МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 2

Решение задачи Коши для дифференциального уравнения первого порядка или системы дифференциальных уравнений первого порядка

Подвариант 1: 1 - 4, 2 - 14

ОТЧЁТ

о выполненном задании

студента 205 учебной группы факультета ВМК МГУ
Лабутина Антона Александровича

гор. Москва
2019 г.

Постановка задачи

Рассматривается обыкновенное дифференциальное уравнение первого порядка, разрешенное относительно производной и имеющее вид:

$$\frac{dy}{dx} = f(x,\ y),\ \ x_0 < x, \tag{1}$$

с дополнительным начальным условием, заданным в точке $x = x_0$:

$$y(x_0) = y_0 \tag{2}$$

Предполагается, что правая часть уравнения (1) функция такова, что гарантирует существование и единственность решения Коши (1)-(2).

В том случае, если рассматривается не одно дифференциальное уравнение вида (1), а система обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производных неизвестных функций, то соответствующая задача Коши имеет вид (на примере двух дифференциальных уравнений):

$$\begin{cases} \frac{dy_1}{dx} = f_1(x,\ \ y_1,\ y_2), \\ \frac{dy_2}{dx} = f_2(x,\ \ y_1,\ y_2),\ \ x > x_0 \end{cases} \tag{3}$$

Дополнительные (начальные) условия задаются в точке $x = x_0$:

$$y_1(x_0) = y_1^{(0)},\ \ y_2(x_0) = y_2^{(0)}\ . \tag{4}$$

Также предполагается, что правые части уравнений из (3) заданы так, что это гарантирует существование и единственность решения задачи Коши (3)-(4), но уже для системы обыкновенных дифференциальных уравнений первого порядка в форме, разрешенной относительно производных неизвестных функций.

Цели и задачи практической работы

1. Решить задачу Коши (1)-(2) (или (3)-(4)) наиболее известными и широко используемыми на практике методами Рунге-Кутта второго и четвертого порядка точности, аппроксимировав дифференциальную задачу соответствующей разностной схемой (на равномерной сетке); полученное конечно-разностное уравнение (или уравнение в случае системы), представляющее фактически некоторую рекуррентную формулу, просчитать численно;
2. Найти численное решение задачи и построить ее график;
3. Найти численное решение, сравнить с точным решением дифференциального уравнения (подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций, при проверке можно использовать ресурсы on-line системы http://www.wolframalpha.com, пакета Maple и т. д.).

Алгоритмы решения

Методом Рунге−Кутта второго порядка называется метод, имеющий общий вид:
$k_1 = f(x_{n-1},\ y_{n-1})$,
$k_2 = f(x_{n-1} + h\,c_2,\ y_{n-1} + h\,a_{21}k_1)$,
$x_n = x_{n-1} + h$,
$y_n = y_{n-1} + h(b_1k_1 + b_2k_2)$,
где $a_{21},\ b_1\ ,\ b_2,\ c_2$ - постоянные, подлежащие определению, h — величина шага сетки x.

Методом Рунге-Кутта четвертого порядка называется метод, имеющий общий вид:
$k_1 = f(x_{n-1},\ y_{n-1})$,
$k_2 = f\left(x_{n-1} + \frac{h}{2},\ y_{n-1} + \frac{h}{2}k_1\right)$,

$$k_3 = f\left(x_{n-1} + \frac{h}{2},\ y_{n-1} + \frac{h}{2}k_2\right),$$
$$k_4 = f\left(x_{n-1} + h,\ y_{n-1} + h\,k_3\right),$$
$$x_n = x_{n-1} + h,$$
$$y_n = y_{n-1} + \frac{h}{6}\left(k_1 + 2(k_2 + k_3) + k_2\right),$$

где h - величина шага сетки x.

Описание программы

Программе на стандартный поток ввода подаются следующие параметры:

- *test_num* – номер теста;
- *ITER_CNT* – количество итераций;
- *step* – размер шага.

В программе реализована функция *Runge_Kutta_2*, вычисляющая методом Рунге-Кутта второго порядка решение заданного уравнения, и функция *Runge_Kutta_4*, вычисляющая методом Рунге-Кутта четвертого порядка решение того же уравнения. Данные функции выводят решение уравнения на стандартный поток в виде вектора (x, $\vec{y}$).

Для того чтобы протестировать некоторую функцию, необходимо сначала добавить ее прототип в код программы. Изначально реализованы прототипы следующих функций:

- из таблицы 1 – 1, 2, 4, ;
- из таблицы 2 – 1, 2, 14.

Текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int ITER_CNT = 0; // количество итераций
int EQUATIONS_CNT; // количество уравнений
int TESTS_CNT = 6;

double step; // шаг сетки
double x_0;
double *y_0;

double
(*function[2]) (double, double *); // массив указателей на функцию

double
(*solution) (double);

double
f1_1(double x, double *y) // уравнение из таблицы 1 - 1
{
    return (3 - y[0] - x);
}

void
init_f1_1(void) // начальные условия для уравнения из таблицы 1 - 1
{
    EQUATIONS_CNT = 1;
    y_0 = malloc(EQUATIONS_CNT * sizeof(double));
    x_0 = 0.0;
    y_0[0] = 0.0;
}

double
sol_f1_1(double x)
```

```
{
    return (4 - x - 4 * exp(-x));
}

double
f1_2(double x, double *y) // уравнение из таблицы 1 - 2
{
    return (sin(x) - y[0]);
}

void
init_f1_2(void) // начальные условия для уравнения из таблицы 1 - 2
{
    EQUATIONS_CNT = 1;
    y_0 = malloc(EQUATIONS_CNT * sizeof(double));
    x_0 = 0.0;
    y_0[0] = 10.0;
}

double
sol_f1_2(double x)
{
    return (0.5 * (-cos(x) + sin(x) + 21 * exp(-x)));
}

double
f1_4(double x, double *y) // уравнение из таблицы 1 - 4
{
    return (y[0] - y[0] * x);
}

void
init_f1_4(void) // начальные условия для уравнения из таблицы 1 - 4
{
    EQUATIONS_CNT = 1;
    y_0 = malloc(EQUATIONS_CNT * sizeof(double));
    x_0 = 0.0;
    y_0[0] = 5.0;
}

double
sol_f1_4(double x)
{
    return (5 * exp(-0.5 * x * (-2 + x)));
}

double
f2_1_1(double x, double *y) // первое уравнение системы из таблицы 2 - 1
{
    return ((y[0] - y[1]) / x);
}

double
f2_1_2(double x, double *y) // второе уравнение системы из таблицы 2 - 1
{
    return ((y[0] + y[1]) / x);
}

void
init_f2_1(void) // начальные условия для уравнения из таблицы 2 - 1
{
    EQUATIONS_CNT = 2;
    y_0 = malloc(EQUATIONS_CNT * sizeof(double));
    x_0 = 1.0;
    y_0[0] = 1.0;
    y_0[1] = 1.0;
```

```
}

double
f2_2_1(double x, double *y) // первое уравнение системы из таблицы 2 - 2
{
    return (x * y[0] + y[1]);
}

double
f2_2_2(double x, double *y) // второе уравнение системы из таблицы 2 - 2
{
    return (y[0] - y[1]);
}

void
init_f2_2(void) // начальные условия для уравнения из таблицы 2 - 2
{
    EQUATIONS_CNT = 2;
    y_0 = malloc(EQUATIONS_CNT * sizeof(double));
    x_0 = 0.0;
    y_0[0] = 0.0;
    y_0[1] = 1.0;
}

double
f2_14_1(double x, double *y) // первое уравнение системы из таблицы 2 - 14
{
    return (exp(-( pow(y[0], 2) + pow(y[1], 2) )) + 2 * x);
}

double
f2_14_2(double x, double *y) // второе уравнение системы из таблицы 2 - 14
{
    return (2 * pow(y[0], 2) + y[1]);
}

void
init_f2_14(void) // начальные условия для уравнения из таблицы 2 - 14
{
    EQUATIONS_CNT = 2;
    y_0 = malloc(EQUATIONS_CNT * sizeof(double));
    x_0 = 0.0;
    y_0[0] = 0.5;
    y_0[1] = 1.0;
}

void
init_test(int test_num) // выбор номера теста и соответствующих начальных
условий
{
    switch (test_num) {
        case 1:
            init_f1_1();
            function[0] = f1_1;
            solution = sol_f1_1;
            break;
        case 2:
            init_f1_2();
            function[0] = f1_2;
            solution = sol_f1_2;
            break;
        case 3:
            init_f1_4();
            function[0] = f1_4;
            solution = sol_f1_4;
            break;
```

```
        case 4:
            init_f2_1();
            function[0] = f2_1_1;
            function[1] = f2_1_2;
            solution = NULL;
            break;
        case 5:
            init_f2_2();
            function[0] = f2_2_1;
            function[1] = f2_2_2;
            solution = NULL;
            break;
        case 6:
            init_f2_14();
            function[0] = f2_14_1;
            function[1] = f2_14_2;
            solution = NULL;
            break;
        default:
            break;
    }
}

void
print_vector(double x, double *y) // печать вектора (x, *y)
{
    printf("(%.10g", x);
    for (int i = 0; i < EQUATIONS_CNT; ++i) {
        printf("; %.10g", y[i]);
    }
    printf(")\n");
}

void
RungeKutta_2(double x_0, double *y_0) // метод Рунге-Кутта второго порядка
точности
{
    double *k = malloc(EQUATIONS_CNT * sizeof(double));
    double *y = malloc(EQUATIONS_CNT * sizeof(double));
    print_vector(x_0, y_0);

    for (int i = 0; i < ITER_CNT; ++i) {
        for (int j = 0; j < EQUATIONS_CNT; ++j) {
            k[j] = y_0[j] + function[j](x_0, y_0) * step;
        }

        for (int j = 0; j < EQUATIONS_CNT; ++j) {
            y[j] = y_0[j] + (function[j](x_0, y_0) + function[j](x_0 + step,
k)) * step / 2;
        }

        x_0 += step;

        print_vector(x_0, y);

        for (int j = 0; j < EQUATIONS_CNT; ++j) {
            y_0[j] = y[j];
        }
    }

    free(y);
    free(k);
}
```

```
void
RungeKutta_4 (double x_0, double *y_0) // метод Рунге-Кутта четвертого
порядка точности
{
    double *y =  malloc(EQUATIONS_CNT * sizeof(double));
    double *k1 = malloc(EQUATIONS_CNT * sizeof(double));
    double *k2 = malloc(EQUATIONS_CNT * sizeof(double));
    double *k3 = malloc(EQUATIONS_CNT * sizeof(double));
    double *k4 = malloc(EQUATIONS_CNT * sizeof(double));

    print_vector(x_0, y_0);

    for(int i = 0; i < ITER_CNT; ++i){
        for(int j = 0; j < EQUATIONS_CNT; ++j){
            k1[j] = function[j](x_0, y_0);
            y[j] = y_0[j] + step * k1[j] / 2;
        }

        for(int j = 0; j < EQUATIONS_CNT; ++j) {
            k2[j] = function[j](x_0 + step / 2, y);
            y[j] = y_0[j] + step * k2[j] / 2;
        }

        for(int j = 0; j < EQUATIONS_CNT; ++j) {
            k3[j] = function[j](x_0 + step / 2, y);
            y[j] = y_0[j] + step * k3[j];
        }

        for(int j = 0; j < EQUATIONS_CNT; ++j) {
            k4[j] = function[j](x_0 + step, y);
            y[j] = y_0[j] + (k1[j] + 2 * k2[j] + 2 * k3[j] + k4[j]) * step/6;
        }

        x_0 += step;

        print_vector(x_0, y);

        for(int j = 0; j < EQUATIONS_CNT; ++j) {
            y_0[j] = y[j];
        }
    }

    free(k1);
    free(k2);
    free(k3);
    free(k4);
    free(y);
}

void
exact_solution(double x_0, double *y_0)
{
    double *y = malloc(EQUATIONS_CNT * sizeof(double));
    print_vector(x_0, y_0);

    double x = x_0;
    for (int i = 0; i < ITER_CNT; ++i) {
        x += step;
        y[0] = solution(x);
        print_vector(x, y);
    }

    free(y);
}
```

```
int
main(void)
{
    int test_num;
    do {
        printf("Input test number:\n");
        printf("1: example 1-1\n");
        printf("2: example 1-2\n");
        printf("3: example 1-4\n");
        printf("4: example 2-1\n");
        printf("5: example 2-2\n");
        printf("6: example 2-14\n");
        scanf("%d", &test_num);
    } while (test_num < 1 && test_num > TESTS_CNT);

    printf("Input count of iterations:\n");
    scanf("%d", &ITER_CNT);

    printf("Input step:\n");
    scanf("%lf", &step);

    printf("\n");
    init_test(test_num);
    printf("Runge Kutta method of the second order of accuracy:\n");
    RungeKutta_2(x_0, y_0);
    printf("\n");

    init_test(test_num);
    printf("Runge Kutta method of the fourth order of accuracy:\n");
    RungeKutta_4(x_0, y_0);
    printf("\n");

    if (solution != NULL) {
        init_test(test_num);
        printf("Exact solution:\n");
        exact_solution(x_0, y_0);
        printf("\n");
    }

    return 0;
}
```

Тестирование

Решения были проверены с помощью http://www.wolframalpha.com

Тест 1

Пример 1- 1
Количество итераций – 8
Размер шага – 0.25

Точное решение:
$y(x) = -x - 4e^{-x} + 4$

Вывод:

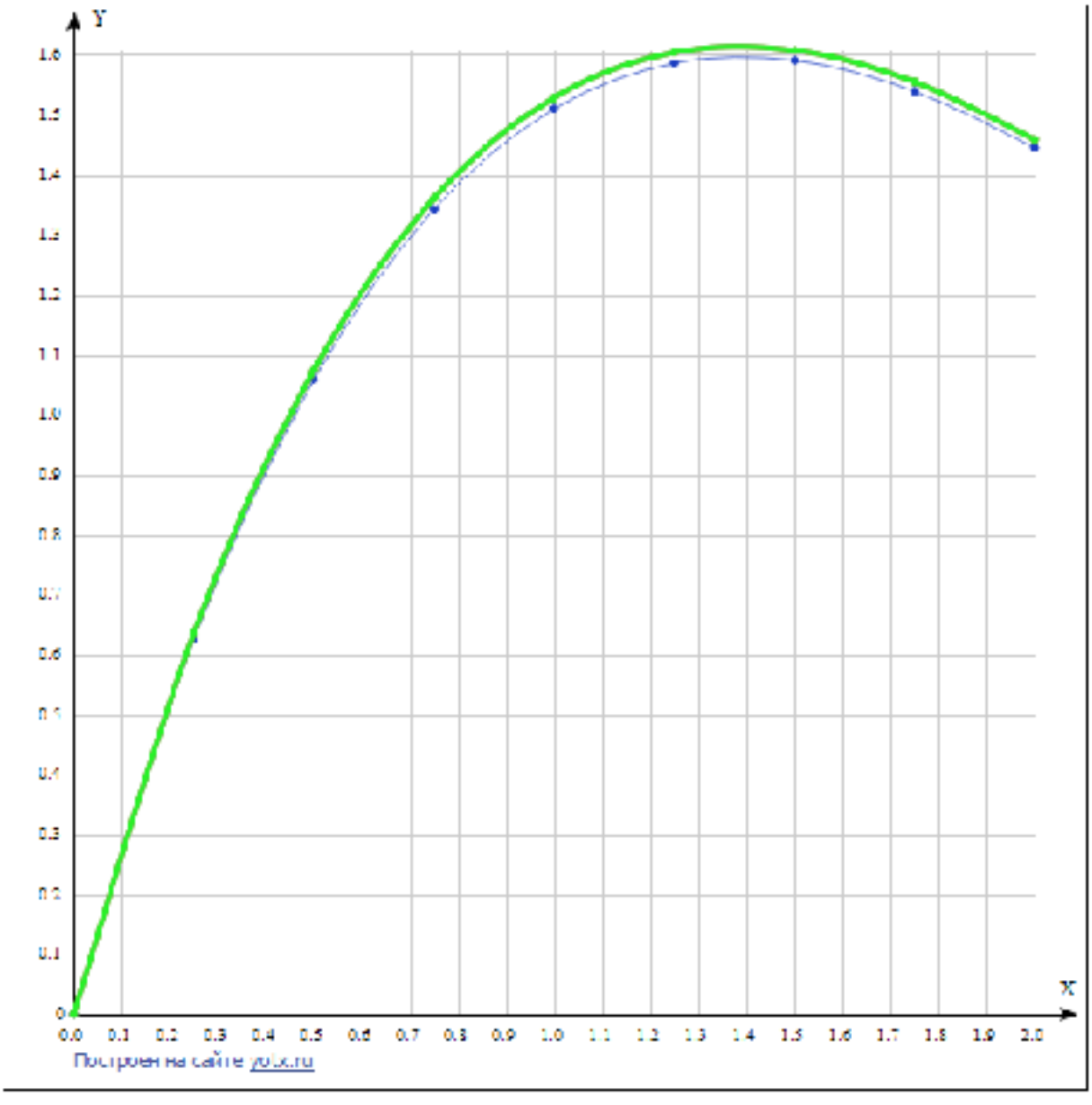
```
Runge Kutta method of the second order of accuracy:
(0; 0)
(0.25; 0.625)
(0.5; 1.05859375)
(0.75; 1.342651367)
(1; 1.509883881)
(1.25; 1.585846782)
(1.5; 1.590505298)
```

```
(1.75; 1.539457264)
(2; 1.444888488)


Runge Kutta method of the fourth order of accuracy:
(0; 0)
(0.25; 0.634765625)
(0.5; 1.073828697)
(0.75; 1.360476939)
(1; 1.528423202)
(1.25; 1.603923344)
(1.5; 1.607425651)
(1.75; 1.554855426)
(2; 1.458615432)

Exact solution:
(0; 0)
(0.25; 0.6347968677)
(0.5; 1.073877361)
(0.75; 1.360533789)
(1; 1.528482235)
(1.25; 1.603980813)
(1.5; 1.607479359)
(1.75; 1.554904226)
(2; 1.458658867)
```

Совмещая три графика (метод Рунге-Кутта 2 порядка - тонкой линией, метод Рунге-Кутта 4 порядка и точное решение – толстой линией), получаем:



Тест 2

Пример 1-2
Количество итераций – 8
Размер шага – 0.25

Точное решение:
$y(x) = 0.5(21e^{-x} + \sin(x) - \cos(x))$

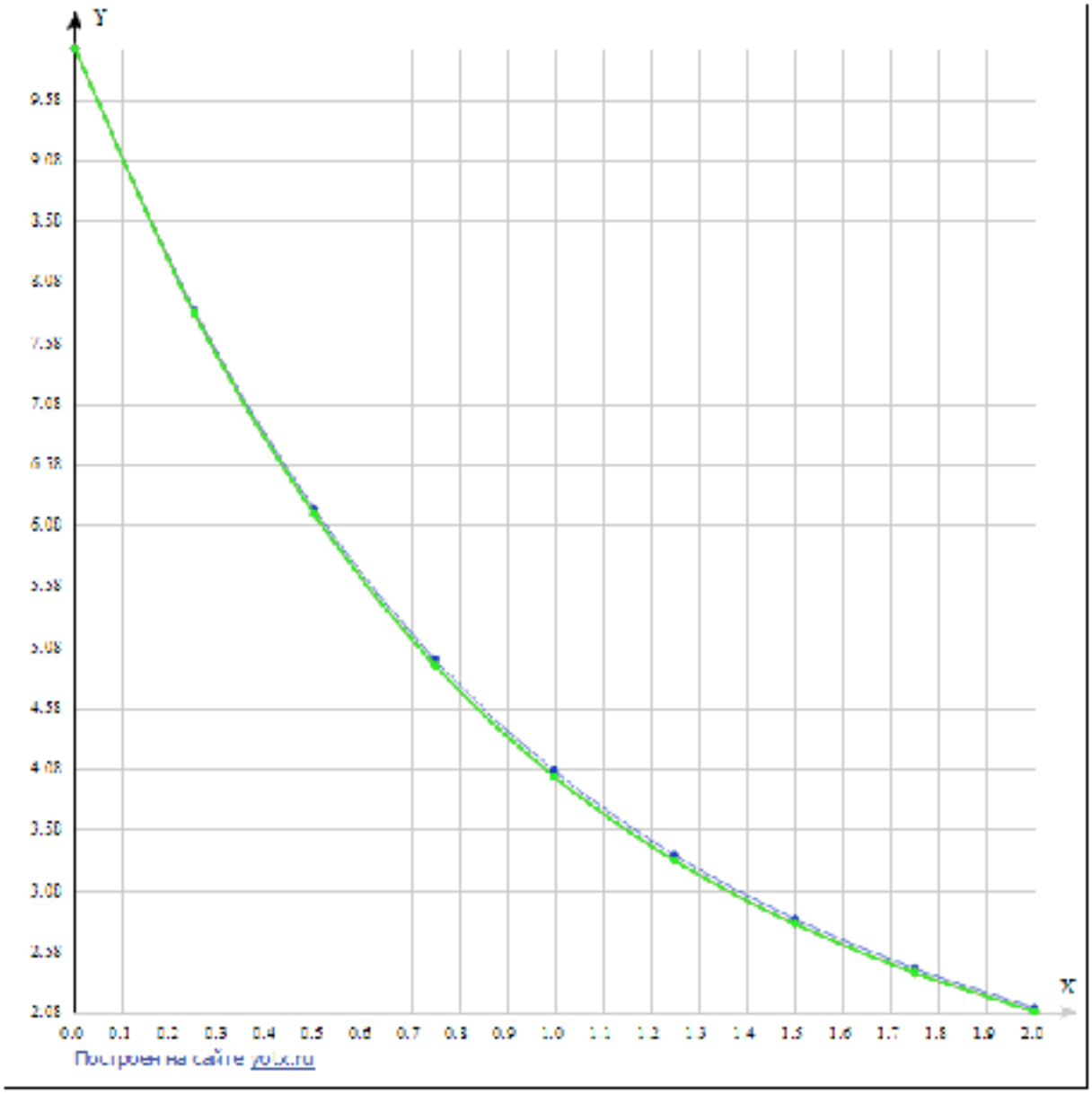
Вывод:

```
Runge Kutta method of the second order of accuracy:
(0; 10)
(0.25; 7.843425495)
(0.5; 6.210798481)
(0.75; 4.982337303)
(1; 4.061538525)
(1.25; 3.370587955)
(1.5; 2.846926021)
(1.75; 2.440674352)
(2; 2.112687699)

Runge Kutta method of the fourth order of accuracy:
(0; 10)
(0.25; 7.816738505)
(0.5; 6.169624095)
(0.75; 4.93497468)
(1; 4.013472632)
(1.25; 3.325278197)
(1.5; 2.806378605)
(1.75; 2.405858702)
(2; 2.083841272)

Exact solution:
(0; 10)
(0.25; 7.816653991)
(0.5; 6.169493415)
(0.75; 4.934823749)
(1; 4.013318472)
(1.25; 3.325131496)
(1.5; 2.806245574)
(1.75; 2.405742407)
(2; 2.083742606)
```

Совмещая три графика (метод Рунге-Кутта 2 порядка - тонкой линией, метод Рунге-Кутта 4 порядка и точное решение – толстой линией), получаем:



Тест 3

Пример 1-4
Количество итераций – 8
Размер шага – 0.5

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

Вывод:

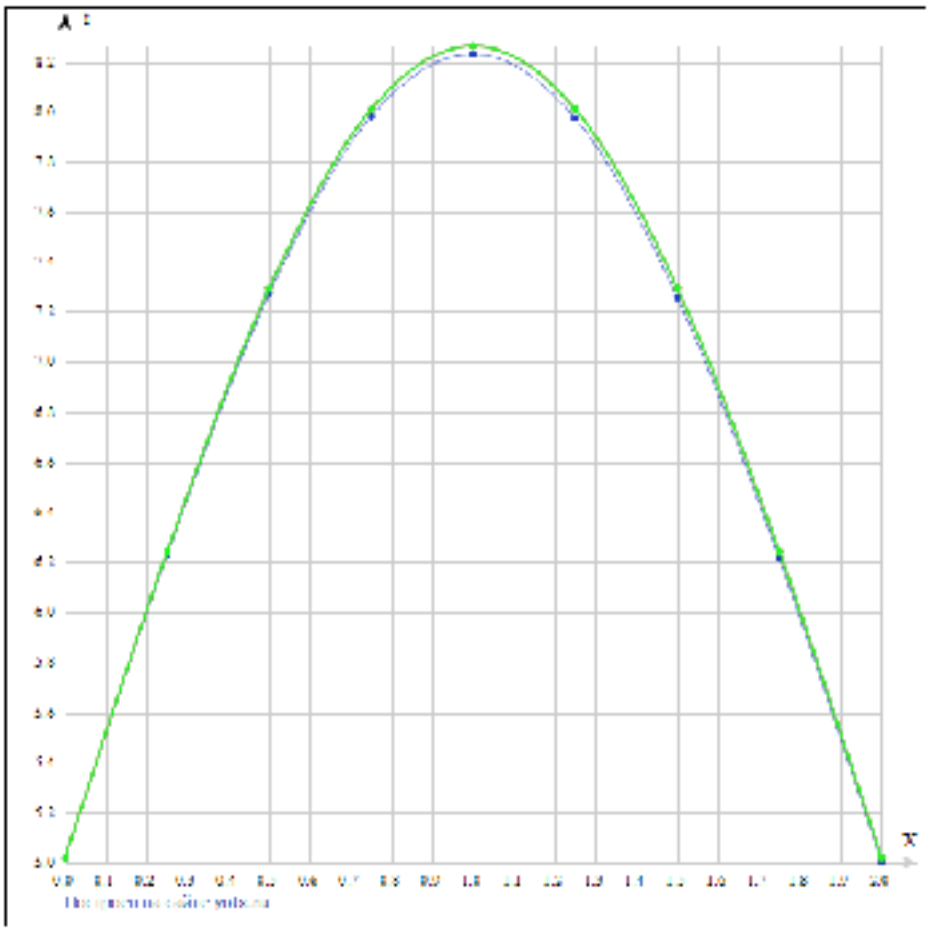
```
Runge Kutta method of the second order of accuracy:
(0; 5)
(0.25; 6.2109375)
(0.5; 7.254180908)
(0.75; 7.962597013)
(1; 8.211428169)
(1.25; 7.954821039)
(1.5; 7.240130086)
```

```
(1.75; 6.193705035)
(2; 4.98399702)

Runge Kutta method of the fourth order of accuracy:
(0; 5)
(0.25; 6.222569148)
(0.5; 7.274909234)
(0.75; 7.989921661)
(1; 8.24354868)
(1.25; 7.989921026)
(1.5; 7.274903498)
(1.75; 6.222552989)
(2; 4.999974732)

Exact solution:
(0; 5)
(0.25; 6.222600539)
(0.5; 7.274957073)
(0.75; 7.98997725)
(1; 8.243606354)
(1.25; 7.98997725)
(1.5; 7.274957073)
(1.75; 6.222600539)
(2; 5)
```

Совмещая три графика (метод Рунге-Кутта 2 порядка - тонкой линией, метод Рунге-Кутта 4 порядка и точное решение – толстой линией), получаем:



Тесты для системы дифференциальных уравнений

Тест 4

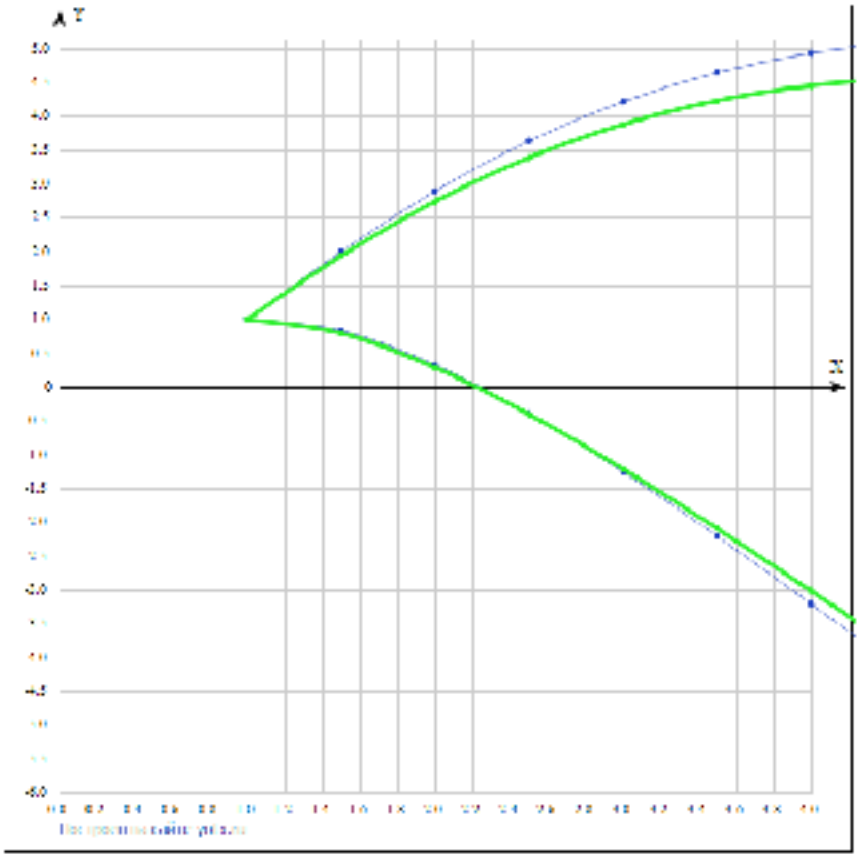
Пример 2-1
Количество итераций – 8
Размер шага – 0.5

Вывод:

```
Runge Kutta method of the second order of accuracy:
(1; 1; 1)
(1.5; 0.8333333333; 2)
(2; 0.3263888889; 2.895833333)
(2.5; -0.3965277778; 3.637152778)
(3; -1.257274306; 4.218049769)
(3.5; -2.205075645; 4.646329916)
(4; -3.205644781; 4.933907262)
(4.5; -4.235090609; 5.093415406)
(5; -5.276359749; 5.136959794)

Runge Kutta method of the fourth order of accuracy:
(1; 1; 1)
(1.5; 0.7931555556; 1.935975309)
(2; 0.2858183504; 2.740696942)
(2.5; -0.4067761692; 3.388320394)
(3; -1.215073146; 3.881830111)
(3.5; -2.094466163; 4.232415282)
(4; -3.015048622; 4.453152221)
(4.5; -3.956105858; 4.556856121)
(5; -4.902942777; 4.555361228)
```

Совмещая два графика (метод Рунге-Кутта 2 порядка - тонкой линией, метод Рунге-Кутта 4 порядка – толстой линией), получаем:



Тест 5

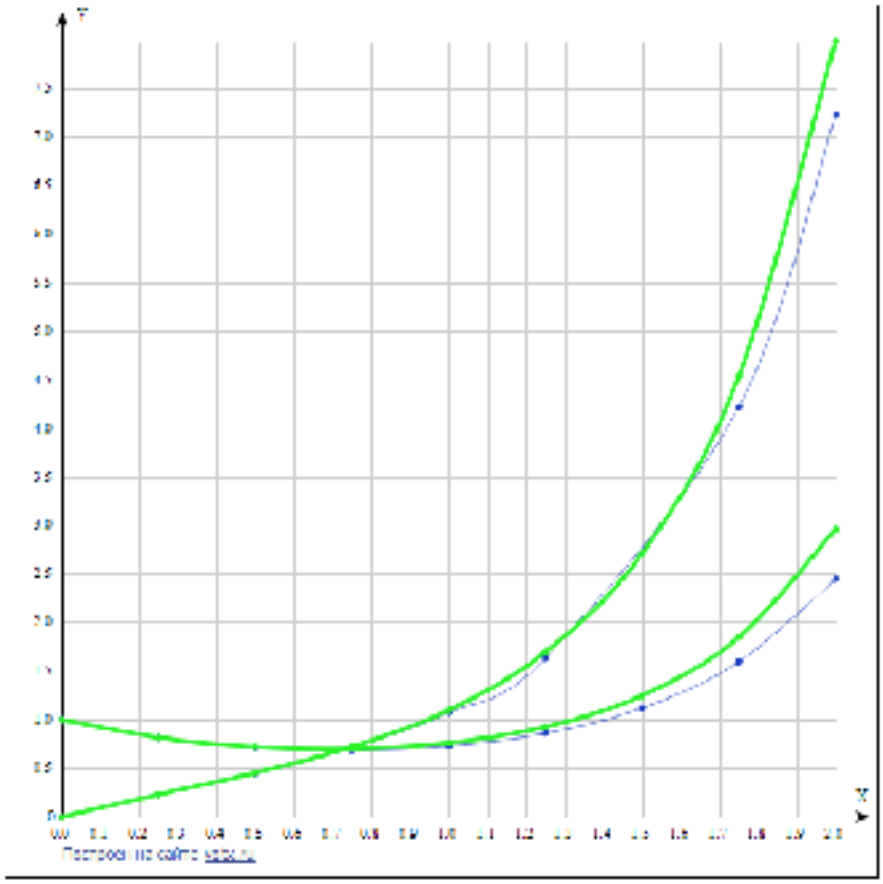
Пример 2-2
Количество итераций – 8
Размер шага – 0.25

Вывод:

```
Runge Kutta method of the second order of accuracy:
(0; 0; 1)
(0.25; 0.2265625; 0.8125)
(0.5; 0.4461975098; 0.7114868164)
(0.75; 0.7074016333; 0.6826605797)
(1; 1.071496912; 0.725985554)
(1.25; 1.635363196; 0.8577374907)
(1.5; 2.572282978; 1.118528785)
(1.75; 4.214512084; 1.592067304)
(2; 7.230499314; 2.445960332)

Runge Kutta method of the fourth order of accuracy:
(0; 0; 1)
(0.25; 0.2294514974; 0.8133744134)
(0.5; 0.4522124521; 0.7177626019)
(0.75; 0.7199038637; 0.6993220393)
(1; 1.098872998; 0.7608936016)
(1.25; 1.696670586; 0.9255397233)
(1.5; 2.711296711; 1.248557994)
(1.75; 4.534636705; 1.845820848)
(2; 7.983913929; 2.957815485)
```

Совмещая два графика (метод Рунге-Кутта 2 порядка - тонкой линией, метод Рунге-Кутта 4 порядка – толстой линией), получаем:



Тест 6

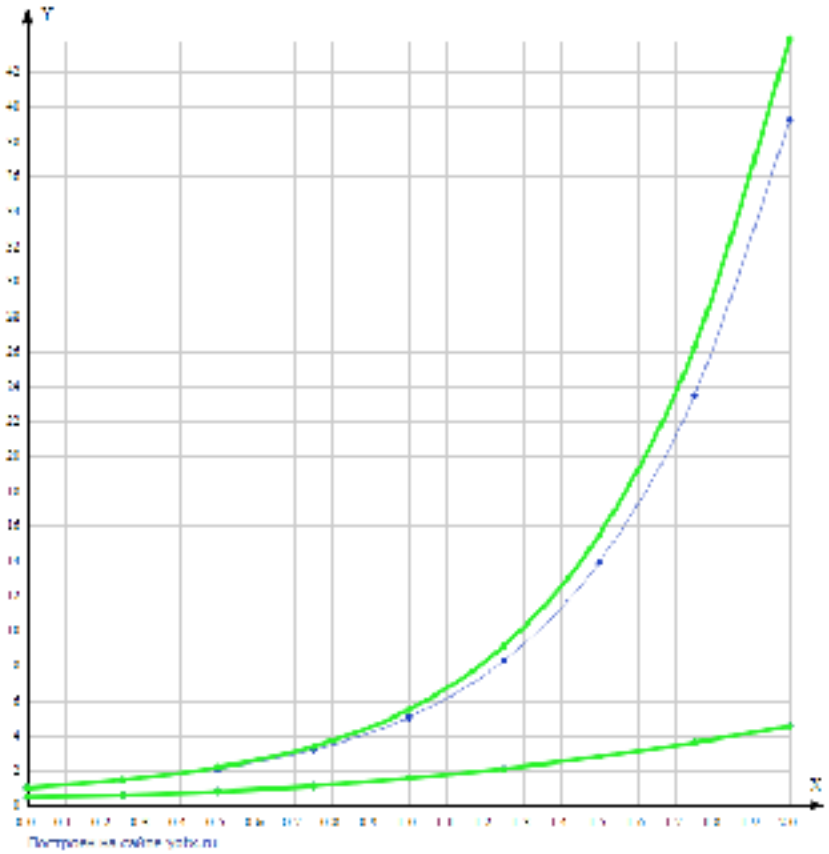
Пример 2-14
Количество итераций – 8
Размер шага – 0.25

Вывод:

```
Runge Kutta method of the second order of accuracy:
(0; 0.5; 1)
(0.25; 0.6119248338; 1.441064128)
(0.5; 0.8115475528; 2.107200416)
(0.75; 1.124816684; 3.188197249)
(1; 1.562318042; 5.042621213)
(1.25; 2.124818042; 8.286909127)
(1.5; 2.812318042; 13.9188683)
(1.75; 3.624818042; 23.47768145)
(2; 4.562318042; 39.24890304)

Runge Kutta method of the fourth order of accuracy:
(0; 0.5; 1)
(0.25; 0.606524177; 1.468111009)
(0.5; 0.8019422091; 2.192233429)
(0.75; 1.11467421; 3.394734597)
(1; 1.55217433; 5.482476615)
(1.25; 2.11467433; 9.147764175)
(1.5; 2.80217433; 15.50039606)
(1.75; 3.61467433; 26.2393139)
(2; 4.55217433; 43.8731817)
```

Совмещая два графика (метод Рунге-Кутта 2 порядка - тонкой линией, метод Рунге-Кутта 4 порядка – толстой линией), получаем:



Выводы

В ходе практической работы я изучил метод Рунге-Кутта второго и четвертого порядка для решения задачи Коши дифференциального уравнения первого порядка и системы дифференциальных уравнений первого порядка. Данные методы были реализованы в виде программы на языке С. Они просты в реализации и обладают относительно высокой точностью. Также были найдены численные решения задачи Коши и построены графики ее решения.

В методе Рунге-Кутта второго порядка функция f(x, y) вычисляется дважды на каждом шаге, а в методе Рунге-Кутта четвертого порядка – четыре раза. Однако эта сложность алгоритма компенсируется более высокой точностью, в чем я убедился на рассмотренных выше примерах.