**Лабораторная работа №1.**

Численное решение задачи Коши для ОДУ

Выполнил:

Группа: Вариант:

1. ***Постановка задачи***

(тестовая, основная №1, основная №2)

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

(запись метода, оценка погрешности, управление шагом метода)

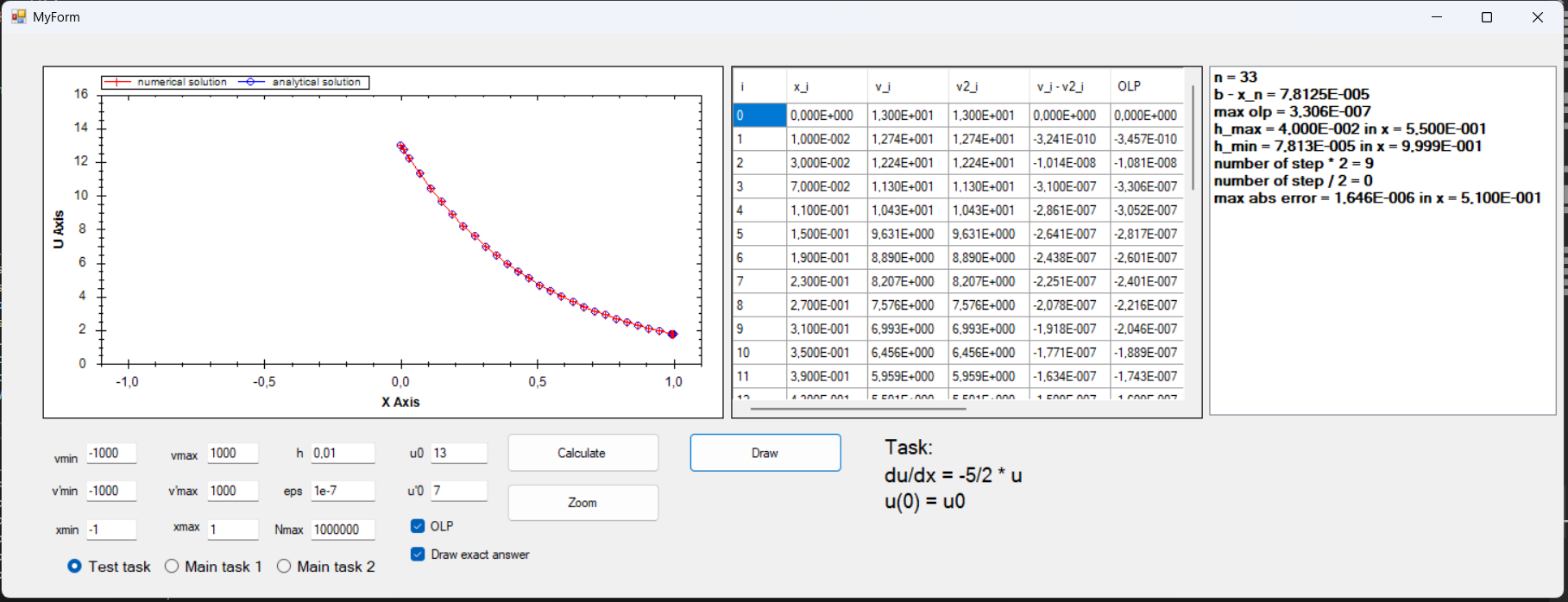
1. ***Исследование порядка сходимости для тестовой задачи***

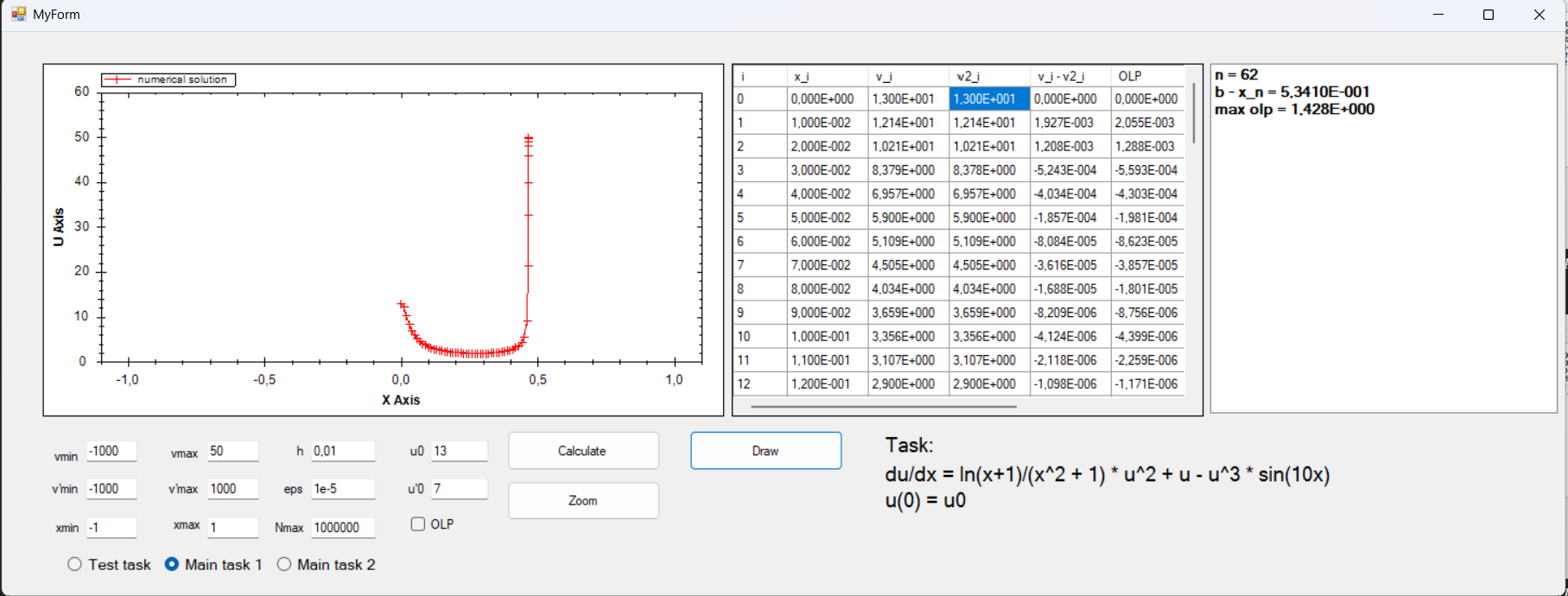
Метод

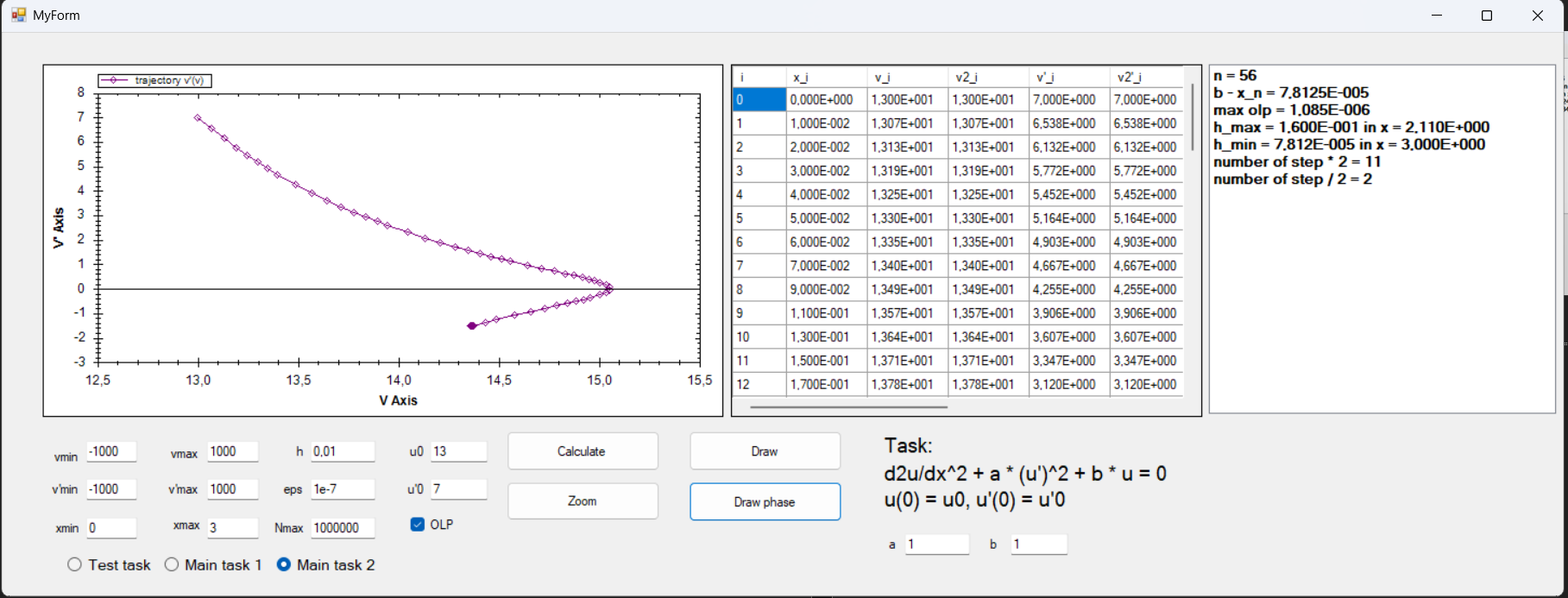
|  |  |
| --- | --- |
| *h* | *погрешность* |
| *h1* |  |
| *h2*. |
| *h3*. |
| *h4*. |
| *Порядок* |  |

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

(графики, таблицы)







1. ***Наблюдения и выводы***
2. ***Текст программы***

std::pair <std::vector < std::vector< point > >,

std::vector <std::vector <int>>> solve\_ivp(

int type, int maxN, const point& S, double h, double tol,

const point& minP, const point& maxP, bool withOLP,

double val\_a, double val\_b) {

a = val\_a;

b = val\_b;

double eps = 1e-4;

tol = std::abs(tol);

size\_t n = S.V.size();

Answer answer(type);

Rhs rhs(type);

point maxPeps(n), minPeps(n);

std::vector<double> One(n);

for (size\_t i = 0; i < One.size(); ++i)

One[i] = 1.0;

maxPeps.x = maxP.x - eps;

maxPeps.V = maxP.V + (-eps) \* One;

minPeps.x = minP.x + eps;

minPeps.V = minP.V + eps \* One;

std::vector<point> V;

std::vector<point> E;

std::vector<point> e\_appr;

std::vector<point> U;

std::vector<point> V\_half;

std::vector<int> C1vec;

std::vector<int> C2vec;

int p = 4;

point curP(n), tmp1P(n), tmp2P(n);

curP = S;

point curE(n);

point cure(n);

point curU(n);

int C1, C2;

bool abort = false;

bool next;

bool pomitsya = true;

double Spar;

curE.x = S.x;

cure.x = S.x;

curE.V = One - One;

cure.V = One - One;

E.push\_back(curE);

e\_appr.push\_back(cure);

V.push\_back(S);

U.push\_back(S);

V\_half.push\_back(S);

C1vec.push\_back(0);

C2vec.push\_back(0);

while (!abort) {

abort = false;

next = false;

C1 = 0;

C2 = 0;

while (!next) {

next = true;

curP = RK4(curP, h, rhs);

if (isless\_all(curP, maxPeps) && ismore\_all(curP, minPeps)) {

}

else if (ismore\_any(curP, maxP) || isless\_any(curP, minP)) {

curP = V[V.size() - 1];

if (pomitsya) {

h /= 2;

continue;

}

else {

abort = true;

break;

}

}

else {

abort = true;

}

tmp1P = RK4(V[V.size() - 1], h / 2, rhs);

tmp2P = RK4(tmp1P, h / 2, rhs);

if (ismore\_any(tmp2P, maxP) || isless\_any(tmp2P, minP)) {

curP = V[V.size() - 1];

if (pomitsya) {

h /= 2;

continue;

}

else {

tmp2P = V\_half[V\_half.size() - 1];

break;

}

}

cure.V = tmp2P.V - curP.V;

if (withOLP) {

Spar = norm(cure.V) / (double((1ull << p) - 1));

if (Spar > tol && std::abs(h) > 1e-16) {

h /= 2;

next = false;

curP = V[V.size() - 1];

++C1;

}

else if (Spar < tol / double(1ull << (p + 1))) {

h \*= 2;

++C2;

}

}

}

curU.x = curP.x;

cure.x = curP.x;

curE.x = curP.x;

curU.V = answer(curP.x, S.V);

cure.V = (double(1ull << p) / ((1ull << p) - 1)) \* (tmp2P.V - curP.V);

curE.V = curU.V - curP.V;

if (curP.x != V[V.size() - 1].x) {

U.push\_back(curU);

E.push\_back(curE);

e\_appr.push\_back(cure);

V.push\_back(curP);

V\_half.push\_back(tmp2P);

C1vec.push\_back(C1);

C2vec.push\_back(C2);

if (V.size() >= maxN) break;

}

}

std::vector <std::vector <point> > points;

points.push\_back(V);

points.push\_back(E);

points.push\_back(e\_appr);

points.push\_back(U);

points.push\_back(V\_half);

std::vector< std::vector <int> >C;

C.push\_back(C1vec);

C.push\_back(C2vec);

std::pair <std::vector < std::vector< point > >, std::vector <std::vector <int>>> ret(points, C);

return ret;

}