**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)

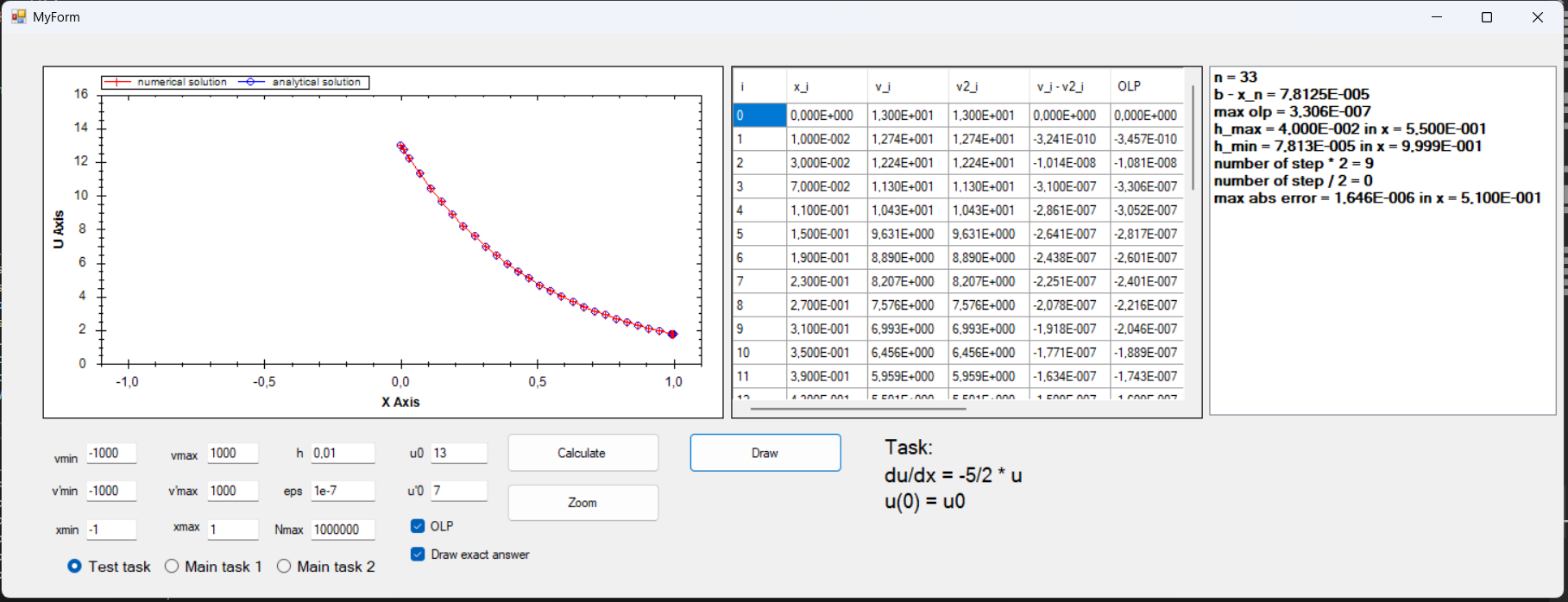
1. ***Исследование порядка сходимости для тестовой задачи***

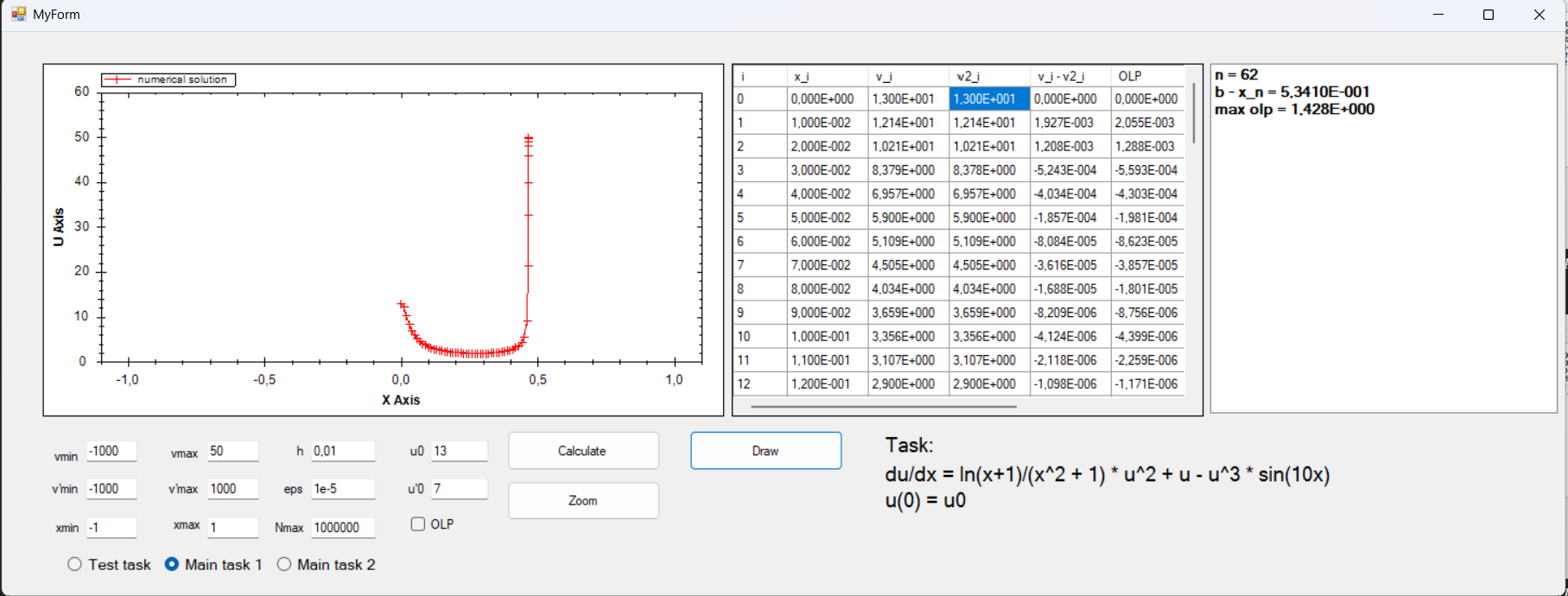
Метод

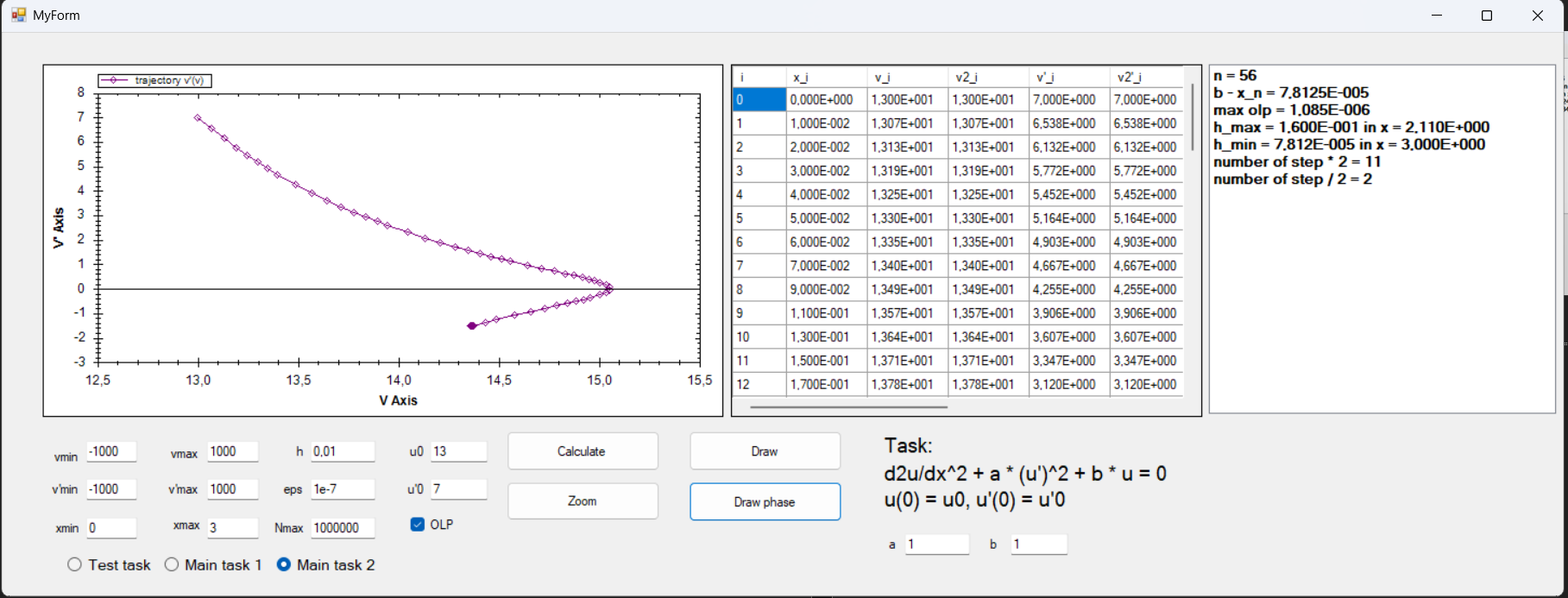
|  |  |
| --- | --- |
| *h* | *погрешность* |
| *h1* |  |
| *h2*. |
| *h3*. |
| *h4*. |
| *Порядок* |  |

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 eps, const point& minP,

const point& maxP, bool withOLP,

double val\_a, double val\_b, double eps\_gr) {

a = val\_a;

b = val\_b;

// OLP mode, border parameter

double delta = eps\_gr;

eps = std::abs(eps);

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

Answer answer(type);

Rhs rhs(type);

// declaring and initializing all vars

point maxPdelta(n), minPdelta(n);

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

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

One[i] = 1.0;

maxPdelta.x = maxP.x - delta;

maxPdelta.V = maxP.V + (-delta) \* One;

minPdelta.x = minP.x + delta;

minPdelta.V = minP.V + delta \* 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; // order

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 = delta != 0.0;

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, maxPdelta) && ismore\_all(curP, minPdelta)) {

}

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 > eps && std::abs(h) > 1e-16) {

h /= 2;

next = false;

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

++C1;

}

else if (Spar < eps / 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;

}

point RK4(const point& P, double h, Rhs rhs) {

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

point Res(n);

Res.x = P.x + h;

std::vector<double> k1(n);

std::vector<double> k2(n);

std::vector<double> k3(n);

std::vector<double> k4(n);

k1 = rhs(P.x, P.V);

k2 = rhs(P.x + h / 2, P.V + h / 2 \* k1);

k3 = rhs(P.x + h / 2, P.V + h / 2 \* k2);

k4 = rhs(P.x + h, P.V + h \* k3);

Res.V = P.V + h / 6 \* (1 \* k1 + 2 \* k2 + 2 \* k3 + 1 \* k4);

return Res;

}

struct Answer {

private:

int Type;

public:

Answer(int type) : Type(type) {}

std::vector<double> operator() (double x, const std::vector<double>& S) const {

switch (Type) {

case 0:

return answer0(x, S);

case 1:

return answer1(x, S);

case 2:

return answer2(x, S);

case 3:

return answer3(x, S);

default:

throw std::logic\_error("Unknown rhs");

}

}

};

bool isless\_all(const point& r, const point& l) {

if (r.x > l.x) return false;

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

if (r.V[i] > l.V[i]) return false;

return true;

}

bool ismore\_all(const point& r, const point& l) {

if (r.x < l.x) return false;

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

if (r.V[i] < l.V[i]) return false;

return true;

}

bool isless\_any(const point& r, const point& l) {

if (r.x < l.x) return true;

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

if (r.V[i] < l.V[i]) return true;

return false;

}

bool ismore\_any(const point& r, const point& l) {

if (r.x > l.x) return true;

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

if (r.V[i] > l.V[i]) return true;

return false;

}

struct Rhs {

private:

int Type;

public:

Rhs(int type) : Type(type) {}

std::vector<double> operator() (double x, const std::vector<double>& S) const {

switch (Type) {

case 1:

return rhs1(x, S);

case 2:

return rhs2(x, S);

case 3:

return rhs3(x, S);

default:

throw std::logic\_error("Unknown rhs");

}

}

};

std::vector<double> answer0(double x, const std::vector<double>& S) {

std::vector<double> Y(S.size());

double C2 = (S[0] + S[1]) / 2, C1 = (S[1] - S[0]) / 2;

std::vector<double> v1(2), v2(2);

v1[0] = -1.0; v2[0] = 1.0;

v1[1] = 1.0; v2[1] = 1.0;

Y = C1 \* exp(-1000 \* x) \* v1 + C2 \* exp(-0.01 \* x) \* v2;

return Y;

}

std::vector<double> answer1(double x, const std::vector<double>& S) {

std::vector<double> Y(S.size());

Y[0] = S[0] \* std::exp(-5 / 2 \* x);

return Y;

}

std::vector<double> answer2(double x, const std::vector<double>& S) {

std::vector<double> Y(S.size());

return Y;

}

std::vector<double> answer3(double x, const std::vector<double>& S) {

std::vector<double> Y(S.size());

return Y;

}

point::point(int N) : x(0.0) {

V.resize(N);

}

point::point(const point& p) : x(p.x), V(p.V) {}

point::point(point&& p) : x(p.x), V(std::move(p.V)) {}

point& point::operator= (const point& p) {

x = p.x;

V = p.V;

return \*this;

}

point& point::operator= (point&& p) {

x = p.x;

V = p.V;

return \*this;

}

std::vector<double> rhs1(double x, const std::vector<double>& V) {

std::vector<double> Y(V.size());

Y[0] = -5 / 2 \* V[0];

return Y;

}

std::vector<double> rhs2(double x, const std::vector<double>& V) {

std::vector<double> Y(V.size());

Y[0] = std::log(x + 1) / (x \* x + 1) \* V[0] \* V[0] + V[0] - V[0] \* V[0] \* V[0] \* std::sin(10 \* x);

return Y;

}

double a = 1.0, b = -1.0;

std::vector<double> rhs3(double x, const std::vector<double>& V) {

std::vector<double> Y(V.size());

Y[0] = V[1];

Y[1] = -a \* V[1] \* V[1] - b \* std::sin(V[0]);

return Y;

}

double norm2(const std::vector<double>& v) {

double res = 0.0;

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

res += v[i] \* v[i];

}

return std::sqrt(res);

}

double norminf(const std::vector<double>& v) {

double res = 0.0;

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

res = std::max(std::abs(v[i]), res);

}

return res;

}

double norm(const std::vector<double>& v) {

return norminf(v);

}

std::vector<double> operator+ (std::vector<double> l, std::vector<double> r) {

std::vector<double> res(l.size());

for (size\_t i = 0; i < res.size(); ++i) res[i] = l[i] + r[i];

return res;

}

std::vector<double> operator- (std::vector<double> l, std::vector<double> r) {

std::vector<double> res(l.size());

for (size\_t i = 0; i < res.size(); ++i) res[i] = l[i] - r[i];

return res;

}

std::vector<double> operator\* (double a, std::vector<double> r) {

std::vector<double> res(r.size());

for (size\_t i = 0; i < res.size(); ++i) res[i] = a \* r[i];

return res;

}