МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по курсу
“ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ”
ЗАДАНИЕ № 2

ОТЧЕТ
о выполненном задании
студента 203 учебной группы факультета ВМК МГУ
Хачатряна Артема Всеволодовича

Численные методы решения дифференциальных уравнений.

Москва, 2015

Описание программы:

В переменных программы задаются параметры алгоритмов, после чего последовательно вычисляются рекуррентные формулы. Реализован алгоритм Рунге-Кутты 2 и 4 порядка точности. Реализован метод прогонки решения краевой задачи для дифференциального уравнения второго порядка.

Тестирование:

Таблица 1, вариант 3

$$y' = -y - x^2, (x_0, y_0) = (0, 10)$$

Точное решение:

$$y(x) = -x^2 + 2x - 2 + 12e^{-x}$$

n = 10
h = 0.1

runge_kutta(2):

(0.00000;10.00000) (0.01000;9.90050) (0.02000;9.80199) (0.03000;9.70445) (0.04000;9.60788) (0.05000;9.51226) (0.06000;9.41758) (0.07000;9.32384) (0.08000;9.23101) (0.09000;9.13909) (0.10000;9.04806)

Отличие от аналитического решения:
0.00000000000000000000
0.00000174500998335707
0.00000345280643637442
0.00000512395715125246
0.00000675902230533147
0.00000835855455640511
0.00000992309913691158
0.00001145319394697114
0.00001294936964633259
0.00001441214974519291
0.00001584205069394586

runge_kutta(4):

(0.00000;10.00000) (0.01000;9.90050) (0.02000;9.80198) (0.03000;9.70445) (0.04000;9.60787) (0.05000;9.51225) (0.06000;9.41757) (0.07000;9.32383) (0.08000;9.23100) (0.09000;9.13907) (0.10000;9.04805)

Отличие от аналитического решения:

0.00000000000000000000
0.00000000000790002352
0.00000000001562210517
0.00000000002316900231
0.00000000003054343844
0.00000000003774809632

0.00000000004478562746
0.00000000005165864091
0.00000000005836971705
0.00000000006492139552
0.00000000007131618473

Невооруженным глазом заметны отличия в точности вычисления.

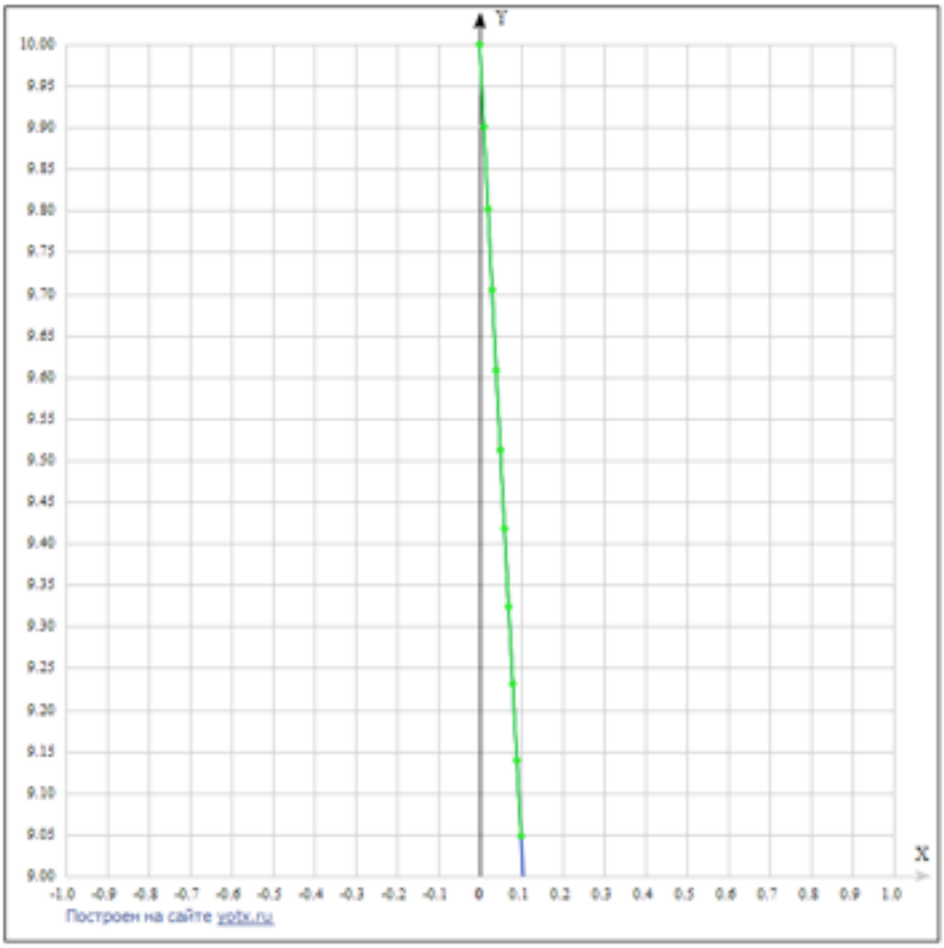


Таблица 2, вариант 17

$u'(x) = \sin(1.4u^2) - x + v$ $\quad v'(x) = x + u - 2.2v^2 + 1$ $\quad x_0 = 0$ $\quad u_0 = 1$ $\quad v_0 = 0.5$

Аналитического решения у данной системы найдено не было.

n = 10
h = 0.1

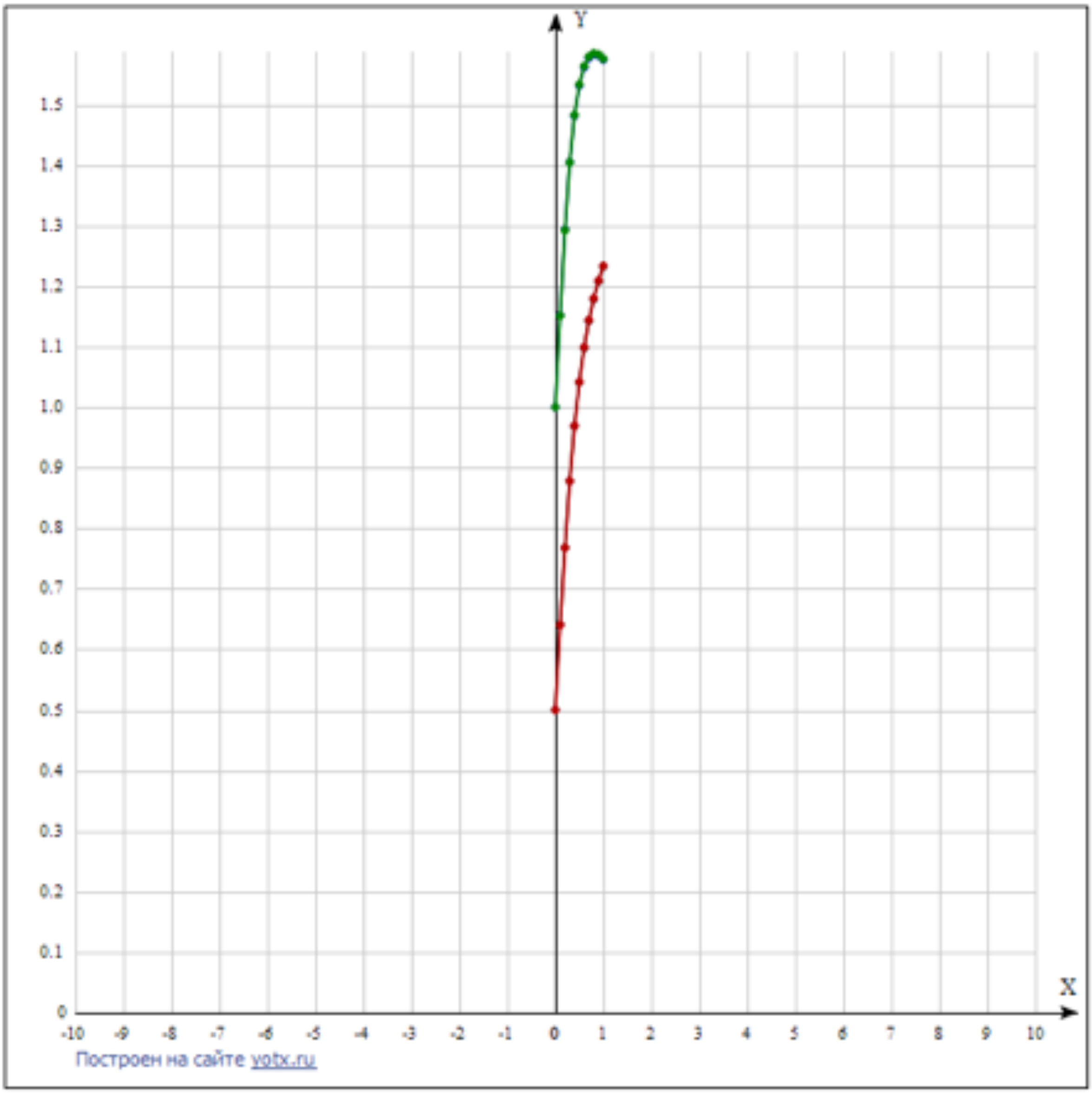
runge_kutta(2):

u(x):
(0.00000;1.00000) (0.10000;1.15215) (0.20000;1.29384) (0.30000;1.40469) (0.40000;1.48138) (0.50000;1.53088) (0.60000;1.56077) (0.70000;1.57665) (0.80000;1.58240) (0.90000;1.58080) (1.00000;1.57385)

v(x):
(0.00000;0.50000) (0.10000;0.64032) (0.20000;0.76781) (0.30000;0.87791) (0.40000;0.96851) (0.50000;1.04068) (0.60000;1.09744) (0.70000;1.14223) (0.80000;1.17811) (0.90000;1.20753) (1.00000;1.23232)

runge_kutta(4):

u(x):
(0.00000;1.00000) (0.10000;1.15111) (0.20000;1.29223) (0.30000;1.40441) (0.40000;1.48291) (0.50000;1.53346) (0.60000;1.56365) (0.70000;1.57941) (0.80000;1.58489) (0.90000;1.58295) (1.00000;1.57566)
v(x):
(0.00000;0.50000) (0.10000;0.64043) (0.20000;0.76798) (0.30000;0.87825) (0.40000;0.96933) (0.50000;1.04209) (0.60000;1.09927) (0.70000;1.14422) (0.80000;1.18003) (0.90000;1.20924) (1.00000;1.23376)



Рассмотрим дополнительный тест, в котором существует аналитическое решение системы уравнений.

$$u'(x) = -2u + 4v$$
$$v'(x) = 3v - u$$
$$x_0 = 0$$
$$u_0 = 3$$
$$v_0 = 0$$

Ее аналитическое решение:

$$u(x) = 4e^{-x} - e^{2x}$$
$$v(x) = e^{-x} - e^{2x}$$

runge_kutta(2):
u(x):
(0.00000;3.00000) (0.10000;2.40000) (0.20000;1.78770) (0.30000;1.14902) (0.40000;0.46787) (0.50000;-0.27441) (0.60000;-1.09969) (0.70000;-2.03387) (0.80000;-3.10781) (0.90000;-4.35849) (1.00000;-5.83047)
v(x):
(0.00000;0.00000) (0.10000;-0.31500) (0.20000;-0.66938) (0.30000;-1.07463) (0.40000;-1.54453) (0.50000;-2.09563) (0.60000;-2.74790) (0.70000;-3.52550) (0.80000;-4.45773) (0.90000;-5.58018) (1.00000;-6.93609)

runge_kutta(4):
u(x):
(0.00000;3.00000) (0.10000;2.39795) (0.20000;1.78311) (0.30000;1.14117) (0.40000;0.45576) (0.50000;-0.29213) (0.60000;-1.12482) (0.70000;-2.06879) (0.80000;-3.15563) (0.90000;-4.42324) (1.00000;-5.91737)
v(x):
(0.00000;0.00000) (0.10000;-0.31656) (0.20000;-0.67309) (0.30000;-1.08129) (0.40000;-1.55520) (0.50000;-2.11172) (0.60000;-2.77126) (0.70000;-3.55855) (0.80000;-4.50361) (0.90000;-5.64295) (1.00000;-7.02101)

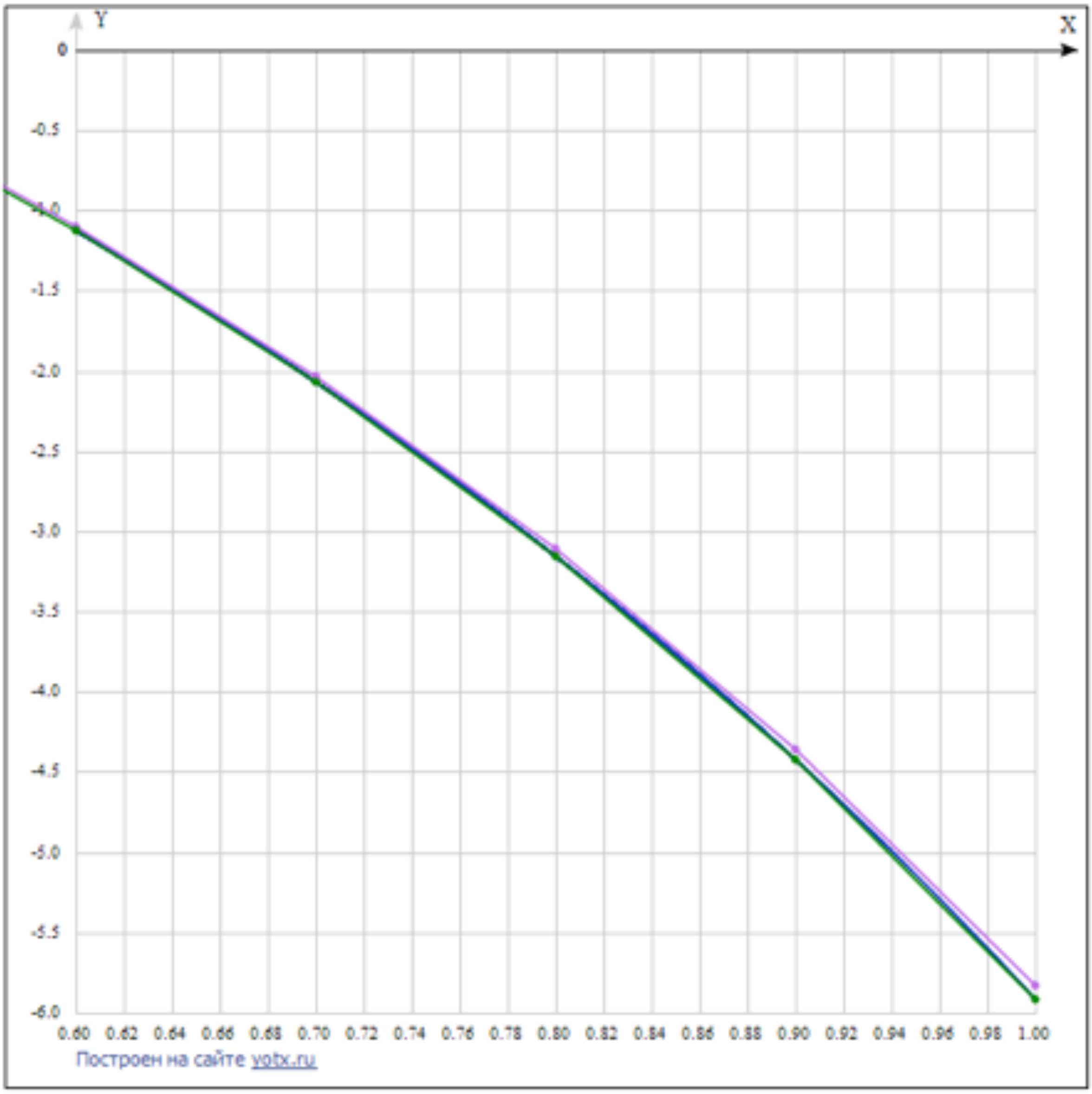
Рассмотрим отдельно u(x):

Отличие от аналитического решения:

runge_kutta(2):
0.00000000000000000000
0.00205308601633154178
0.00460168532934288402
0.00786841766363751186
0.01213398684991040397
0.01775408767101154450
0.02518068969900426454
0.03498883632267559929
0.04791038304505460107
0.06487645612301333594
0.08707085815117033721

runge_kutta(4):
0.00000000000000000000
0.00000308601633154139
0.00000733095434288320
0.00001314932434844790
0.00002107424131143453
0.00003179069609621639
0.00004617761521281570
0.00006536098480495415
0.00009078088985030059
0.00012427603809773546
0.00016819023541642239

Невооруженным глазом заметно улучшение точности. Аналогичные расчеты с v(x).



Метод прогонки:

Для проверки корректности алгоритма тестирование проводилось не только на предложенных тестах, но и на тестах из головы, имеющих аналитическое решение.

$$y''(x) - 7y'(x) + 12y(x) = 3e^{4x}$$
$$y(0.7) = 0.5$$
$$2y(1) + 3y'(1) = 1.2$$

Решение:

$$y(x) = e^{3x}(e^{x}(3x - 5.83027) + 7.56306)$$

N = 10

(0.70000;0.50000) (0.73000;0.18627) (0.76000;-0.14740) (0.79000;-0.49450) (0.82000;-0.84598) (0.85000;-1.18954) (0.88000;-1.50897) (0.91000;-1.78322) (0.94000;-1.98539) (0.97000;-2.08147) (1.00000;-2.02889)

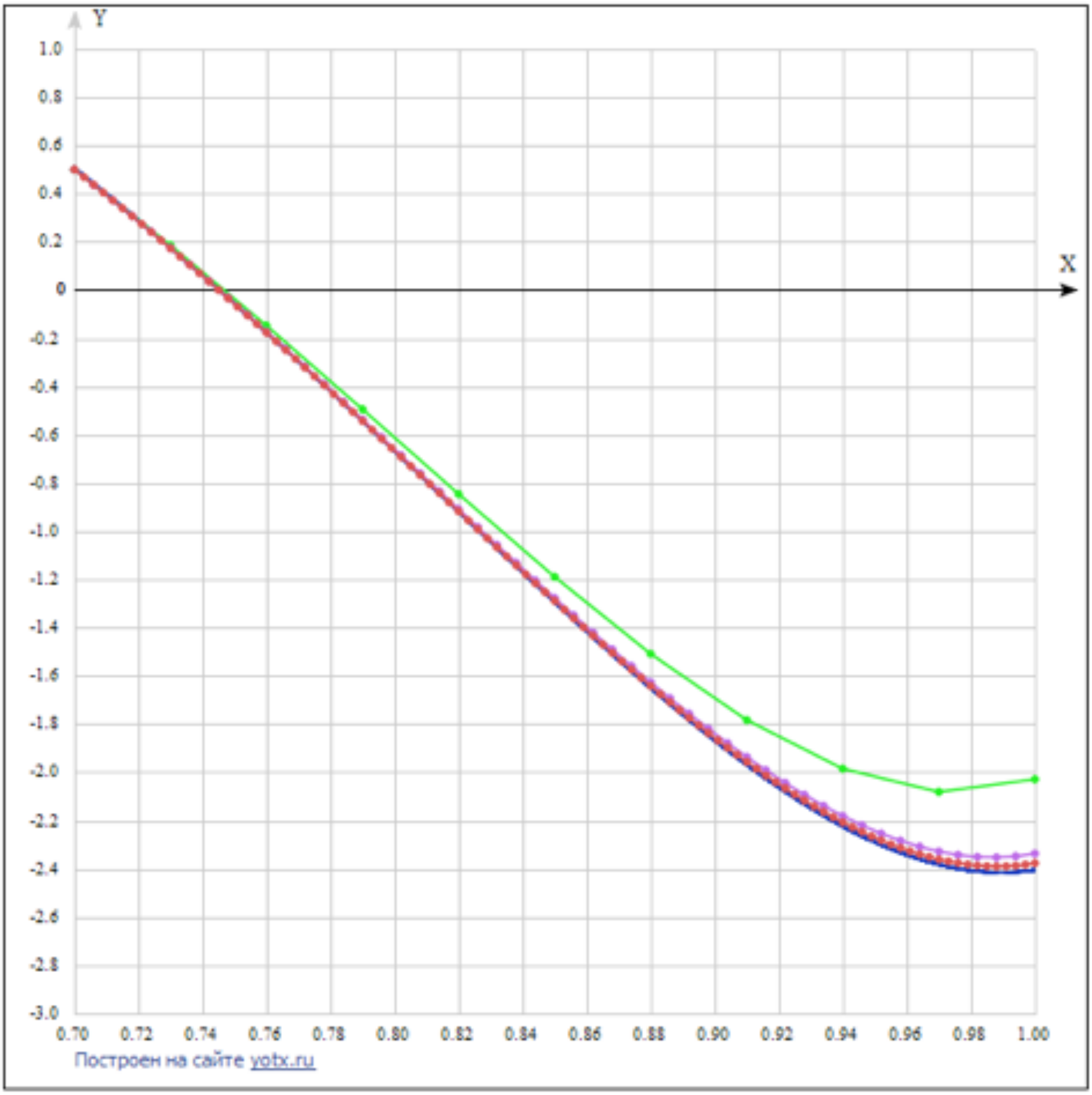
N = 50

(0.70000;0.50000) (0.70600;0.43698) (0.71200;0.37293) (0.71800;0.30786) (0.72400;0.24182) (0.73000;0.17484) (0.73600;0.10695) (0.74200;0.03819) (0.74800;-0.03138) (0.75400;-0.10173) (0.76000;-0.17280) (0.76600;-0.24454) (0.77200;-0.31688) (0.77800;-0.38977) (0.78400;-0.46314) (0.79000;-0.53690) (0.79600;-0.61100) (0.80200;-0.68534) (0.80800;-0.75985) (0.81400;-0.83441) (0.82000;-0.90895) (0.82600;-0.98335) (0.83200;-1.05751) (0.83800;-1.13130) (0.84400;-1.20462) (0.85000;-1.27731) (0.85600;-1.34926) (0.86200;-1.42031) (0.86800;-1.49032) (0.87400;-1.55912) (0.88000;-1.62654) (0.88600;-1.69242) (0.89200;-1.75655) (0.89800;-1.81875) (0.90400;-1.87881) (0.91000;-1.93652) (0.91600;-1.99164) (0.92200;-2.04395) (0.92800;-2.09319) (0.93400;-2.13910) (0.94000;-2.18142) (0.94600;-2.21985) (0.95200;-2.25409) (0.95800;-2.28384) (0.96400;-2.30876) (0.97000;-2.32852) (0.97600;-2.34276) (0.98200;-2.35110) (0.98800;-2.35316) (0.99400;-2.34853) (1.00000;-2.33678)

N = 100

(0.70000;0.50000) (0.70300;0.46848) (0.70600;0.43670) (0.70900;0.40465) (0.71200;0.37234) (0.71500;0.33978) (0.71800;0.30697) (0.72100;0.27391) (0.72400;0.24060) (0.72700;0.20706) (0.73000;0.17328) (0.73300;0.13927) (0.73600;0.10504) (0.73900;0.07059) (0.74200;0.03592) (0.74500;0.00105) (0.74800;-0.03403) (0.75100;-0.06931) (0.75400;-0.10478) (0.75700;-0.14043) (0.76000;-0.17626) (0.76300;-0.21226) (0.76600;-0.24842) (0.76900;-0.28474) (0.77200;-0.32121) (0.77500;-0.35781) (0.77800;-0.39456) (0.78100;-0.43142) (0.78400;-0.46840) (0.78700;-0.50548) (0.79000;-0.54266) (0.79300;-0.57993) (0.79600;-0.61728) (0.79900;-0.65469) (0.80200;-0.69215) (0.80500;-0.72966) (0.80800;-0.76721) (0.81100;-0.80478) (0.81400;-0.84235) (0.81700;-0.87993) (0.82000;-0.91749) (0.82300;-0.95502) (0.82600;-0.99251) (0.82900;-1.02994) (0.83200;-1.06730) (0.83500;-1.10458) (0.83800;-1.14176) (0.84100;-1.17883) (0.84400;-1.21576) (0.84700;-1.25255) (0.85000;-1.28918) (0.85300;-1.32562) (0.85600;-1.36186) (0.85900;-1.39789) (0.86200;-1.43368) (0.86500;-1.46922) (0.86800;-1.50448) (0.87100;-1.53945) (0.87400;-1.57410) (0.87700;-1.60842) (0.88000;-1.64238) (0.88300;-1.67596) (0.88600;-1.70913) (0.88900;-1.74188) (0.89200;-1.77418) (0.89500;-1.80600) (0.89800;-1.83732) (0.90100;-1.86811) (0.90400;-1.89835) (0.90700;-1.92802) (0.91000;-1.95707) (0.91300;-1.98549) (0.91600;-2.01324) (0.91900;-2.04030) (0.92200;-2.06663) (0.92500;-2.09220) (0.92800;-2.11698) (0.93100;-2.14094) (0.93400;-2.16405) (0.93700;-2.18627)

(0.94000;-2.20756) (0.94300;-2.22789) (0.94600;-2.24722) (0.94900;-2.26552) (0.95200;-2.28274) (0.95500;-2.29885) (0.95800;-2.31381) (0.96100;-2.32757) (0.96400;-2.34009) (0.96700;-2.35134) (0.97000;-2.36126) (0.97300;-2.36982) (0.97600;-2.37696) (0.97900;-2.38263) (0.98200;-2.38680) (0.98500;-2.38941) (0.98800;-2.39042) (0.99100;-2.38976) (0.99400;-2.38739) (0.99700;-2.38325) (1.00000;-2.37730)



$$y'' - 3y' - \frac{y}{x} = x + 1$$
$$y'(1.2) = 1$$
$$2y(1.5) - y'(1.5) = 0.5$$ N = 10

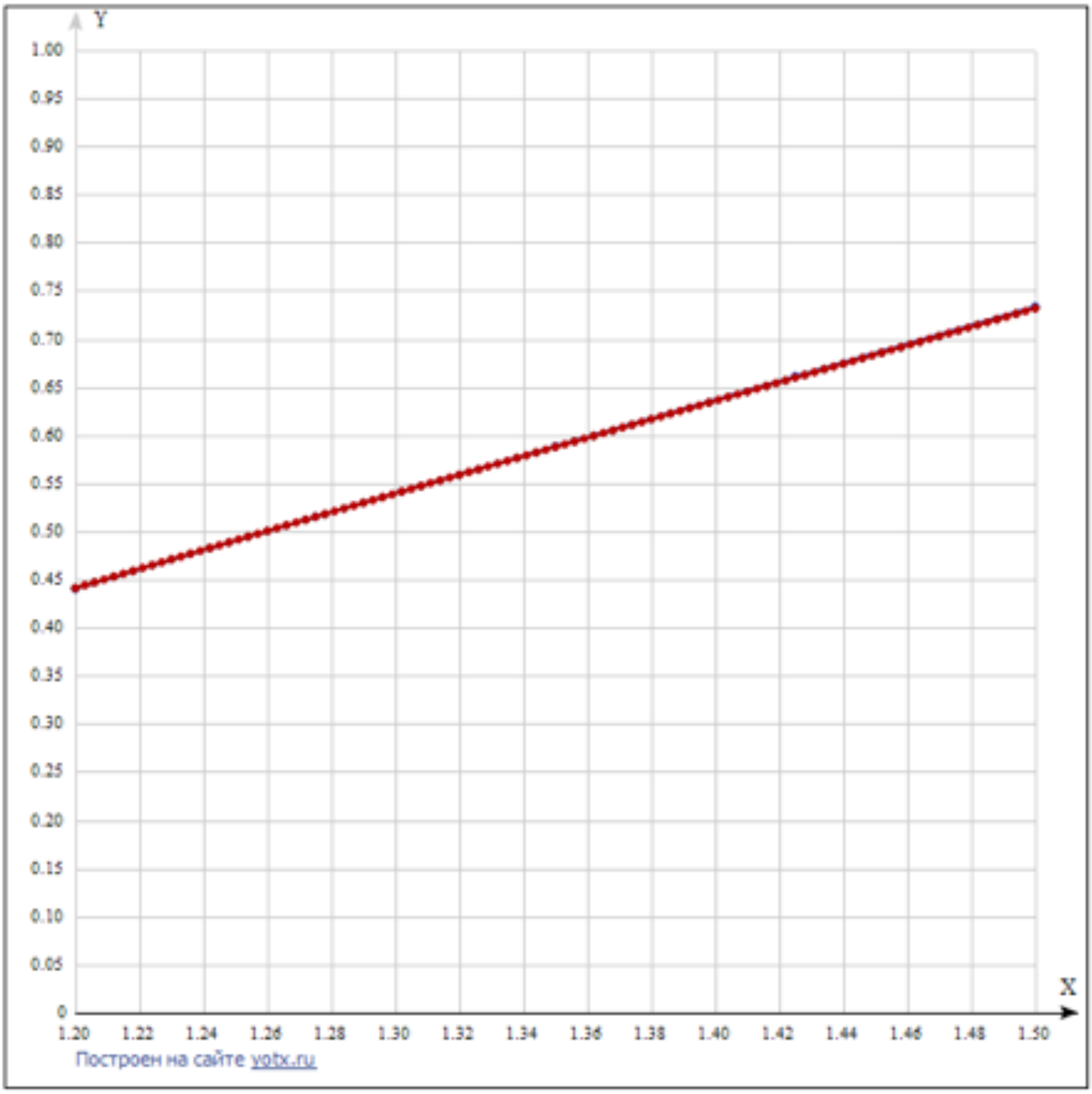
(1.20000;0.44044) (1.23000;0.47044) (1.26000;0.50011) (1.29000;0.52951) (1.32000;0.55870) (1.35000;0.58774) (1.38000;0.61668) (1.41000;0.64556) (1.44000;0.67443) (1.47000;0.70331) (1.50000;0.73225)

N = 50

(1.20000;0.44088) (1.20600;0.44688) (1.21200;0.45286) (1.21800;0.45883) (1.22400;0.46479) (1.23000;0.47073) (1.23600;0.47666) (1.24200;0.48258) (1.24800;0.48849) (1.25400;0.49438) (1.26000;0.50027) (1.26600;0.50614) (1.27200;0.51201) (1.27800;0.51786) (1.28400;0.52371) (1.29000;0.52955) (1.29600;0.53538) (1.30200;0.54120) (1.30800;0.54702) (1.31400;0.55283) (1.32000;0.55863) (1.32600;0.56443) (1.33200;0.57022) (1.33800;0.57601) (1.34400;0.58179) (1.35000;0.58757) (1.35600;0.59335) (1.36200;0.59912) (1.36800;0.60489) (1.37400;0.61066) (1.38000;0.61642) (1.38600;0.62218) (1.39200;0.62794) (1.39800;0.63370) (1.40400;0.63946) (1.41000;0.64522) (1.41600;0.65098) (1.42200;0.65673) (1.42800;0.66249) (1.43400;0.66825) (1.44000;0.67401) (1.44600;0.67977) (1.45200;0.68553) (1.45800;0.69129) (1.46400;0.69705) (1.47000;0.70282) (1.47600;0.70859) (1.48200;0.71436) (1.48800;0.72013) (1.49400;0.72591) (1.50000;0.73169)

N = 100

(1.20000;0.44094) (1.20300;0.44394) (1.20600;0.44694) (1.20900;0.44993) (1.21200;0.45292) (1.21500;0.45591) (1.21800;0.45889) (1.22100;0.46187) (1.22400;0.46484) (1.22700;0.46781) (1.23000;0.47078) (1.23300;0.47374) (1.23600;0.47671) (1.23900;0.47967) (1.24200;0.48262) (1.24500;0.48557) (1.24800;0.48852) (1.25100;0.49147) (1.25400;0.49442) (1.25700;0.49736) (1.26000;0.50030) (1.26300;0.50323) (1.26600;0.50617) (1.26900;0.50910) (1.27200;0.51203) (1.27500;0.51496) (1.27800;0.51788) (1.28100;0.52080) (1.28400;0.52373) (1.28700;0.52664) (1.29000;0.52956) (1.29300;0.53248) (1.29600;0.53539) (1.29900;0.53830) (1.30200;0.54121) (1.30500;0.54412) (1.30800;0.54702) (1.31100;0.54993) (1.31400;0.55283) (1.31700;0.55573) (1.32000;0.55863) (1.32300;0.56153) (1.32600;0.56443) (1.32900;0.56732) (1.33200;0.57022) (1.33500;0.57311) (1.33800;0.57600) (1.34100;0.57889) (1.34400;0.58178) (1.34700;0.58467) (1.35000;0.58756) (1.35300;0.59045) (1.35600;0.59333) (1.35900;0.59622) (1.36200;0.59910) (1.36500;0.60199) (1.36800;0.60487) (1.37100;0.60775) (1.37400;0.61063) (1.37700;0.61352) (1.38000;0.61640) (1.38300;0.61928) (1.38600;0.62216) (1.38900;0.62504) (1.39200;0.62791) (1.39500;0.63079) (1.39800;0.63367) (1.40100;0.63655) (1.40400;0.63943) (1.40700;0.64231) (1.41000;0.64518) (1.41300;0.64806) (1.41600;0.65094) (1.41900;0.65382) (1.42200;0.65669) (1.42500;0.65957) (1.42800;0.66245) (1.43100;0.66533) (1.43400;0.66821) (1.43700;0.67108) (1.44000;0.67396) (1.44300;0.67684) (1.44600;0.67972) (1.44900;0.68260) (1.45200;0.68548) (1.45500;0.68836) (1.45800;0.69124) (1.46100;0.69412) (1.46400;0.69700) (1.46700;0.69988) (1.47000;0.70277) (1.47300;0.70565) (1.47600;0.70853) (1.47900;0.71142) (1.48200;0.71430) (1.48500;0.71719) (1.48800;0.72008) (1.49100;0.72296) (1.49400;0.72585) (1.49700;0.72874) (1.50000;0.73163)



Вывод

В ходе практической работы были реализованы метод Рунге-Кутты второго и четвёртого порядков точности, применительно как к «простым» ОДУ первого порядка, разрешённым относительно производной, так и к соответствующим системам.

Тестирование показало, что метод Рунге-Кутты четвёртого порядка точности действительно намного более точный, чем метод второго порядка точности.

В применении к системам из двух ОДУ первого порядка, метод Рунге-Кутты четвёртого порядка показывает ещё большее преимущество.

Реализован и протестирован метод прогонки. Отмечено, что даже небольшое увеличение количества итераций, может существенно улучшить точность вычислений.

```
#include <iostream>
#include <vector>
#include <assert.h>
#include <cmath>
#include <algorithm>

using namespace std;

struct Point
{
    long double x, y;
    Point(long double _x = 0, long double _y = 0) {
        x = _x;
        y = _y;
    }
    bool in(long double from, long double to) {
        return from <= x && x <= to;
    }
};



vector <Point> runge_kutta(function <long double(Point)> func, Point pt, long double d_len, int iters, bool four_mode = 0)
{
    assert(d_len > 0);
    assert(iters > 0);
    vector <Point> result;
    result.push_back(pt);
    for (int i = 1; i <= iters; i++) {
        long double k1 = func(pt);
        long double k2 = func(Point(pt.x + d_len / 2, pt.y + d_len / 2 * k1));
        if (four_mode) {
            long double k3 = func(Point(pt.x + d_len / 2, pt.y + d_len / 2 * k2));
            long double k4 = func(Point(pt.x + d_len, pt.y + d_len * k3));
            pt = Point(pt.x + d_len, pt.y + d_len / 6 * (k1 + 2 * k2 + 2 * k3 + k4));
        } else {
            pt = Point(pt.x + d_len, pt.y + d_len * k2);
        }
        result.push_back(pt);
    }
    return result;
}

vector < vector<Point> > runge_kutta_sys(function <long double(long double, long double, long double)> func1, Point pt1,
                                         function <long double(long double, long double, long double)> func2, Point pt2,
                                         long double d_len, int iters, bool four_mode = 0)
{
    assert(d_len > 0);
    assert(iters > 0);
    vector <Point> result1;
    vector <Point> result2;
    result1.push_back(pt1);
    result2.push_back(pt2);
```

```
    for (int i = 1; i <= iters; i++) {
        long double k1 = func1(pt1.x, pt1.y, pt2.y);
        long double m1 = func2(pt2.x, pt1.y, pt2.y);

        long double k2 = func1(pt1.x + d_len / 2, pt1.y + d_len / 2 * k1, pt2.y + d_len / 2 * m1);
        long double m2 = func2(pt2.x + d_len / 2, pt1.y + d_len / 2 * k1, pt2.y + d_len / 2 * m1);

        if (four_mode) {
            long double k3 = func1(pt1.x + d_len / 2, pt1.y + d_len / 2 * k2, pt2.y + d_len / 2 * m2);
            long double m3 = func2(pt2.x + d_len / 2, pt1.y + d_len / 2 * k2, pt2.y + d_len / 2 * m2);
            long double k4 = func1(pt1.x + d_len, pt1.y + d_len * k3, pt2.y + d_len * m3);
            long double m4 = func2(pt2.x + d_len, pt1.y + d_len * k3, pt2.y + d_len * m3);
            pt1 = Point(pt1.x + d_len, pt1.y + d_len / 6 * (k1 + 2 * k2 + 2 * k3 + k4));
            pt2 = Point(pt2.x + d_len, pt2.y + d_len / 6 * (m1 + 2 * m2 + 2 * m3 + m4));
        } else {
            pt1 = Point(pt1.x + d_len, pt1.y + d_len * k2);
            pt2 = Point(pt2.x + d_len, pt2.y + d_len * m2);
        }
        result1.push_back(pt1);
        result2.push_back(pt2);
    }
    vector < vector<Point> > result(2);
    result[0] = result1;
    result[1] = result2;
    return result;
}

vector <Point> tridiagonal_method(  function <long double(long double)> P,
                                    function <long double(long double)> Q,
                                    function <long double(long double)> F,
                                    long double xb, long double xe,
                                    long double k1, long double k2,
                                    long double l1, long double l2,
                                    long double a, long double b, int iters)
{
    assert(iters > 0);
    long double h = (xe - xb) / iters;
    vector <long double> aa(iters + 1);
    vector <long double> bb(iters + 1);
    vector <long double> cc(iters + 1);
    vector <long double> ff(iters + 1);
    vector <long double> x(iters + 1);

    for (int i = 0; i <= iters; ++i) {
        x[i] = xb + h * i;
        aa[i] = 1 - P(x[i]) * h / 2;
        bb[i] = 1 + P(x[i]) * h / 2;
        cc[i] = 2 - Q(x[i]) * h * h;
        ff[i] = h * h * F(x[i]);
    }
    vector <long double> al(iters + 1);
    vector <long double> bet(iters + 1);
    vector <long double> y(iters + 1);
    al[1] = k2 / (k2 - k1 * h);
    bet[1] = - (a * h) / (k2 - k1 * h);
```

```
    for (int i = 1; i < iters; ++i) {
        al[i + 1] = bb[i] / (cc[i] - al[i] * aa[i]);
        bet[i + 1] = (aa[i] * bet[i] - ff[i]) / (cc[i] - al[i] * aa[i]);
    }
    y[iters] = (l2 * bet[iters] + b * h) / (l2 + h * l1 - l2 * al[iters]);
    for (int i = iters-1; i >=0; --i) {
        y[i] = al[i + 1] * y[i + 1] + bet[i + 1];
    }
    vector <Point> result;
    for (int i = 0; i <= iters; ++i) {
        result.push_back(Point(x[i], y[i]));
    }
    return result;
}

void print_result(vector <Point> res)
{
    for (int i = 0; i < (int)res.size(); i++) {
        cout.precision(5);
        cout << fixed << "(" << res[i].x << ";" << res[i].y << ") ";
    }
    cout << endl;
}

vector <long double> get_diff(vector <Point> res, function <long double(long double)> solution)
{
    vector <long double> tmp;
    for (int i = 0; i < (int)res.size(); i++) {
        cout.precision(20);
        tmp.push_back(abs(res[i].y - solution(res[i].x)));
        cout << fixed << abs(res[i].y - solution(res[i].x)) << endl;
    }
    return tmp;
}

void print_result_sys(vector < vector <Point> > res)
{
    for (int i = 0; i < (int)res.size(); i++) {
        for (int j = 0; j < (int)res[i].size(); j++) {
            cout.precision(5);
            cout << fixed << "(" << res[i][j].x << ";" << res[i][j].y << ") ";
        }
        cout << endl;
    }
}

int main(int argc, char **argv)
{

    int iters = 10;
    long double x0 = 0;
    long double y0 = 10;
    long double h = 0.01;

    auto f = [](Point p) {
```

```
        return -p.y - p.x * p.x;
    };

    auto solution = [](long double x) {
        return -x * x + 2 * x - 2 + 12 * exp(-x);
    };

    int siters = 10;
    long double sx0 = 0;
    long double sy0 = 1;
    long double sy1 = 0.5;
    long double sh = 0.1;

    auto f1 = [](long double x, long double u, long double v) {
        return sin(1.4 * u * u) - x + v;
    };

    auto f2 = [](long double x, long double u, long double v) {
        return x + u - 2.2 * v * v + 1;
    };


    int titers = 10;
    long double tx0 = 0;
    long double ty0 = 3;
    long double ty1 = 0;
    long double th = 0.1;

    auto tf1 = [](long double x, long double u, long double v) {
        return -2 * u + 4 * v;
    };
    auto tf2 = [](long double x, long double u, long double v) {
        return 3 * v - u;
    };

    auto answer = runge_kutta(f, Point(x0, y0), h, iters, 0);
    print_result(answer);
    cout << endl;
    auto d1 = get_diff(answer, solution);
    cout << endl;
    answer = runge_kutta(f, Point(x0, y0), h, iters, 1);
    print_result(answer);
    cout << endl;
    auto d2 = get_diff(answer, solution);
    cout << endl;
    for (int i = 0; i < (int)d1.size(); i++) {
        cout.precision(20);
        cout << fixed << abs(d1[i] - d2[i]) << endl;
    }
    print_result_sys(runge_kutta_sys(f1, Point(sx0, sy0), f2, Point(sx0, sy1), sh, siters, 0));
    cout << endl;
    print_result_sys(runge_kutta_sys(f1, Point(sx0, sy0), f2, Point(sx0, sy1), sh, siters, 1));
    cout << endl;
    auto solv1 = [](long double x){
        return 4 * exp(-x) - exp(2 * x);
    };
```

```
    auto answer1 = runge_kutta_sys(tf1, Point(tx0, ty0), tf2, Point(tx0, ty1), th, titers, 0);
    auto ans1 = answer1[0];
    get_diff(ans1, solv1);
    print_result_sys(answer1);
    cout << endl;
    auto answer2 = runge_kutta_sys(tf1, Point(tx0, ty0), tf2, Point(tx0, ty1), th, titers, 1);
    auto ans2 = answer2[0];
    get_diff(ans2, solv1);
    print_result_sys(answer2);
    cout << endl;


    auto P = [](long double x) {
        return -3 * x;
    };
    auto Q = [](long double x) {
        return 2;
    };
    auto F = [](long double x) {
        return 1.5;
    };

    auto sec_pow_result = tridiagonal_method(P, Q, F, 0.7, 1, 0, 1, 0.5, 1, 1.3, 2, 50);
    print_result(sec_pow_result);

    auto P1 = [](long double x) {
        return 3;
    };
    auto Q1 = [](long double x) {
        return -1 / x;
    };
    auto F1 = [](long double x) {
        return x + 1;
    };

    auto P2 = [](long double x) {
        return -7;
    };
    auto Q2 = [](long double x) {
        return 12;
    };
    auto F2 = [](long double x) {
        return 3 * exp(4 * x);
    };

    auto sec_pow_result2 = tridiagonal_method(P1, Q1, F1, 1.2, 1.5, 0, 1, 2, -1, 1, 0.5, 50);
    print_result(sec_pow_result2);

    auto sec_pow_result3 = tridiagonal_method(P2, Q2, F2, 0.7, 1, 1, 0, 2, 3, 0.5, 1.2, 100);
    print_result(sec_pow_result3);

    return 0;
}
```