Отчёт по лабораторной работе

Решение задачи Коши для обыкновенных дифференциальных уравнений и систем

**Выполнила**:

студентка гр. 382003\_4

Суслова Виктория Павловна

1. **Постановка задачи**

Решить тестовую задачу, и основные задачи, используя метод Рунге-Кутта 2, 3 и 4-го порядка без контроля локальной погрешности и с контролем. Для тестовой задачи построить графики точного и приближенного решений. Для основной задачи №1 – график приближенного решения, для основной задачи №2 – графики приближенного решения, а также фазовую траекторию.

Тестовая задача: , y(0) = 1;

Основная задача №1: , y(1) = 1, b = 2;

Основная задача №2: , y(0) = 1, z(0) = 1, b = 1;

Написать программу, реализующую алгоритм решения задачи Коши для обыкновенных дифференциальных уравнений и систем методом Рунге-Кутта 2-го, 3-го и 4-го порядков. К программе предъявляются следующие требования:

* реализация в виде консольного приложения на одном из языков С, C++, C# или Fortran;
* ввод данных из текстового файла;
* форматированный вывод результатов расчетов в текстовый фал. При этом допускается вывод в окно консоли справочной информации и информации о ходе выполнения программы;
* в выходных данных программы должны быть указаны:
* число шагов n ;
* , где b – правая граница;
* , где LEE – оценка локальной погрешности;
* общее число удвоений шага и общее число деления шага;
* , при ;
* , при ;
* , при для тестовой задачи;
* возможность изменения области интегрирования и задания других начальных условий;
* возможность проведения расчетов с постоянным шагом;
* возможность проведения расчетов с контролем локальной погрешности;
* контроль максимально допустимого числа итераций, а также контроль выхода на правую границу;
* наличие текстового файла с описанием конкретной решаемой задачи и структуры входного и выходного файлов.

1. **Краткие сведения по численным метода решения ОДУ**

Метод Рунге-Кутта 2 порядка:

Метод Рунге-Кутта 3 порядка:

Метод Рунге-Кутта 4 порядка:

Схема с половинным шагом и корректировкой значения:

Сначала вычисляется точка из точки с шагом , затем вычисляется точка из точки с шагом .

Контрольное слагаемое:

Оценка локальной погрешности:

,

где p – порядок метода

Метод управления шагом ( для этого вводим малое :

1. если , то принимаем точку , счет продолжаем с тем же шагом ;
2. если , то принимаем точку , но для следующей точки берем шаг ;
3. если , вычисленная точка не подходит, возвращаемся к точке и вычисляем новую шагом .

Скорректированное значение:

Глобальная погрешность (для тестовой задачи):

,

где U – точное решение тестовой задачи, V – численное решение тестовой задачи, вычисленное методом Рунге – Кутта порядка p.

1. **Исследование порядка сходимости для тестовой задачи**

Метод сходится в точке , если в этой точке при .

Метод имеет порядок p, если .

Погрешность () и шаг () подсчитаны с помощью программы с условиями: , правая граница () = 1, максимальное количество шагов () = 1000, контроль локальной погрешности с .

Метод 2 порядка:

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | . |
|  |  |  |

Так как выполняются все условия метод сходится с порядком p = 2.

Метод 3 порядка:

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | . |
|  |  |  |

Так как выполняются все условия метод сходится с порядком p = 3.

Метод 4 порядка:

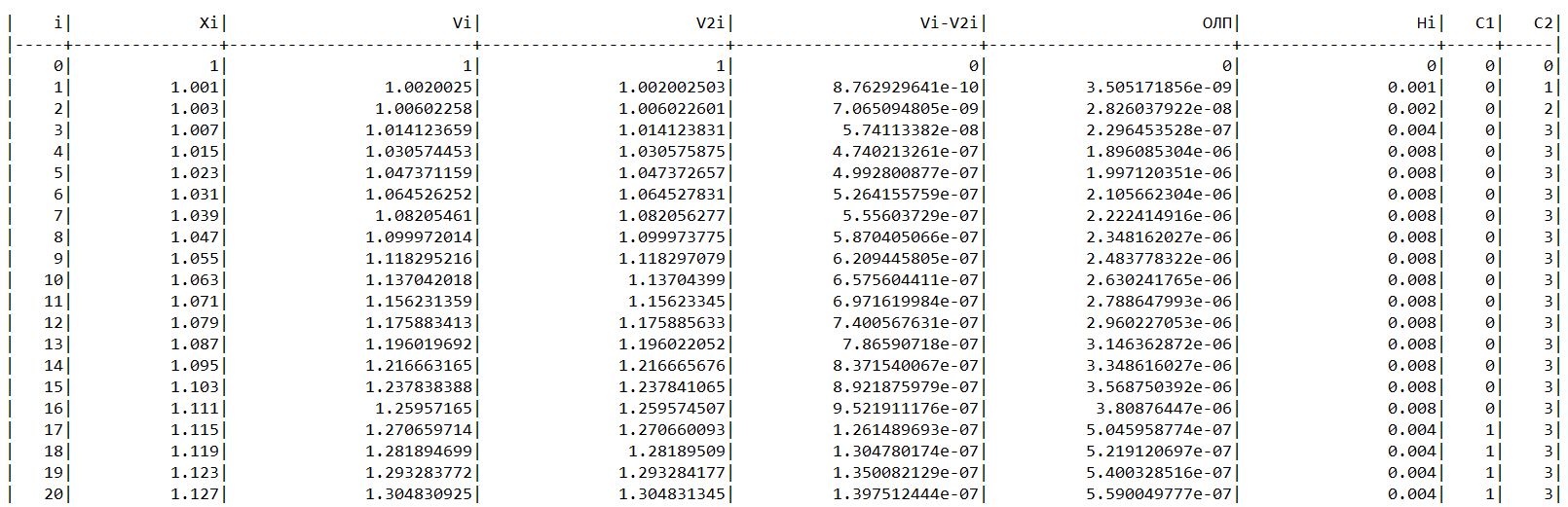
|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | . |
|  | 0 |  |

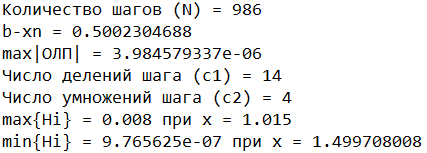
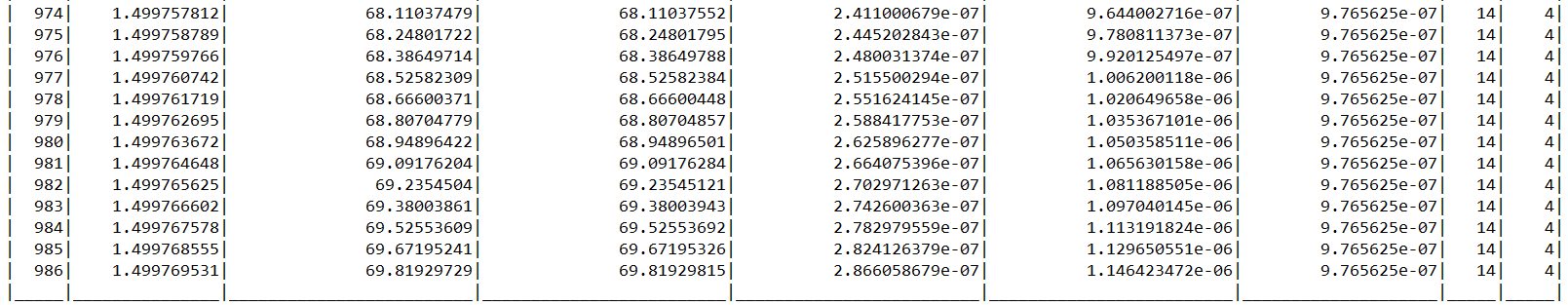
Так как выполняются все условия метод сходится с порядком p = 4.

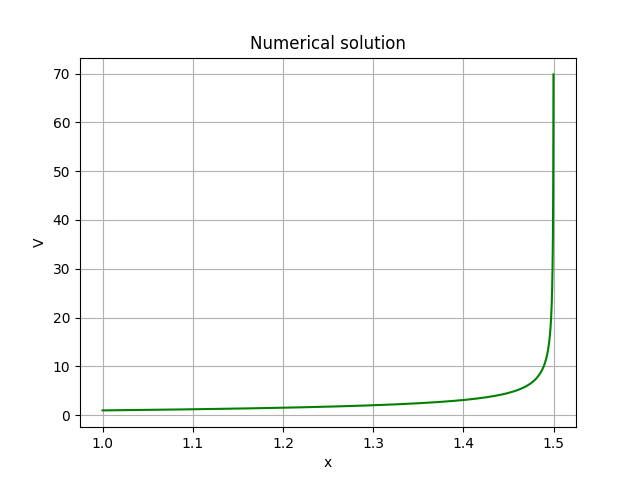
1. **Результаты численных экспериментов для основных задач**

Основная задача 1 решена со следующими начальными условиями: , шаг сетки () = 0.001, правая граница () = 2, максимальное количество шагов () = 1000, контроль локальной погрешности с .

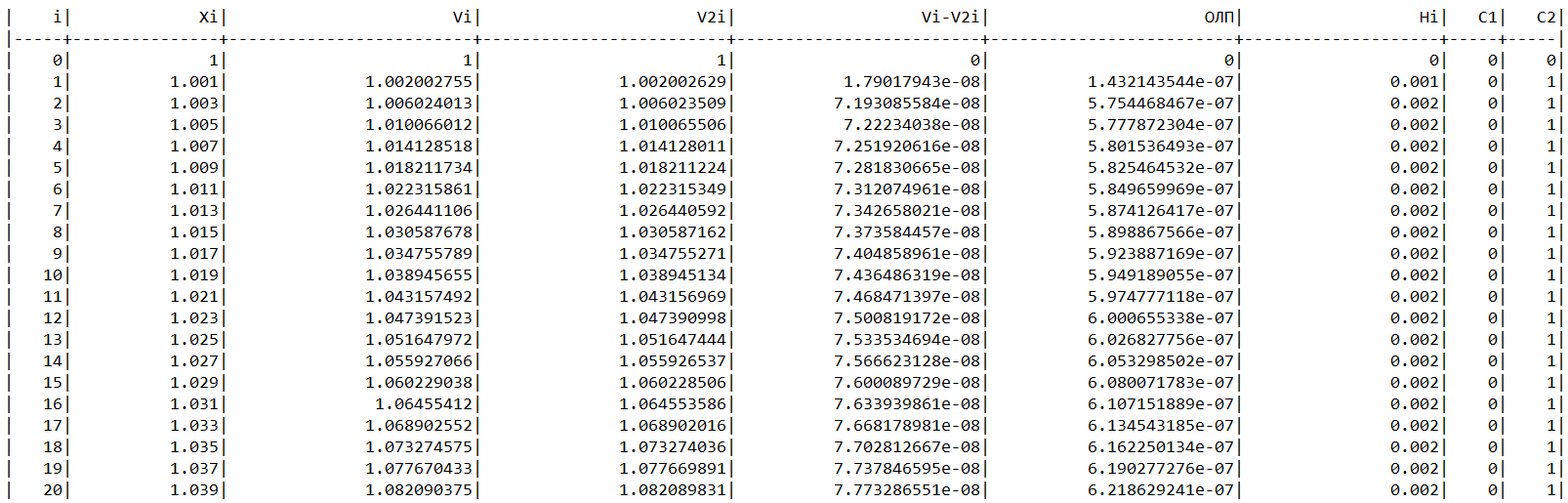
Метод 2 порядка:



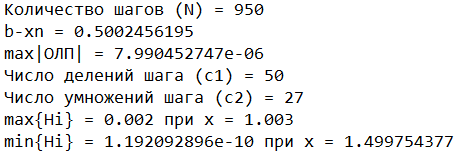
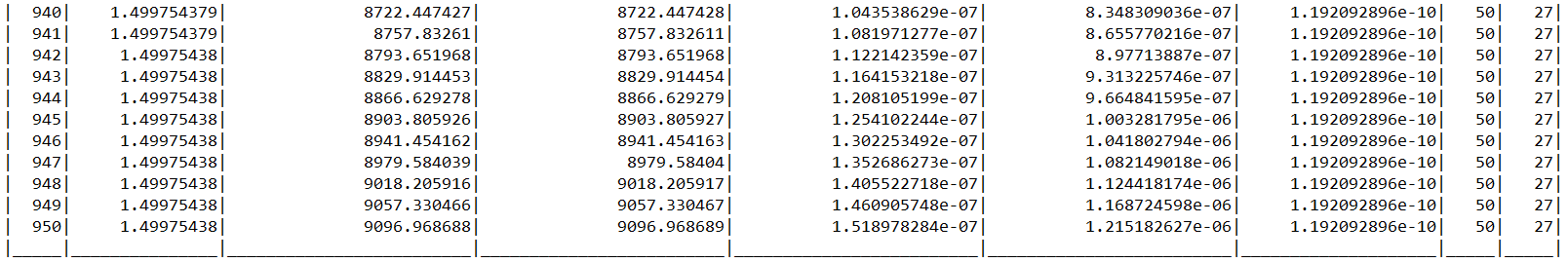
… 

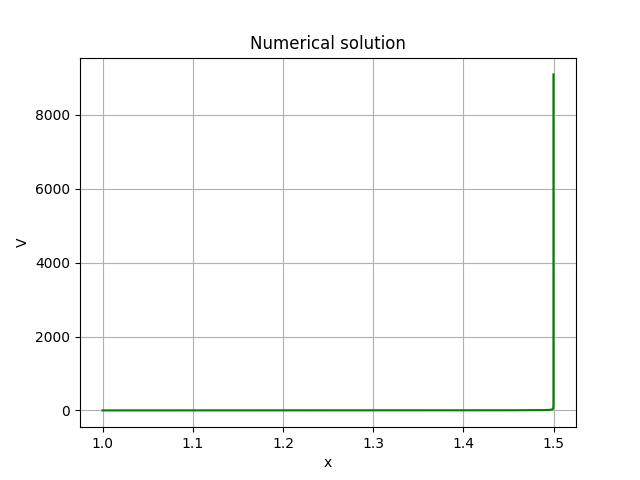
****

Метод 3 порядка:

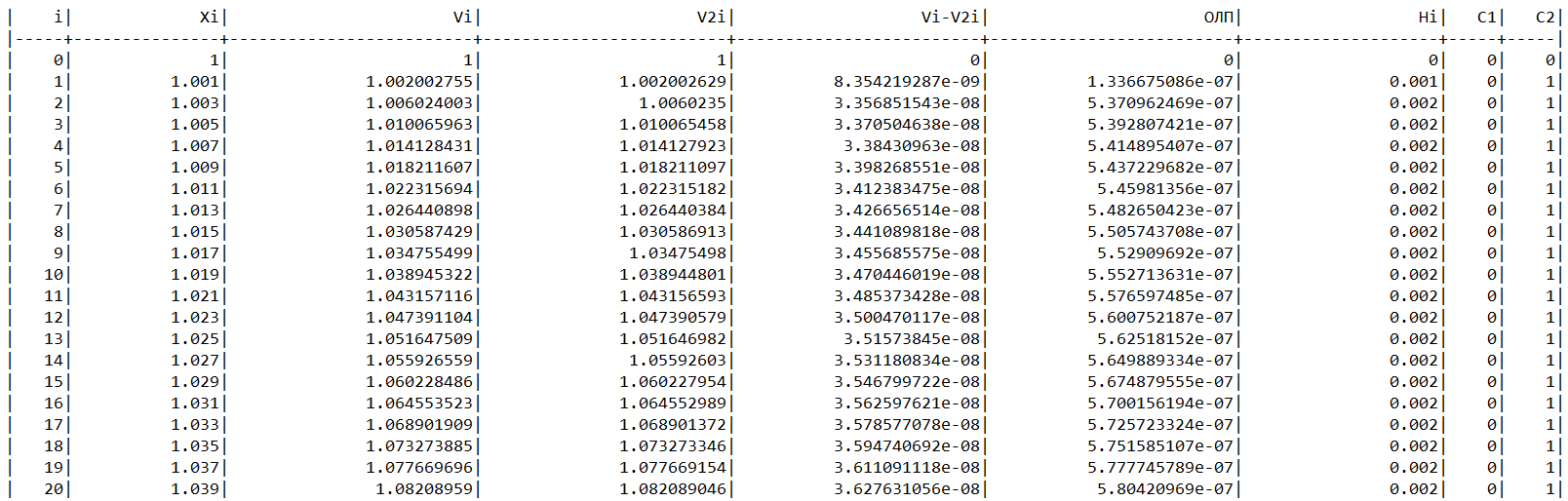


…

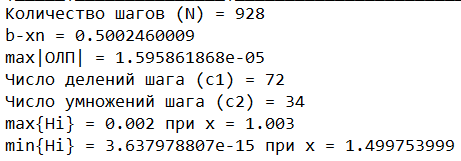
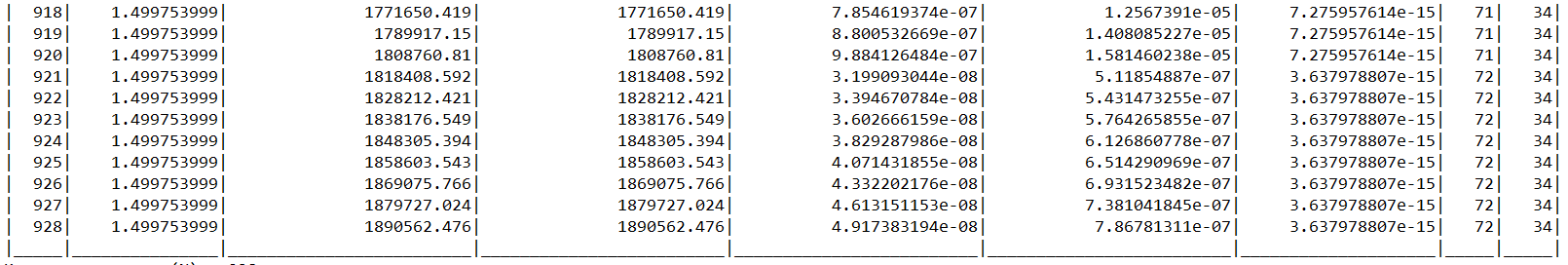


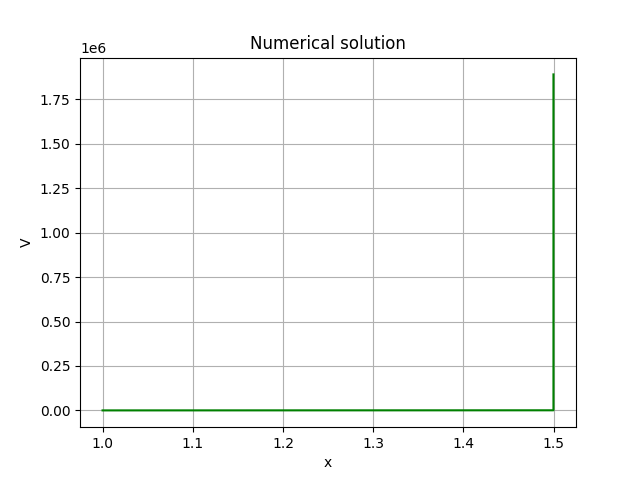
****

Метод 4 порядка:



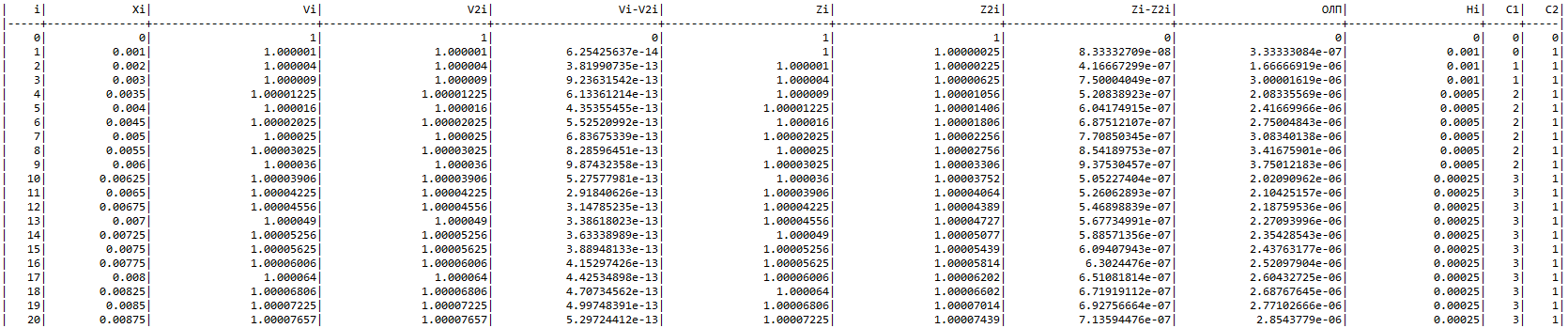
…



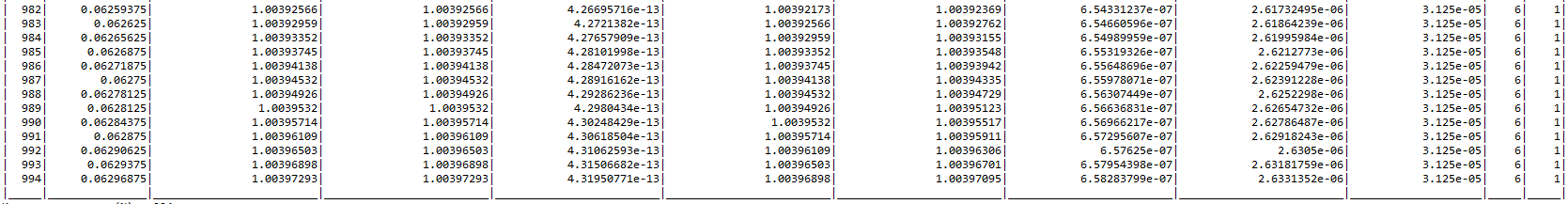


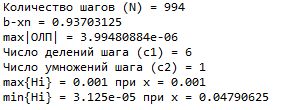
Основная задача 2 решена со следующими начальными условиями: , шаг сетки () = 0. 001, правая граница () = 1, максимальное количество шагов () = 1000, контроль локальной погрешности с

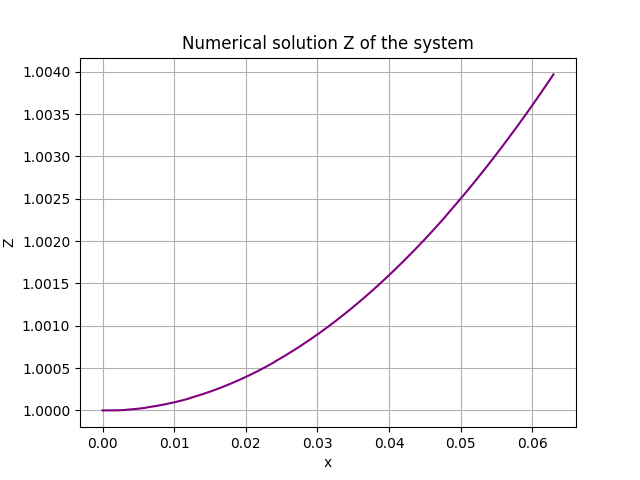
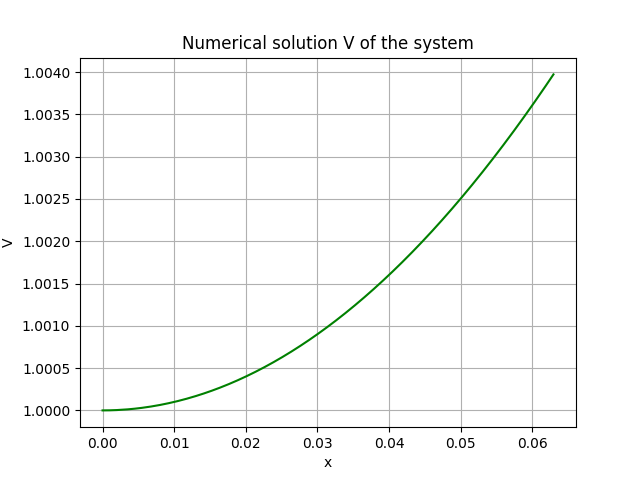
Метод 2 порядка:

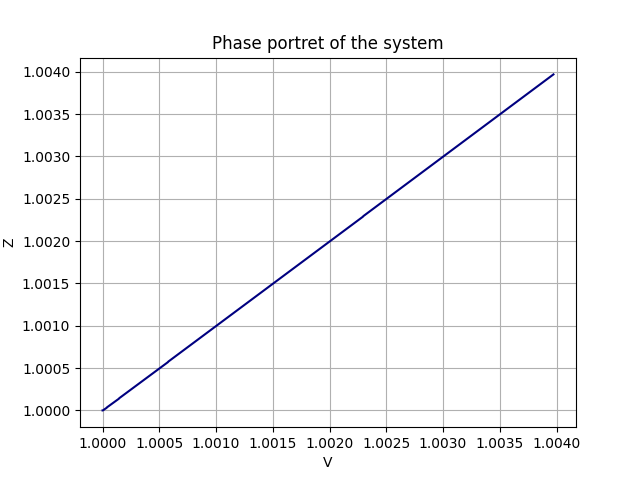
****

…

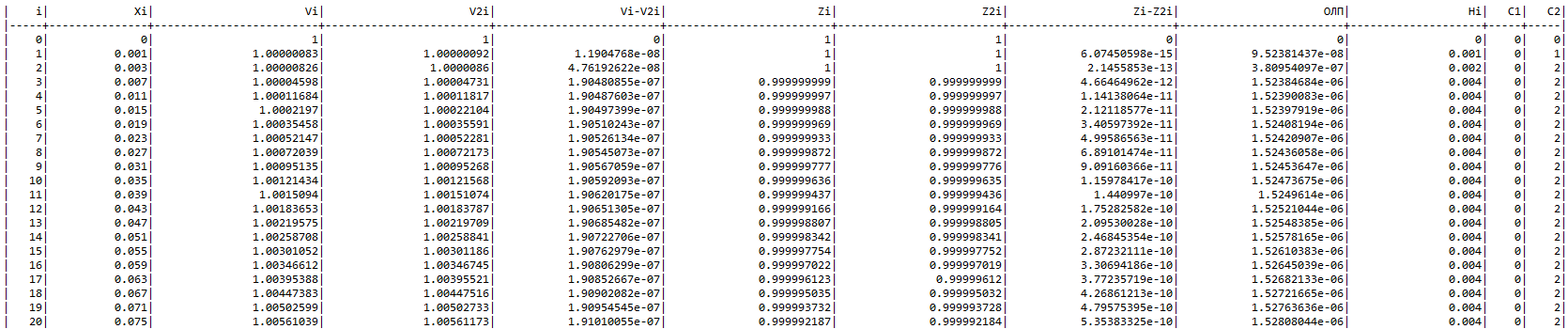
****

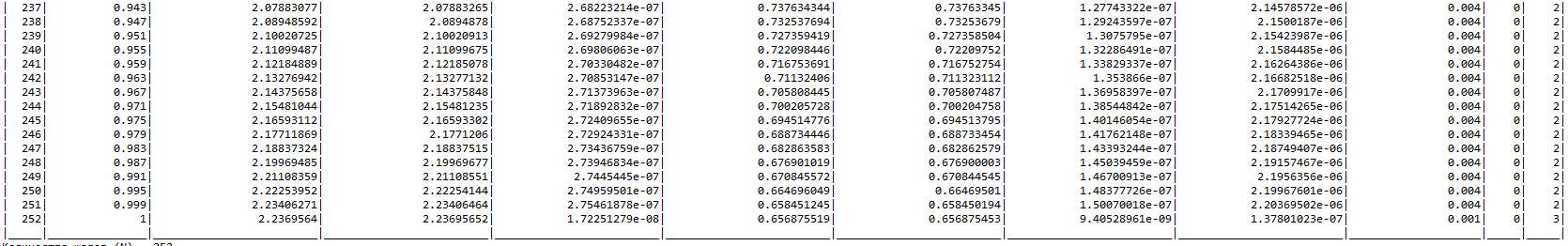
****

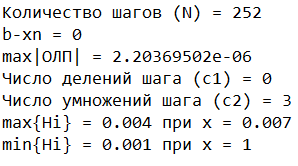
****

****

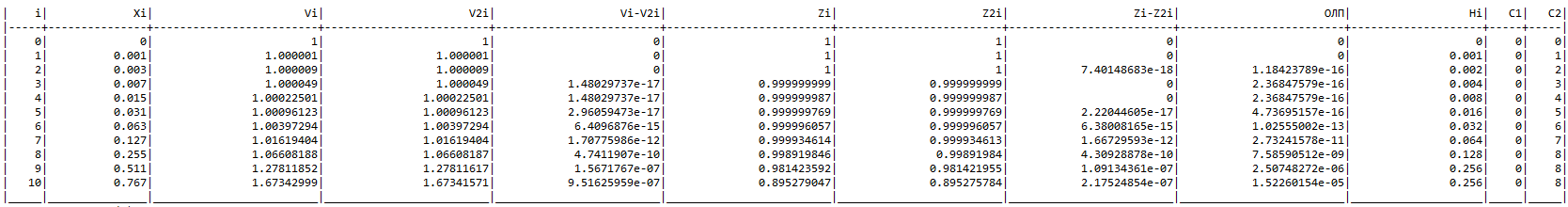
Метод 3 порядка:

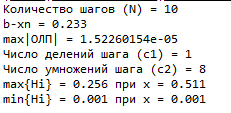
****

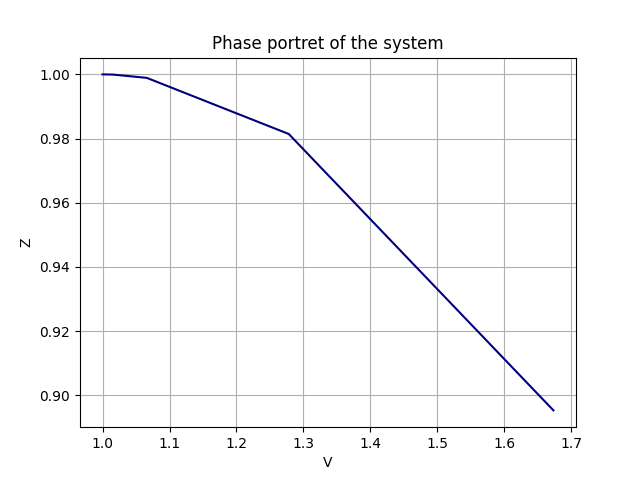
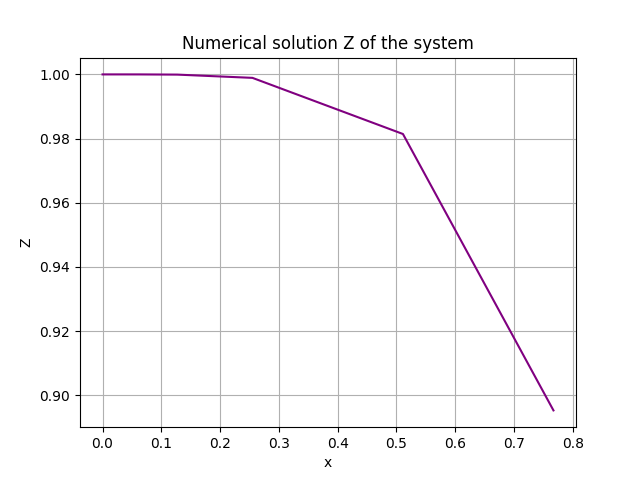
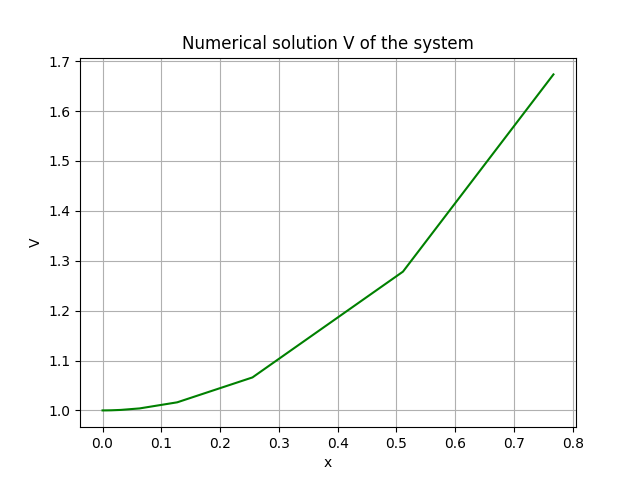
****

****

Метод 4 порядка:

****

****

****

1. **Наблюдения и выводы**

Тестовая, основная №1, основная №2 задачи решены методами Рунге-Кутты 2, 3 и 4 порядков с контролем локальной погрешности и без контроля, то есть с постоянным шагом. В итоге получены следующие результаты:

* В тестовой задаче на отрезке от 0 до 1 с КЛП с точностью 0.00000001 более точным оказался метод 4 порядка, количество шагов в каждом методе отличается: 2 порядок – 121 шагов, 3 – 21 шагов, 4 – 14 шагов. Без КЛП точнее также оказался метод 4 порядка. Решение методом 4 порядка без КЛП точнее, чем с КЛП.
* В основной задаче 1 с КЛП с точностью 0.0001 при одних и тех же начальных условиях метод 4 порядка показал более точные результаты.
* В основной задаче 2 с КЛП с точностью 0.0001 при одних и тех же начальных условиях метод 4 порядка оказался точнее. Метод 2 порядка за максимальное количество шагов не смог посчитать траекторию до правой границы. Количество шагов в методах 3 – 36 шагов, 4 – 10 шагов. Без контроля локальной погрешности 4 порядок точнее, но при этом метод 4 порядка без КЛП оказался точнее, чем метод 4 порядка с КЛП.

Таким образом, можно сделать вывод, что метод 4 порядка оказывается точнее, чем методы низших порядков. При этом методы с постоянным шагом дают более точный результат, чем методы с контролем локальной погрешности. Однако предпочтительнее использовать методы с контролем локальной погрешности.

1. **Текст программы**

Файл main.cpp

#include "MainWork\_1.h"

#include "TestWork.h"

#include "MainWork\_2.h"

#include <fstream>

#include <clocale>

void menu()

{

std::cout << ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

std::cout << ("Tasks:\n");

std::cout << ("1. Test task\n");

std::cout << ("2. Main task 1\n");

std::cout << ("3. Main task 2\n");

std::cout << ("0. exit\n");

std::cout << ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

void main()

{

setlocale(LC\_CTYPE, "rus");

std::ifstream test\_in;

test\_in.open("input.txt");

double xt, x1, x2, vt, v1, v2, z;

double bt, b1, b2, h, eps;

int p, n;

bool olp;

test\_in >> xt >> vt >> bt >> x1 >> v1 >> b1 >> x2 >> v2 >> z >> b2;

int k = 10;

while(k != 0)

{

menu();

std::cin >> k;

switch (k)

{

case 1:

{

//тестовая задача

std::ofstream test\_out;

test\_out.open("test\_output.txt");

if (test\_out.is\_open())

{

std::cout << "Method order - p (2, 3, 4): "; std::cin >> p;

std::cout << "Maximum number of steps - n: "; std::cin >> n;

std::cout << "Local error control (yes - 1, no - 0): "; std::cin >> olp;

if (olp == 1)

{

std::cout << "Enter eps: ";

std::cin >> eps;

}

else { eps = 0; }

std::cout << std::endl;

h = (bt - xt) / n;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl << std::endl;

std::cout << "Solusion of the problem by the " << p << "th order Runge-Kutta method with initial conditions: "

<< "\n- y(" << xt << ") = " << vt << "\n- grid step (h) = " << h

<< "\n- right border (b) = " << bt << "\n- maximum number of steps (n) = " << n << std::endl;

if (olp == 1)

{

std::cout << "- local error control with eps = " << eps << " ." << std::endl;

}

else std::cout << "- without local error control." << std::endl;

std::cout << std::endl;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

test\_out << "Решение тестовой задачи методом Рунге-Кутта " << p << " порядка с начальными условиями: "

<< "\n- y(" << xt << ") = " << vt << "\n- шаг сетки (h) = " << h

<< "\n- правая граница (b) = " << bt << "\n- максимальное количество шагов (n) = " << n << std::endl;

if (olp == 1)

{

test\_out << "- контроль локальной погрешности с eps = " << eps << " ." << std::endl;

}

else test\_out << "- без контроля локальной погрешности." << std::endl;

test\_out << std::endl;

TestWork d(vt, h, bt, p, n, olp, eps);

d.DoStep();

d.Out(test\_out);

d.Draw();

test\_out.close();

}

std::cout << std::endl << "The result is written to a file: \"test\_output.txt\"" << std::endl;

break;

}

case 2:

{

//основная задача 1

std::ofstream main1\_out;

main1\_out.open("main1\_output.txt");

if (main1\_out.is\_open())

{

std::cout << "Method order - p (2, 3, 4): "; std::cin >> p;

std::cout << "Maximum number of steps - n: "; std::cin >> n;

std::cout << "Local error control (yes - 1, no - 0): "; std::cin >> olp;

if (olp == 1)

{

std::cout << "Enter eps: ";

std::cin >> eps;

}

else { eps = 0; }

std::cout << std::endl;

h = (b1 - x1) / n;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl << std::endl;

std::cout << "Solusion of the problem by the " << p << "th order Runge-Kutta method with initial conditions: "

<< "\n- y(" << x1 << ") = " << v1 << "\n- grid step (h) = " << h

<< "\n- right border (b) = " << b1 << "\n- maximum number of steps (n) = " << n << std::endl;

if (olp == 1)

{

std::cout << "- local error control with eps = " << eps << " ." << std::endl;

}

else std::cout << "- without local error control." << std::endl;

std::cout << std::endl;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl << std::endl;

main1\_out << "Решение основной задачи 1 методом Рунге-Кутта " << p << " порядка с начальными условиями: "

<< "\n- y(" << x1 << ") = " << v1 << "\n- шаг сетки (h) = " << h

<< "\n- правая граница (b) = " << b1 << "\n- максимальное количество шагов (n) = " << n << std::endl;

if (olp == 1)

{

main1\_out << "- контроль локальной погрешности с eps = " << eps << " ." << std::endl;

}

else main1\_out << "- без контроля локальной погрешности." << std::endl;

main1\_out << std::endl;

MainWork\_1 a(v1, h, b1, p, n, olp, eps);

a.DoStep();

a.Out(main1\_out);

a.Draw();

main1\_out.close();

}

std::cout << std::endl << "The result is written to a file: \"main1\_output.txt\"" << std::endl;

break;

}

case 3:

{

//основная задача 2

std::ofstream main2\_out;

main2\_out.open("main2\_output.txt");

if (main2\_out.is\_open())

{

std::cout << "Method order - p (2, 3, 4): "; std::cin >> p;

std::cout << "Maximum number of steps - n: "; std::cin >> n;

std::cout << "Local error control (yes - 1, no - 0): "; std::cin >> olp;

if (olp == 1)

{

std::cout << "Enter eps: ";

std::cin >> eps;

}

else { eps = 0; }

std::cout << std::endl;

h = (b2 - x2) / n;

main2\_out << "Решение основной задачи 2 методом Рунге-Кутта " << p << " порядка с начальными условиями: \n"

<< "- y(" << x2 << ") = " << v2 << "\tz(" << x2 << ") = " << z << "\n- шаг сетки (h) = " << h

<< "\n- правая граница (b) = " << b2 << "\n- максимальное количество шагов (n) = " << n << std::endl;

if (olp == 1)

{

main2\_out << "- контроль локальной погрешности с eps = " << eps << " ." << std::endl;

}

else main2\_out << "- без контроля локальной погрешности." << std::endl;

main2\_out << std::endl;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl << std::endl;

std::cout << "Solusion of the problem by the " << p << "th order Runge-Kutta method with initial conditions: "

<< "\n- y(" << x2 << ") = " << v2 << "\tz(" << x2 << ") = " << z << "\n- grid step (h) = " << h

<< "\n- right border (b) = " << b2 << "\n- maximum number of steps (n) = " << n << std::endl;

if (olp == 1)

{

std::cout << "- local error control with eps = " << eps << " ." << std::endl;

}

else std::cout << "- without local error control." << std::endl;

std::cout << std::endl;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl << std::endl;

MainWork\_2 e(v2, z, h, b2, p, n, olp, eps);

e.DoStep();

e.Out(main2\_out);

e.Draw();

main2\_out.close();

}

std::cout << std::endl << "The result is written to a file: \"main2\_output.txt\"" << std::endl;

break;

}

default:

{

k = 0;

}

}

}

}

Файл TestWork.h

#pragma once

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

#include <fstream>

#include "matplotlibcpp.h"

namespace plt = matplotlibcpp;

class TestWork

{

public:

std::vector<double> Grid;

std::vector<double> Step;

std::vector<double> Num;

std::vector<double> DNum;

std::vector<double> FinalNum;

std::vector<double> TrueVal;

std::vector<double> Main;

std::vector<double> OLP;

std::vector<double> MultStep;

std::vector<double> DivStep;

double RightBorder;

double eps;

int MaxSteps;

bool CtrlLocalPogr;

int p;

int Mult;

int Div;

TestWork(double init\_y, double init\_h, double right\_border, int p, int steps, bool ctrl\_local\_pogr, double eps);

double F(double u);

double TrueF(double x);

double Method(int p, double v, double h);

void NumStep();

void DoStep();

void Out(std::ostream& os);

void Draw();

};

Файл TestWork.cpp

#include "TestWork.h"

TestWork::TestWork(double init\_y, double init\_h, double right\_border, int p, int max\_steps, bool ctrl\_local\_pogr, double eps)

{

this->RightBorder = right\_border;

this->p = p;

this->MaxSteps = max\_steps;

this->CtrlLocalPogr = ctrl\_local\_pogr;

Mult = 0;

Div = 0;

this->eps = eps;// 1e-8;

Grid.push\_back(0);

Step.push\_back(init\_h);

Num.push\_back(init\_y);

DNum.push\_back(init\_y);

FinalNum.push\_back(init\_y);

TrueVal.push\_back(init\_y);

Main.push\_back(0);

OLP.push\_back(0);

MultStep.push\_back(Mult);

DivStep.push\_back(Div);

}

double TestWork::F(double u)

{

return 0.81 \* u;

}

double TestWork::TrueF(double x)

{

return exp(0.81 \* x);

}

double TestWork::Method(int p, double v, double h)

{

double y = 0;

switch (p)

{

case 2:

{

double k1 = F(v);

double k2 = F(v + 0.5 \* h \* k1);

y = v + h \* k2;

break;

}

case 3:

{

double k1 = F(v);

double k2 = F(v + 0.5 \* h \* k1);

double k3 = F(v + h \* (2 \* k2 - k1));

y = v + (1.0 / 6.0) \* h \* (k1 + 4 \* k2 + k3);

break;

}

case 4:

{

double k1 = F(v);

double k2 = F(v + 0.5 \* h \* k1);

double k3 = F(v + 0.5 \* h \* k2);

double k4 = F(v + h \* k3);

y = v + (1.0 / 6.0) \* h \* (k1 + 2 \* k2 + 2 \* k3 + k4);

break;

}

default:

{

y = 0;

break;

}

}

return y;

}

void TestWork::NumStep()

{

double h = Step.back();

double y = Method(p, FinalNum.back(), h);

double tmp = Method(p, FinalNum.back(), h / 2);

double y1 = Method(p, tmp, h / 2);

double s = abs(y1 - y) / (pow(2, p) - 1);

if (s > eps && CtrlLocalPogr)

{

Step.back() /= 2;

Div++;

}

else

{

const double x = Grid.back() + h;

double u = TrueF(x);

if (s < eps / pow(2, p + 1) && CtrlLocalPogr)

{

h \*= 2;

Mult++;

}

double S = s \* pow(2, p);

double y\_corr = y + S;

Grid.push\_back(x);

Step.push\_back(h);

Num.push\_back(y);

DNum.push\_back(y1);

FinalNum.push\_back(y\_corr);

TrueVal.push\_back(u);

Main.push\_back(s);

OLP.push\_back(S);

MultStep.push\_back(Mult);

DivStep.push\_back(Div);

}

}

void TestWork::DoStep()

{

int n = 0;

while (Grid.back() + Step.back() < RightBorder && n < MaxSteps)

{

NumStep();

n++;

}

if (RightBorder - Grid.back() > 1e-8 && Grid.back() + Step.back() > RightBorder)

{

Step.back() = RightBorder - Grid.back();

NumStep();

}

}

void TestWork::Out(std::ostream& os)

{

setlocale(LC\_ALL, "rus");

unsigned long long n = Grid.size();

os.precision(9);

os << "|" << std::setw(5) << "i" << "|" << std::setw(13) << "Xi" << "|" << std::setw(21) << "Vi" << "|"

<< std::setw(21) << "V2i" << "|" << std::setw(27) << "Vi-V2i" << "|" << std::setw(27) << "ОЛП" << "|"

<< std::setw(13) << "Hi" << "|" << std::setw(5) << "C1" << "|" << std::setw(5) << "C2" << "|"

<< std::setw(21) << "Ui" << "|" << std::setw(27) << "|Ui - Vi|" << "|" << std::endl;

os << "|-----+-------------+---------------------+---------------------+---------------------------+---------------------------+-------------+-----+-----+---------------------+---------------------------|" << std::endl;

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

{

os << "|" << std::setw(5) << i << "|" << std::setw(13) << Grid[i] << "|" << std::setw(21) << Num[i] << "|"

<< std::setw(21) << DNum[i] << "|" << std::setw(27) << Main[i] << "|" << std::setw(27) << OLP[i] << "|"

<< std::setw(13) << Step[i] << "|" << std::setw(5) << DivStep[i] << "|" << std::setw(5) << MultStep[i] << "|"

<< std::setw(21) << TrueVal[i] << "|" << std::setw(27) << abs(TrueVal[i] - Num[i]) << "|" << std::endl;

}

os << "|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << std::endl;

double olp\_max = OLP[0];

double h\_max = Step[0], h\_min = Step[0], h\_max\_ind = Grid[0], h\_min\_ind = Grid[0];

double E\_max = abs(TrueVal[0] - Num[0]), ind = Grid[0];

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

{

if (olp\_max < OLP[i]) olp\_max = OLP[i];

if (h\_max < Step[i]) { h\_max = Step[i]; h\_max\_ind = Grid[i]; }

if (h\_min > Step[i]) { h\_min = Step[i]; h\_min\_ind = Grid[i]; }

if (E\_max < abs(TrueVal[i] - Num[i])) { E\_max = abs(TrueVal[i] - Num[i]); ind = Grid[i]; }

}

os << "\nКоличество шагов (N) = " << n - 1 << "\nb-xn = " << RightBorder - Grid.back() << "\nmax|ОЛП| = " << olp\_max

<< "\nЧисло делений шага (с1) = " << Div << "\nЧисло умножений шага (с2) = " << Mult

<< "\nmax{Hi} = " << h\_max << " при x = " << h\_max\_ind << "\nmin{Hi} = " << h\_min << " при x = " << h\_min\_ind

<< "\nmax|Ui - Vi| = " << E\_max << " при x = " << ind << std::endl;

}

void TestWork::Draw()

{

plt::figure(1);

plt::plot(Grid, Num, { {"color", "green"} });

plt::xlabel("x");

plt::ylabel("V");

plt::title("Numerical solution");

plt::savefig("Numerical.png");

plt::figure(2);

plt::plot(Grid, TrueVal, { {"color", "purple"} });

plt::xlabel("x");

plt::ylabel("U");

plt::title("True solution");

plt::savefig("True.png");

plt::show();

}

Файл MainWork\_1.h

#pragma once

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

#include <fstream>

#include "matplotlibcpp.h"

namespace plt = matplotlibcpp;

class MainWork\_1

{

public:

std::vector<double> Grid;

std::vector<double> Step;

std::vector<double> Num;

std::vector<double> DNum;

std::vector<double> FinalNum;

std::vector<double> Main;

std::vector<double> OLP;

std::vector<double> MultStep;

std::vector<double> DivStep;

double RightBorder;

double eps;

int MaxSteps;

bool CtrlLocalPogr;

int p;

int Mult;

int Div;

MainWork\_1(double init\_y, double init\_h, double right\_border, int p, int steps, bool ctrl\_local\_pogr, double eps);

double F(double x, double u);

double Method(int p, double x, double v, double h);

void NumStep();

void DoStep();

void Out(std::ostream& os);

void Draw();

};

Файл MainWork\_1.cpp

#include "MainWork\_1.h"

MainWork\_1::MainWork\_1(double init\_y, double init\_h, double right\_border, int p, int max\_steps, bool ctrl\_local\_pogr, double eps)

{

this->RightBorder = right\_border;

this->p = p;

this->MaxSteps = max\_steps;

this->CtrlLocalPogr = ctrl\_local\_pogr;

this->eps = eps;

Mult = 0;

Div = 0;

Grid.push\_back(1);

Step.push\_back(init\_h);

Num.push\_back(init\_y);

DNum.push\_back(init\_y);

FinalNum.push\_back(init\_y);

Main.push\_back(0);

OLP.push\_back(0);

MultStep.push\_back(Mult);

DivStep.push\_back(Div);

}

double MainWork\_1::F(double x, double u)

{

return (x \* u + pow(u, 3)) / pow(x, 2);

}

double MainWork\_1::Method(int p, double x, double v, double h)

{

double y = 0;

switch (p)

{

case 2:

{

double k1 = F(x, v);

double k2 = F(x + 0.5 \* h, v + 0.5 \* h \* k1);

y = v + h \* k2;

break;

}

case 3:

{

double k1 = F(x, v);

double k2 = F(x + 0.5 \* h, v + 0.5 \* h \* k1);

double k3 = F(x + 0.5 \* h, v + h \* (2 \* k2 - k1));

y = v + (1.0 / 6.0) \* h \* (k1 + 4 \* k2 + k3);

break;

}

case 4:

{

double k1 = F(x, v);

double k2 = F(x + 0.5 \* h, v + 0.5 \* h \* k1);

double k3 = F(x + 0.5 \* h, v + 0.5 \* h \* k2);

double k4 = F(x + 0.5 \* h, v + h \* k3);

y = v + (1.0 / 6.0) \* h \* (k1 + 2 \* k2 + 2 \* k3 + k4);

break;

}

default:

{

y = 0;

break;

}

}

return y;

}

void MainWork\_1::NumStep()

{

double h = Step.back();

double y = Method(p, Grid.back(), FinalNum.back(), h);

double tmp = Method(p, Grid.back(), FinalNum.back(), h / 2);

double y1 = Method(p, Grid.back() + h / 2, tmp, h / 2);

double s = abs(y1 - y) / (pow(2, p) - 1);

if (s > eps && CtrlLocalPogr)

{

Step.back() /= 2;

Div++;

}

else

{

const double x = Grid.back() + h;

if (s < eps / pow(2, p + 1) && CtrlLocalPogr)

{

h \*= 2;

Mult++;

}

double S = s \* pow(2, p);

double y\_corr = y + S;

Grid.push\_back(x);

Step.push\_back(h);

Num.push\_back(y);

DNum.push\_back(y1);

FinalNum.push\_back(y\_corr);

Main.push\_back(s);

OLP.push\_back(S);

MultStep.push\_back(Mult);

DivStep.push\_back(Div);

}

}

void MainWork\_1::DoStep()

{

int n = 0;

while (Grid.back() + Step.back() < RightBorder && n < MaxSteps)

{

NumStep();

n++;

}

if (RightBorder - Grid.back() > 1e-8 && Grid.back() + Step.back() > RightBorder)

{

Step.back() = RightBorder - Grid.back();

NumStep();

}

}

void MainWork\_1::Out(std::ostream& os)

{

unsigned long long n = Grid.size();

os.precision(9);

os << "|" << std::setw(5) << "i" << "|" << std::setw(13) << "Xi" << "|" << std::setw(21) << "Vi" << "|"

<< std::setw(21) << "V2i" << "|" << std::setw(27) << "Vi-V2i" << "|" << std::setw(27) << "ОЛП" << "|"

<< std::setw(20) << "Hi" << "|" << std::setw(5) << "C1" << "|" << std::setw(5) << "C2" << "|" << std::endl;

os << "|-----+-------------+---------------------+---------------------+---------------------------+---------------------------+--------------------+-----+-----|" << std::endl;

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

{

os << "|" << std::setw(5) << i << "|" << std::setw(13) << Grid[i] << "|" << std::setw(21) << Num[i] << "|"

<< std::setw(21) << DNum[i] << "|" << std::setw(27) << Main[i] << "|" << std::setw(27) << OLP[i] << "|"

<< std::setw(20) << Step[i] << "|" << std::setw(5) << DivStep[i] << "|" << std::setw(5) << MultStep[i] << "|"

<< std::endl;

}

os << "|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|" << std::endl;

double olp\_max = OLP[0];

double h\_max = Step[0], h\_min = Step[0], h\_max\_ind = Grid[0], h\_min\_ind = Grid[0];

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

{

if (olp\_max < OLP[i]) olp\_max = OLP[i];

if (h\_max < Step[i]) { h\_max = Step[i]; h\_max\_ind = Grid[i]; }

if (h\_min > Step[i]) { h\_min = Step[i]; h\_min\_ind = Grid[i]; }

}

os << "Количество шагов (N) = " << n - 1 << "\nb-xn = " << RightBorder - Grid.back() << "\nmax|ОЛП| = " << olp\_max

<< "\nЧисло делений шага (с1) = " << Div << "\nЧисло умножений шага (с2) = " << Mult

<< "\nmax{Hi} = " << h\_max << " при x = " << h\_max\_ind << "\nmin{Hi} = " << h\_min << " при x = " << h\_min\_ind << std::endl;

}

void MainWork\_1::Draw()

{

plt::plot(Grid, Num, { {"color", "green"} });

plt::xlabel("x");

plt::ylabel("V");

plt::title("Numerical solution");

plt::savefig("Numeric.png");

plt::show();

}

Файл MainWork\_2.h

#pragma once

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

#include <fstream>

#include "matplotlibcpp.h"

namespace plt = matplotlibcpp;

class MainWork\_2

{

public:

std::vector<double> Grid;

std::vector<double> Step;

std::vector<double> NumV;

std::vector<double> DNumV;

std::vector<double> FinalNumV;

std::vector<double> NumZ;

std::vector<double> DNumZ;

std::vector<double> FinalNumZ;

std::vector<double> MainV;

std::vector<double> MainZ;

std::vector<double> OLP;

std::vector<double> MultStep;

std::vector<double> DivStep;

double RightBorder;

bool CtrlLocalPogr;

int MaxSteps;

double eps;

int p;

int Mult;

int Div;

MainWork\_2(double init\_y, double init\_z, double init\_h, double right\_border, int p, int steps, bool ctrl\_local\_pogr, double eps);

double F1(double x, double u, double z);

double F2(double x, double u, double z);

double Method1(int p, double x, double v, double z, double h);

double Method2(int p, double x, double v, double z, double h);

void NumStep();

void DoStep();

void Out(std::ostream& os);

void Draw();

};

Файл MainWork\_2.cpp

#include "MainWork\_2.h"

MainWork\_2::MainWork\_2(double init\_y, double init\_z, double init\_h, double right\_border,

int p, int max\_steps, bool ctrl\_local\_pogr, double eps)

{

this->RightBorder = right\_border;

this->p = p;

this->MaxSteps = max\_steps;

this->CtrlLocalPogr = ctrl\_local\_pogr;

this->eps = eps;

Mult = 0;

Div = 0;

Grid.push\_back(0);

Step.push\_back(init\_h);

NumV.push\_back(init\_y);

DNumV.push\_back(init\_y);

FinalNumV.push\_back(init\_y);

MainV.push\_back(0);

NumZ.push\_back(init\_z);

DNumZ.push\_back(init\_z);

FinalNumZ.push\_back(init\_z);

MainZ.push\_back(0);

OLP.push\_back(0);

MultStep.push\_back(Mult);

DivStep.push\_back(Div);

}

double MainWork\_2::F1(double x, double u, double z)

{

return (u + z) \* x;

}

double MainWork\_2::F2(double x, double u, double z)

{

return (-u + z) \* x;

}

double MainWork\_2::Method1(int p, double x, double v, double z, double h)

{

double vn = 0;

switch (p)

{

case 2:

{

double k1 = F1(x, v, z);

double l1 = F2(x, v, z);

double k2 = F1(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double l2 = F2(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

vn = v + h \* k2;

break;

}

case 3:

{

double k1 = F1(x, v, z);

double l1 = F2(x, v, z);

double k2 = F1(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double l2 = F2(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double k3 = F1(x + 0.5 \* h, v + h \* (2 \* k2 - k1), z + h \* (2 \* l2 - l1));

double l3 = F2(x + 0.5 \* h, v + h \* (2 \* k2 - k1), z + h \* (2 \* l2 - l1));

vn = v + (1.0 / 6.0) \* h \* (k1 + 4 \* k2 + k3);

break;

}

case 4:

{

double k1 = F1(x, v, z);

double l1 = F2(x, v, z);

double k2 = F1(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double l2 = F2(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double k3 = F1(x + 0.5 \* h, v + 0.5 \* h \* k2, z + 0.5 \* h \* l2);

double l3 = F2(x + 0.5 \* h, v + 0.5 \* h \* k2, z + 0.5 \* h \* l2);

double k4 = F1(x + h, v + h \* k3, z + h \* l3);

double l4 = F2(x + h, v + h \* k3, z + h \* l3);

vn = v + (1.0 / 6.0) \* h \* (k1 + 2 \* k2 + 2 \* k3 + k4);

break;

}

default:

{

vn = 0;

break;

}

}

return vn;

}

double MainWork\_2::Method2(int p, double x, double v, double z, double h)

{

double zn = 0;

switch (p)

{

case 2:

{

double k1 = F1(x, v, z);

double l1 = F2(x, v, z);

double k2 = F1(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double l2 = F2(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

zn = v + h \* l2;

break;

}

case 3:

{

double k1 = F1(x, v, z);

double l1 = F2(x, v, z);

double k2 = F1(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double l2 = F2(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double k3 = F1(x + 0.5 \* h, v + h \* (2 \* k2 - k1), z + h \* (2 \* l2 - l1));

double l3 = F2(x + 0.5 \* h, v + h \* (2 \* k2 - k1), z + h \* (2 \* l2 - l1));

zn = z + (1.0 / 6.0) \* h \* (l1 + 4 \* l2 + l3);

break;

}

case 4:

{

double k1 = F1(x, v, z);

double l1 = F2(x, v, z);

double k2 = F1(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double l2 = F2(x + 0.5 \* h, v + 0.5 \* h \* k1, z + 0.5 \* h \* l1);

double k3 = F1(x + 0.5 \* h, v + 0.5 \* h \* k2, z + 0.5 \* h \* l2);

double l3 = F2(x + 0.5 \* h, v + 0.5 \* h \* k2, z + 0.5 \* h \* l2);

double k4 = F1(x + h, v + h \* k3, z + h \* l3);

double l4 = F2(x + h, v + h \* k3, z + h \* l3);

zn = z + (1.0 / 6.0) \* h \* (l1 + 2 \* l2 + 2 \* l3 + l4);

break;

}

default:

{

zn = 0;

break;

}

}

return zn;

}

void MainWork\_2::NumStep()

{

double h = Step.back();

//вычисление с шагом h

double v = Method1(p, Grid.back(), FinalNumV.back(), FinalNumZ.back(), h);

double z = Method2(p, Grid.back(), FinalNumV.back(), FinalNumZ.back(), h);

//вычисление с шагом h/2

double tmp\_v = Method1(p, Grid.back(), FinalNumV.back(), FinalNumZ.back(), h / 2);

double tmp\_z = Method2(p, Grid.back(), FinalNumV.back(), FinalNumZ.back(), h / 2);

double v1 = Method1(p, Grid.back() + h / 2, tmp\_v, tmp\_z, h / 2);

double z1 = Method2(p, Grid.back() + h / 2, tmp\_v, tmp\_z, h / 2);

double s\_v = abs(v1 - v) / (pow(2, p) - 1);

double s\_z = abs(z1 - z) / (pow(2, p) - 1);

double s = std::max(s\_v, s\_z);

if (s > eps && CtrlLocalPogr)

{

Step.back() /= 2;

Div++;

}

else

{

double x = Grid.back() + h;

if (s < eps / pow(2, p + 1) && CtrlLocalPogr)

{

h \*= 2;

Mult++;

}

double S = s \* pow(2, p);

Grid.push\_back(x);

Step.push\_back(h);

NumV.push\_back(v);

DNumV.push\_back(v1);

FinalNumV.push\_back(v + s\_v \* pow(2, p));

MainV.push\_back(s\_v);

NumZ.push\_back(z);

DNumZ.push\_back(z1);

FinalNumZ.push\_back(z + s\_z \* pow(2, p));

MainZ.push\_back(s\_z);

OLP.push\_back(S);

MultStep.push\_back(Mult);

DivStep.push\_back(Div);

}

}

void MainWork\_2::DoStep()

{

int n = 0;

while (Grid.back() + Step.back() <= RightBorder && n < MaxSteps)

{

NumStep();

n++;

}

if (RightBorder - Grid.back() > 1e-8 && Grid.back() + Step.back() >= RightBorder)

{

Step.back() = RightBorder - Grid.back();

NumStep();

}

}

void MainWork\_2::Out(std::ostream& os)

{

unsigned long long n = Grid.size();

os.precision(9);

os << "|" << std::setw(5) << "i" << "|" << std::setw(13) << "Xi" << "|" << std::setw(21) << "Vi" << "|"

<< std::setw(21) << "V2i" << "|" << std::setw(21) << "Vi-V2i" << "|" << std::setw(21) << "Zi" << "|"

<< std::setw(21) << "Z2i" << "|" << std::setw(21) << "Zi-Z2i" << "|" << std::setw(21) << "ОЛП" << "|"

<< std::setw(13) << "Hi" << "|" << std::setw(5) << "C1" << "|" << std::setw(5) << "C2" << "|" << std::endl;

os << "|-----+-------------+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+-------------+-----+-----|" << std::endl;

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

{

os << "|" << std::setw(5) << i << "|" << std::setw(13) << Grid[i] << "|" << std::setw(21) << NumV[i] << "|"

<< std::setw(21) << DNumV[i] << "|" << std::setw(21) << MainV[i] << "|" << std::setw(21) << NumZ[i] << "|"

<< std::setw(21) << DNumZ[i] << "|" << std::setw(21) << MainZ[i] << "|" << std::setw(21) << OLP[i] << "|"

<< std::setw(13) << Step[i] << "|" << std::setw(5) << DivStep[i] << "|" << std::setw(5) << MultStep[i] << "|"

<< std::endl;

}

os << "|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|" << std::endl;

double olp\_max = OLP[0];

double h\_max = Step[0], h\_min = Step[0], h\_max\_ind = Grid[0], h\_min\_ind = Grid[0];

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

{

if (olp\_max < OLP[i]) olp\_max = OLP[i];

if (h\_max < Step[i]) { h\_max = Step[i]; h\_max\_ind = Grid[i]; }

if (h\_min > Step[i]) { h\_min = Step[i]; h\_min\_ind = Grid[i]; }

}

os << "Количество шагов (N) = " << n - 1 << "\nb-xn = " << RightBorder - Grid.back() << "\nmax|ОЛП| = " << olp\_max

<< "\nЧисло делений шага (с1) = " << Div << "\nЧисло умножений шага (с2) = " << Mult

<< "\nmax{Hi} = " << h\_max << " при x = " << h\_max\_ind << "\nmin{Hi} = " << h\_min << " при x = " << h\_min\_ind << std::endl;

}

void MainWork\_2::Draw()

{

plt::figure(1);

plt::plot(Grid, NumV, { {"color", "green"} });

plt::xlabel("x");

plt::ylabel("V");

plt::title("Numerical solution V of the system");

plt::savefig("Numeric\_V.png");

plt::figure(2);

plt::plot(Grid, NumZ, { {"color", "purple"} });

plt::xlabel("x");

plt::ylabel("Z");

plt::title("Numerical solution Z of the system");

plt::savefig("Numeric\_Z.png");

plt::figure(3);

plt::plot(NumV, NumZ, { {"color", "navy"} });

plt::xlabel("V");

plt::ylabel("Z");

plt::title("Phase portret of the system");

plt::savefig("Phase.png");

plt::show();

}