double kv3 = f(x[i - 1] + step \* kx2);

double kx3 = v[i - 1] + step \* kv2;

v[i] = v[i - 1] + step / 6 \* (kv0 + 2 \* kv1 + 2 \* kv2 + kv3);

x[i] = x[i - 1] + step / 6 \* (kx0 + 2 \* kx1 + 2 \* kx2 + kx3);

}

}

};