МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 2.

Численные методы решения дифференциальных уравнений

ОТЧЕТ

о выполненном задании

студента 202 учебной группы факультета ВМК МГУ

Савиных Юрия Сергеевича

гор. Москва

2020 г.

Содержание

Цель работы **2**

Постановка задачи **2**

Задачи практической работы **2**

Алгоритм **3**
Метод Рунге-Кутта второго порядка точности 3
Метод Рунге-Кутта четвёртого порядка точности 3

Описание программы **4**

Код программы **5**

Тестирование программы **8**
Тест №1 (Таблица 1. Вариант 6) 8
Тест №2 (дополнительный) 8
Тест №3 (Таблица 2. Вариант 5) 9
Тест №4 (дополнительный) 12

Выводы **13**

Цель работы

Освоить методы Рунге-Кутта второго и четвертого порядка точности, применяемые для численного решения задачи Коши для дифференциального уравнения (или системы дифференциальных уравнений) первого порядка.

Постановка задачи

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

$$y' = f(x, y),\ x_0 < x \tag{1}$$

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

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

Предполагается, что функция $f(x, y)$, такова, что гарантирует существование и единственность решения заданной задачи Коши.

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):

$$\begin{cases} y_1' = f_1(x, y_1, y_2) \\ y_2' = f_2(x, y_1, y_2) \end{cases} \quad x > x_0 \tag{3}$$

Начальные условия задаются в точке $x = x_0$:

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

Предполагается, что правые части уравнений из (5) заданы так, что это гарантирует существование и единственность решения задачи Коши (5)-(4), но уже для системы обыкновенных дифференциальных уравнений первого порядка в форме, разрешенной относительно производных неизвестных функций.

Задачи практической работы

1. Решить заданную задачу Коши методами Рунге-Кутта второго и четвертого порядка точности, аппроксимировав дифференциальную задачу соответствующей разностной схемой (на равномерной сетке); полученное конечно-разностное уравнение (или уравнения в случае системы) просчитать численно;

2. Найти численное решение задачи и построить его график;

3. Найденное численное решение сравнить с точным решением дифференциального уравнения.

Алгоритм

Метод Рунге-Кутта второго порядка точности

Сопоставим нашей задаче разностную задачу:

$$\frac{y_{i+1} - y_i}{h} = f(x_i, y_i)$$

Положим:

$$f(x_i, y_i) = \frac{h}{2}(\frac{\partial f}{\partial x}(x_i, y_i) + \frac{\partial f}{\partial y}(x_i, y_i)) = \beta f(x_i, y_i) + \alpha f(x_i + \gamma h, y_i + \delta h) + O(h^2)$$

После разложения функции $f(x_i + \gamma h, y_i + \delta h)$ по степеням h до степени h^2, возникают следующие зависимости:

$$\begin{cases} \beta = 1 - \alpha \\ \gamma = \frac{1}{2\alpha} \\ \delta = \frac{1}{2\alpha} f(x_i, y_i) \end{cases} \tag{5}$$

Наиболее удобные разностные схемы соответсвуют $\alpha = \frac{1}{2}$ и $\alpha = 1$. Рассмотрим при $\alpha = \frac{1}{2}$. Выразим y_{i+1} и получим рекурентную формулу:

$$y_{i+1} = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_i + h, y_i + hf(x_i, y_i)))$$

В случае решения задачи Коши для системы дифференциальных уравнений, эта формула справедлива для любой функции y_k.

Метод Рунге-Кутта четвёртого порядка точности

Иногда второго порядка точности бывает недостаточно, поэтому часто используется схема Рунге-Кутта четвертого порядка точности следующего вида:

$$\frac{y_{i+1} - y_i}{h} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

где

$$k_1 = f(x_i, y_i),$$

$$k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1),$$

$$k_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2),$$

$$k_4 = f(x_i + h, y_i + hk_3).$$

В случае решения задачи Коши для системы дифференциальных уравнений, эта формула справедлива для любой функции y_k.

Описание программы

Программа разделена на 3 модуля:

1. main.py - модуль который необходимо запустить для вычисления результата.

2. functions.py - модуль, содержащий функции f_i для тестов. Названия у функций следующего вида: fij, где i - номер теста, j - номер функции в системе уравнений.

3. calc.py - модуль, содержащий реализацию метода Рунге-Кутта с точностью $o(h^2)$ и $o(h^4)$. Функция `def runge(f, x0, y0, h, steps_count, mode)` реализует метод Рунге-Кутта порядка `mode` (`mode` равен 2 или 4). `f` - список функций правой части, `x0` - коородината x_0 начального условия, `y0` - массив значений функций в x_0, `h` - длина шага, `steps_count` - число шагов

При запуске программы необходимо ввести размер шага, число шагов, номер теста и точность метода Рунге-Кутта. Результат работы программы график, построенный по точкам (x, y) $((x, y_1, y_2))$.

Код программы

Код программы можно найти на https://github.com/YuriSavinykh/Numerical-Methods/tree/main/Task2/1

```python
from calc import runge
from functions import *
import numpy as np
import matplotlib.pyplot as plt


def print_dots(x, y, n):
    for i in range(x.size):
        tmp = ""
        for j in range(n):
            tmp += ", "
            tmp += '{:3.7f}'.format(y[j][i])
        print("(" + '{:3.7f}'.format(x[i]) + tmp + ")", end=" ")
    print()


if __name__ == '__main__':
    step = float(input("Input step size: "))

    if step <= 0:
        raise Exception("Incorrect data")

    n = int(input("Input steps count: "))

    if n < 0:
        raise Exception("Incorrect data")

    test_number = int(input("Input test number 1 (Table1-6), 2 (additional test 1), "
                            "3 (Table 2-5),  4(additional test 2): "))

    if test_number < 1 and test_number > 4:
        raise Exception("Incorrect data")

    runge_i = int(input("Input order of accuracy: 2 for Runge-Kutta (O(2)), 4 for Runge-Kutta(O(4)): "))

    if runge_i != 2 and runge_i != 4:
        raise Exception("Incorrect data")

    functions = []

    if test_number == 1:
        x0 = 0
        y0 = np.array([1])
        functions.append(f11)
    elif test_number == 2:
        x0 = 0
        y0 = np.array([0])
        functions.append(f21)
```

```
    elif test_number == 3:
        x0 = 0
        y0 = np.array([1, 0.05])
        functions.append(f31)
        functions.append(f32)
    elif test_number == 4:
        x0 = 0
        y0 = np.array([1, 2])
        functions.append(f41)
        functions.append(f42)
    else:
        raise Exception("Incorrect data")

    x, y = runge(functions, x0, y0, step, n, runge_i)

    print_dots(x, y, y.size // (n + 1))

    if test_number == 1 or test_number == 2:
        plt.plot(x, y[0])
        plt.show()
    elif test_number == 3 or test_number == 4:
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot(x, y[0], y[1], label='solution')
        ax.set_xlabel('x Label')
        ax.set_ylabel('u Label')
        ax.set_zlabel('v Label')
        ax.legend()
        plt.show()
```

Листинг 1: main.py

```
import numpy as np


def runge(f, x0, y0, h, steps_count, mode):
    if mode != 2 and mode != 4:
        raise Exception("Incorrect date")

    equations_count = y0.size
    iterations_count = steps_count + 1

    x = np.empty(iterations_count)
    y = np.empty((equations_count, iterations_count))

    for i in range(equations_count):
        y[i][0] = y0[i]

    for i in range(iterations_count):
        x[i] = x0 + i * h

    for i in range(1, iterations_count):
        for j in range(equations_count):
            if mode == 2:
                tmp = f[j](x[i - 1], y[:, i - 1])
                y[j][i] = y[j][i - 1] + h / 2 * (tmp + f[j](x[i -
```

```
1] + h, y[:, i - 1] + h * tmp))
            else:
                k1 = f[j](x[i - 1], y[:, i - 1])
                k2 = f[j](x[i - 1] + h / 2, y[:, i - 1] + h / 2 * k1)
                k3 = f[j](x[i - 1] + h / 2, y[:, i - 1] + h / 2 * k2)
                k4 = f[j](x[i - 1] + h / 2, y[:, i - 1] + h * k3)
                y[j][i] = y[j][i - 1] + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

    return x, y
```

Листинг 2: calc.py

```
from math import cos, exp


def f11(x, y):
    return (x - x ** 2) * y


def f21(x, y):
    return 2 * x - 12 - 0.25 * exp(0.25 * x)


def f31(x, y):
    return cos(y[0] + 1.1 * y[1]) + 2.1


def f32(x, y):
    return 1.1 / (x + 2.1 * y[0] ** 2) + x + 1


def f41(x, y):
    return y[1] * 2 - 3 * y[0]


def f42(x, y):
    return y[1] - 2 * y[0]
```

Листинг 3: functions.py

Тестирование программы

По результатам работы программы были построенные графики. Красные точки - результат метода Рунге-Кутта второго порядка точности, зелёные - четвёртого. Чёрным цветом показан график аналитического решения. В тесте 3 аналитически решение найти не удалось, поэтому для построения графика был использован интернет ресурс https://www.mathstools.com/section/main/runge_kutta_calculator#.X-RVtiZR1H5, который решил задачу Коши методом Рунге-Кутта с точностью $o(h^2)$ и по результату работы этого ресурса был построен контрольный график.

Тест №1 (Таблица 1. Вариант 6)

$$\begin{cases} y' = (x - x^2)y \\ y(0) = 1 \end{cases}$$

Аналитическое решение:

$$y(x) = e^{-\frac{1}{6}x^2(-3+2x)}$$

Шаг $=$ 0.2. Число шагов $=$ 20.

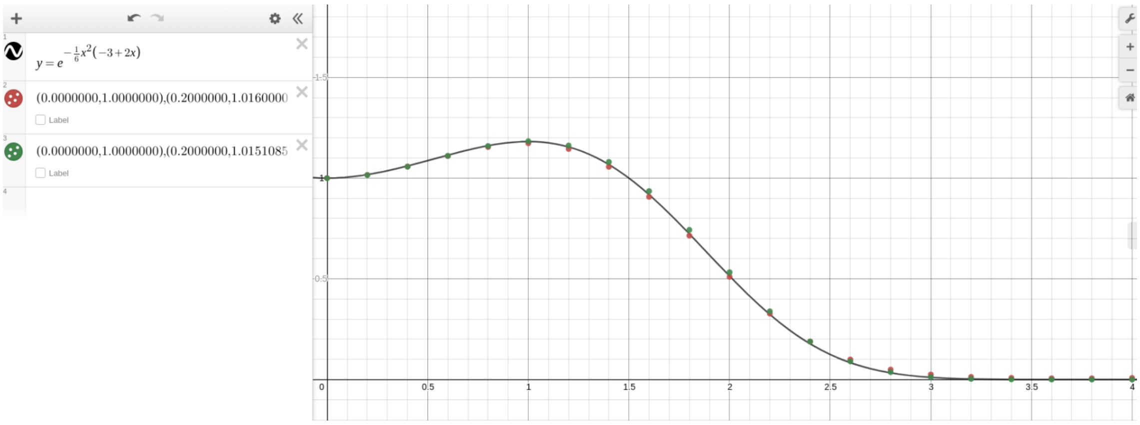


Рис. 1

Тест №2 (дополнительный)

$$\begin{cases} y' = 2x - 12 - \frac{1}{4}e^{\frac{x}{4}} \\ y(0) = 0 \end{cases}$$

Аналитическое решение:

$$y(x) = x^2 - 12x - e^{\frac{x}{4}}x$$

Шаг $=$ 1. Число шагов $=$ 21.

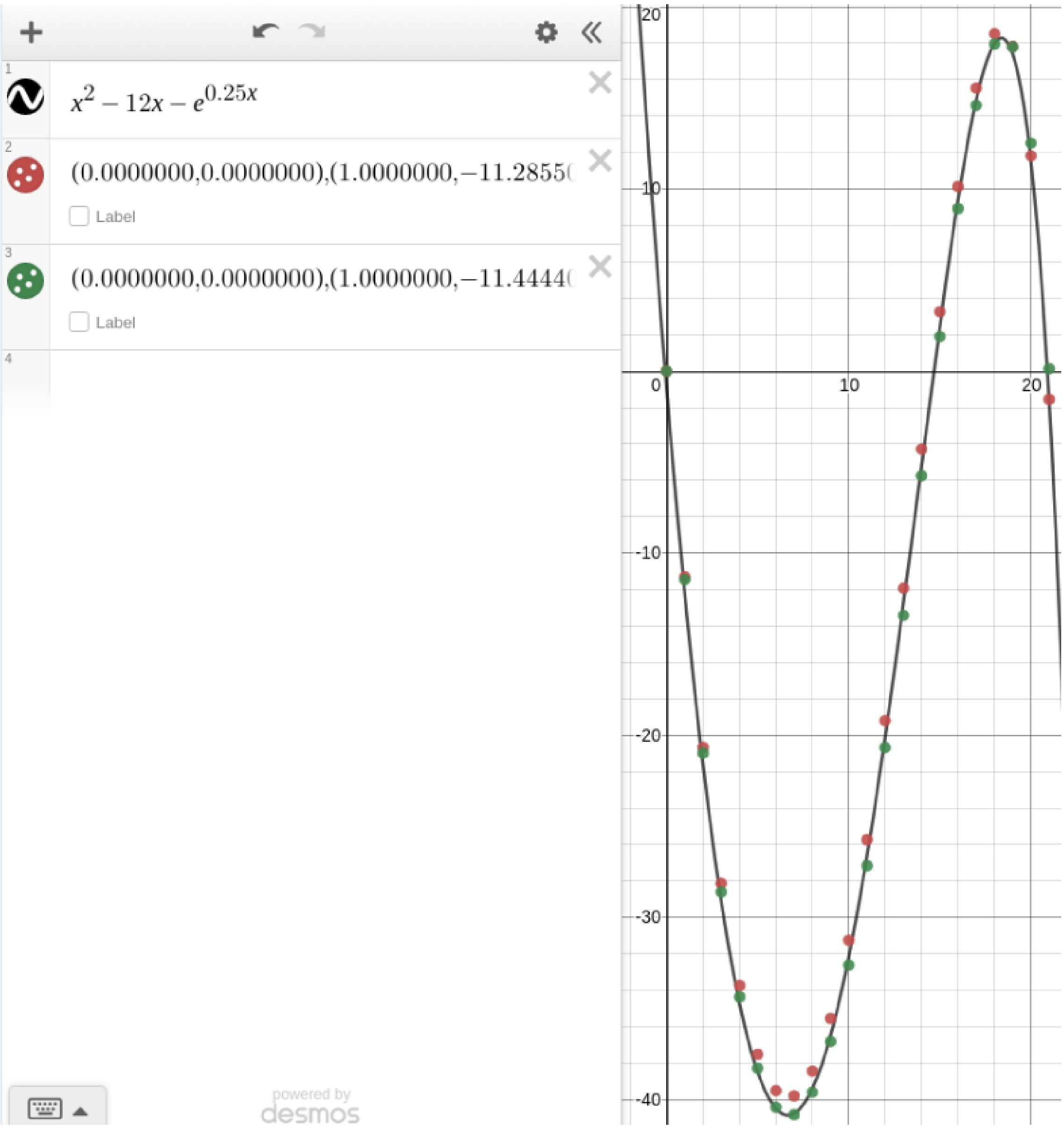


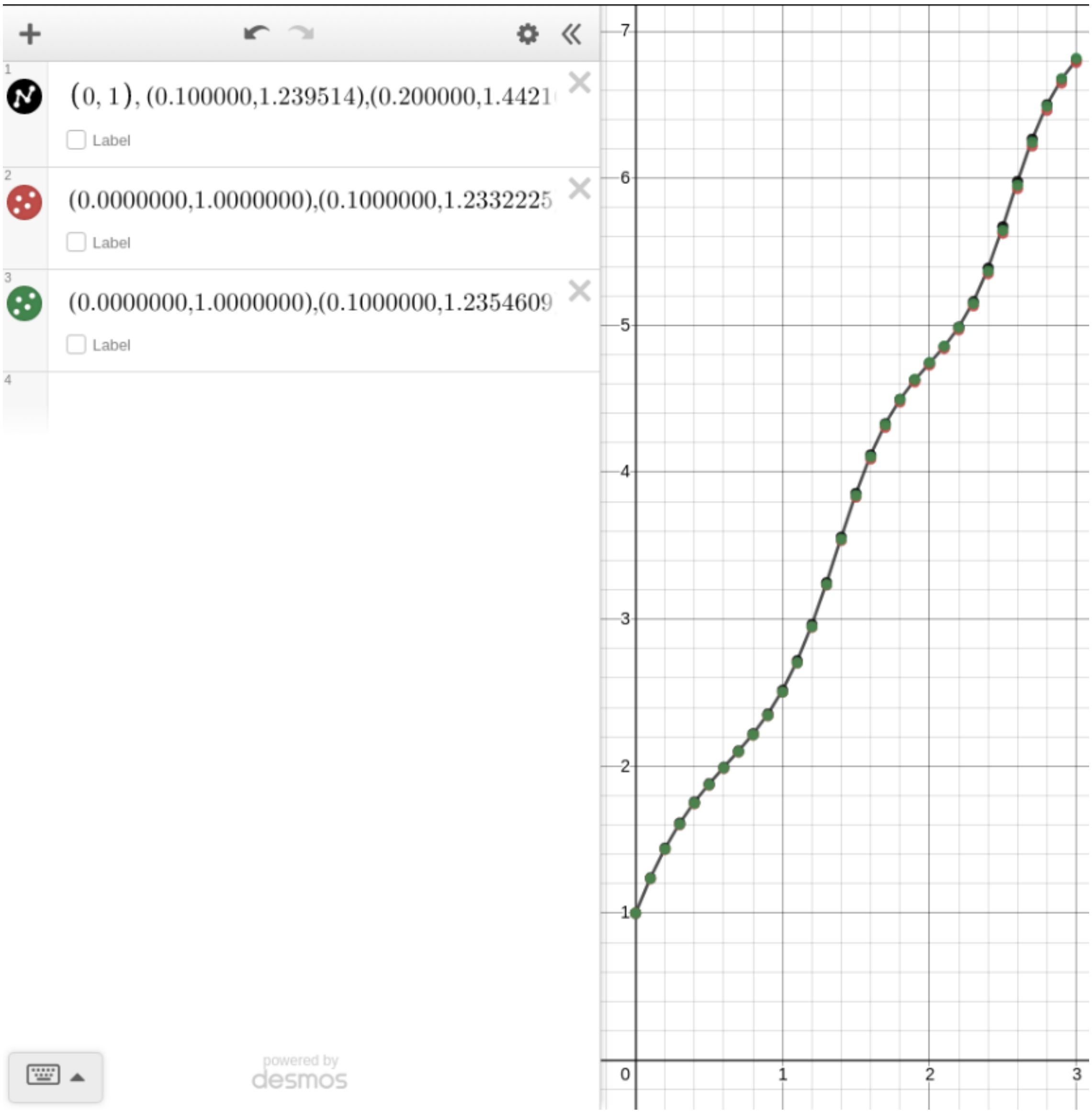
Рис. 2

Тест №3 (Таблица 2. Вариант 5)

$$\begin{cases} y_1' = cos(y_1 + 1.1y_2) + 2.1 \\ y_2' = \frac{1.1}{x+2.1y_1^2} + x + 1 \\ y_1(0) = 1 \\ y_2(0) = 0.05 \end{cases}$$

Для проверки результата была использована платформа https://www.mathstools.com/section/main/runge_kutta_calculator#.X-RVtiZR1H5, решающая задачу Коши методом Рунге-Кутта второго порядка точности.

Шаг = 0.1. Число шагов = 30.



(0, 1), (0.100000,1.239514),(0.200000,1.4421
Label
(0.0000000,1.0000000),(0.1000000,1.2332225
Label
(0.0000000,1.0000000),(0.1000000,1.2354609
Label
powered by
desmos

Рис. 3 а

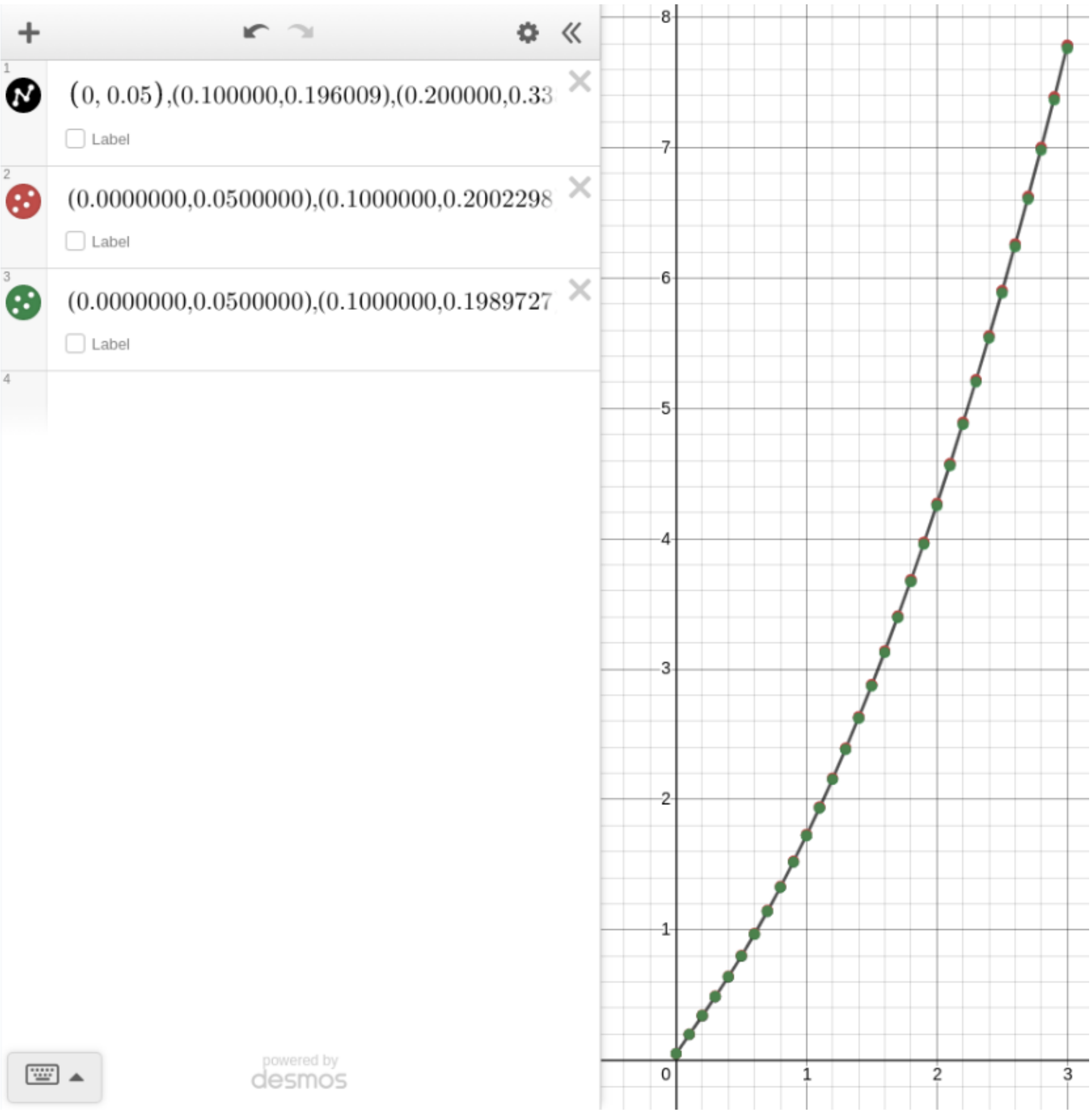

(0, 0.05),(0.100000,0.196009),(0.200000,0.33
Label
(0.0000000,0.0500000),(0.1000000,0.2002298
Label
(0.0000000,0.0500000),(0.1000000,0.1989727
Label
powered by
desmos
8
7
6
5
4
3
2
1
0
1
2
3

Рис. 3 б

Тест №4 (дополнительный)

$$\begin{cases} y_1' = 2y_2 - 3y_1 \\ y_2' = y_2 - 2y_1 \\ y_1(0) = 1 \\ y_2(0) = 2 \end{cases}$$

Аналитическое решение:

$$\begin{cases} y_1 = (1 + 2x)e^{-x} \\ y_2 = (2 + 2x)e^{-x} \end{cases}$$

Шаг $= 0.2$. Число шагов $= 20$.

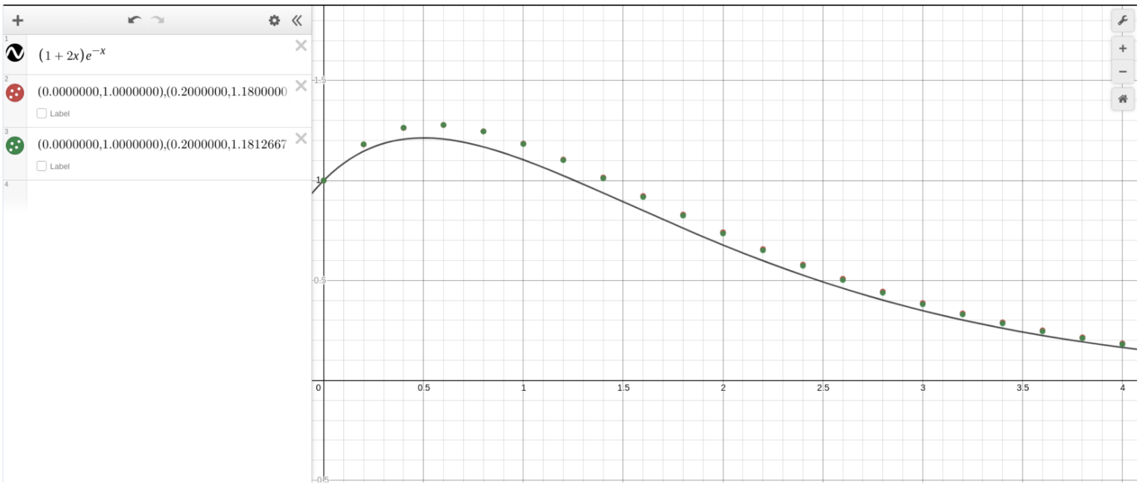


Рис. 4 а

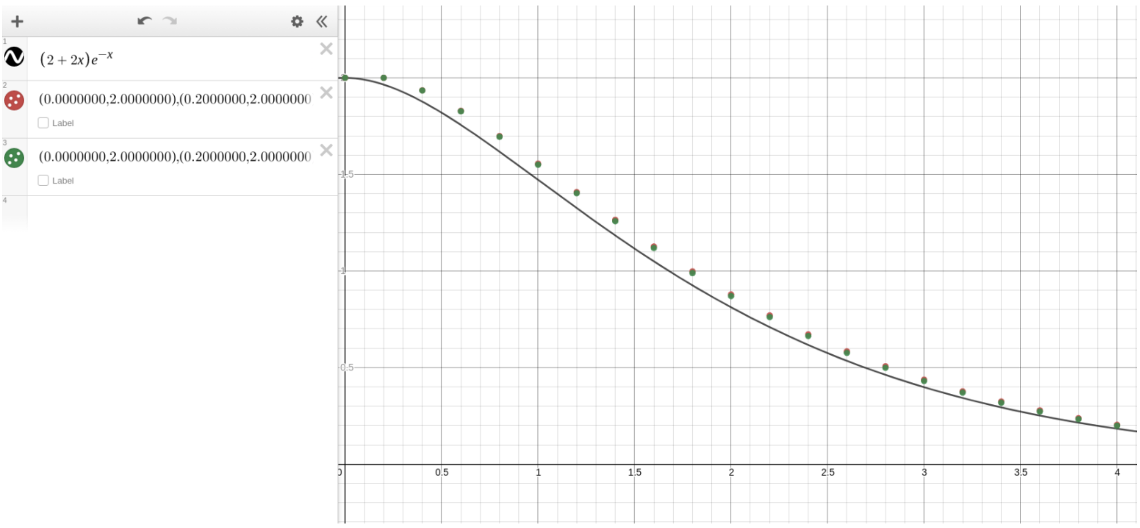


Рис. 4 б

Выводы

В ходе работы были освоены методы Рунге-Кутта второго и четвертого порядка точности. Найдены численные решения задач Коши и построены их соотвествующие графики. Полученные решения сопоставлены с точными решениями соответствующих задач.

Содержание

Цель работы **2**

Постановка задачи **2**

Задачи практической работы **2**

Алгоритм **3**

Задание сетки, аппроксимирование, получение уравнений 3

Решение полученной системы методом прогонки 4

Описание программы **5**

Код программы **6**

Тестирование программы **9**

Тест №1 (Вариант 4) 9

Тест №2 (Дополнительный тест) 10

Тест №3 (Дополнительный тест) 11

Тест №4 (Дополнительный тест) 12

Выводы **13**

Цель работы

Освоить метод прогонки решения краевой задачи для дифференциального уравнения второго порядка.

Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида

$$y'' + p(x)y + q(x)y = -f(x),\ a < x < b$$

с дополнительными условиями в граничных точках

$$\begin{cases} \sigma_1 y(a) + \gamma_1 y'(a) = \delta_1 \\ \sigma_2 y(b) + \gamma_2 y'(b) = \delta_2 \end{cases}$$

Задачи практической работы

1. Решить краевую задачу методом конечных разностей, аппроксимировав ее разностной схемой второго порядка точности (на равномерной сетке); полученную систему конечно-разностных уравнений решить методом прогонки;

2. Найти разностное решение задачи и построить его график;

3. Найденное разностное решение сравнить с точным решением дифференциального уравнения

Алгоритм

Задание сетки, аппроксимирование, получение уравнений

Сначала введём обозначения: $y_i = y(x_i), p_i = p(x_i), q_i = q(x_i), f_i = f(x_i), y_0 = y(a),\ y_n = y(b)$.

Зададим равномерную сетку на $[a, b]$ из $(n+1)$ узла, с узлами $x_i = a + ih, i = 0, 1, ..., n$ и шагом $h = \frac{b-a}{n}$

Проаппроксимируем производные уравнения разностными функциями второго порядка точности:

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + p_i\frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i = -f_i,\ i = 1, ..., n-1$$

Сугруппируем слагаемые:

$$\begin{cases} A_i = 2 - hp_i \\ B_i = -4 + 2h^2 q_i \\ C_i = 2 + hp_i \\ D_i = -2h^2 f_i \end{cases}$$

Получим систему из $(n-1)$ уравнений с $(n+1)$ неизвестными:

$$A_i y_{i-1} + B_i y_i + C_i y_{i+1} = D_i,\ i = 1, ..., n-1$$

Ещё два уравнения получим аппроксимировав производные в граничных условиях:

$$\begin{cases} \sigma_1 y_0 + \gamma_1 \frac{y_1 - y_0}{h} = \delta_1 \\ \sigma_2 y_n + \gamma_2 \frac{y_n - y_{n-1}}{h} = \delta_2 \end{cases}$$

Сугруппируем слагаемые:

$$\begin{cases} B_0 = h\sigma_1 - \gamma_1 \\ C_0 = \gamma_1 \\ D_0 = h\delta_1 \end{cases}$$

$$\begin{cases} A_n = -\gamma_2 \\ B_n = h\sigma_2 + \gamma_2 \\ D_n = h\delta_2 \end{cases}$$

Решение полученной системы методом прогонки

Итак получили систему $(n+1)$ уравнений, в матричной форме имеющая следующий вид:

$$A = \begin{pmatrix} B_0 & C_0 & & & & \\ A_1 & B_1 & C_1 & & 0 & \\ & A_2 & B_2 & C_2 & & \\ & & ... & ... & ... & \\ & 0 & & A_{n-1} & B_{n-1} & C_{n-1} \\ & & & & A_n & B_n \end{pmatrix} f = \begin{pmatrix} D_0 \\ D_1 \\ D_2 \\ ... \\ D_{n-1} \\ D_n \end{pmatrix}$$

$$Ay = f$$

Эту систему будем решать методом прогонки. Решения будем искать в виде

$$y_{i-1} = \alpha_{i-1} y_i + \beta_{i-1},\ i = 1, ..n$$

где α_i, β_i - коэффициенты прогонки.

Подставим рекурентую формулу в уравнения и выразим y_i.

$$y_i = \frac{D_i - A_i\beta_{i-1}}{A_i\alpha_{i-1} + B_i} - \frac{C_i}{A_i\alpha_{i-1} + B_i} y_{i+1},\ i = 1, ..., n-1$$

Получили рекуррентные формулы для прогоночных коэффициентов α_i и β_i:

$$\begin{cases} \alpha_i = -\frac{C_i}{A_i\alpha_{i-1}+B_i},\ i = 1, ..., n-1, \\ \beta_i = \frac{D_i - A_i\beta_{i-1}}{A_i\alpha_{i-1}+B_i},\ i = 1, ..., n-1 \end{cases}$$

Коэффициенты α_0, β_0 определяются из первого граничного условия

$$y_0 = -\frac{C_0}{B_0} y_1 + \frac{D_0}{B_0}$$

$$\begin{cases} \alpha_0 = -\frac{C_0}{B_0}, \\ \beta_0 = \frac{D_0}{B_0} \end{cases}$$

Рассмотрим уравнение для $i = n-1$ и второе граничное условие:

$$\begin{cases} y_{n-1} = \alpha_{n-1} y_n + \beta_{n-1} \\ A_n y_{n-1} + B_n y_n = D_n \end{cases}$$

Выразим y_n:

$$y_n = \frac{D_n - A_n\beta_{n-1}}{A_n\alpha_{n-1} + B_n}$$

Вычислим прогоночные коэфиценты ($i = 0$ по выведенной формуле, $i = 2, ...n-1$ по рекуретной формуле), затем y_n и по рекуретнтой формуле найдём $y_i, i = 0, ..., n-1$.

Описание программы

Программа состоит из 2 модулей:

1. main.py - модуль который необходимо запустить для вычисления результата.
2. functions.py - модуль, содержащий функции $p(x), q(x), -f(x)$ для тестов.

В модуле main.py определена функция `def print_dots(x, y)` которая печатает массив точек `x` и массив точек `y` в виде: $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$.

Код программы

Код программы можно найти на https://github.com/YuriSavinykh/Numerical-Methods/tree/main/Task2/2

```python
import numpy as np
import matplotlib.pyplot as plt
from functions import *

def print_dots(x, y):
    for i in range(x.size):
        print("({:3.7f}, {:3.7f})".format(x[i], y[i]), end="")

        if i != x.size - 1:
            print(", ", end="")
        else:
            print()


if __name__ == '__main__':
    iterations_count = int(input("Input count of iterations: "))
    test_number = int(input("Test number: "))

    if iterations_count < 0:
        raise Exception("Incorrect data")

    if test_number < 1 or test_number > 4:
        raise Exception("Incorrect data")

    if test_number == 1:
        # y'' + 2 * y' - y / x = 3
        # y(0.2)=2
        # 0.5 * y(0.5) - y'(0.5) = 1
        p = p1; q = q1; f = f1

        xa = 0.2; a1 = 1; a2 = 0; a3 = 2
        xb = 0.5; b1 = 1; b2 = 0; b3 = 1
    elif test_number == 2:
        # y'' + y' = 1
        # y'(0) = 0
        # y(1) = 1
        p = p2; q = q2; f = f2

        xa = 0; a1 = 0; a2 = 1; a3 = 0
        xb = 1; b1 = 1; b2 = 0; b3 = 1
    elif test_number == 3:
        # y'' + 2 * y'=1
        # y(0) = 0
        # y'(1) = 1
        p = p3; q = q3; f = f3

        xa = 0; a1 = 1; a2 = 0; a3 = 0
        xb = 1; b1 = 0; b2 = 1; b3 = 1
    else:
        # y'' + 2 * y' - x * y = x^2
        # y'(0.6) = 0.7
```

```python
        # y(0.9) - 0.5 * y'(0.9) = 1
        f = f4; p = p4; q = q4;

        xa = 0.6; a1 = 0; a2 = 1; a3 = 0.7
        xb = 0.9; b1 = 1; b2 = -0.5; b3 = 1

    h = (xb - xa) / iterations_count
    y = np.empty(iterations_count + 1)
    alpha = np.empty(iterations_count)
    betta = np.empty(iterations_count)

    B0 = h * a1 - a2
    C0 = a2
    D0 = h * a3
    alpha[0] = -C0 / B0
    betta[0] = D0 / B0
    x = np.empty(iterations_count + 1)

    for i in range(iterations_count + 1):
        x[i] = xa + i * h

    for i in range(1, iterations_count):
        Ai = 2 - h * p(x[i])
        Bi = -4 + 2 * h ** 2 * q(x[i])
        Ci = 2 + h * p(x[i])
        Di = 2 * h ** 2 * f(x[i])

        alpha[i] = -Ci / (Bi + Ai * alpha[i - 1])
        betta[i] = (Di - Ai * betta[i - 1]) / (Bi + Ai * alpha[i - 1])

    An = -b2
    Bn = h * b1 + b2
    Dn = h * b3
    y[iterations_count] = (Dn - An * betta[iterations_count - 1]) / (An * alpha[iterations_count - 1] + Bn)

    for i in range(iterations_count, 0, -1):
        y[i - 1] = alpha[i - 1] * y[i] + betta[i - 1]

    print_dots(x, y)
    plt.plot(x, y)
    plt.show()
```

Листинг 1: main.py

```python
def p1(x):
    return 2


def q1(x):
    return -1 / x


def f1(x):
    return 3
```

```python


def p2(x):
    return 1


def q2(x):
    return 0


def f2(x):
    return 1


def p3(x):
    return 2


def q3(x):
    return 0


def f3(x):
    return 1


def p4(x):
    return 2


def q4(x):
    return -x


def f4(x):
    return x * x
```

Листинг 2: functions.py

Тестирование программы

По результатам работы программы были построенные графики. Красные точки - результат метода при количестве итераций равном 20, зелёные - 50. Чёрной цветом показан график аналитического решения. В тесте 1, 4 аналитически решение найти не удалось.

Тест №1 (Вариант 4)

$$\begin{cases} y'' + 2y' - \frac{y}{x} = 3 \\ y(0.2) = 2 \\ 0.5y(0.5) - y'(0.5) = 1 \end{cases}$$

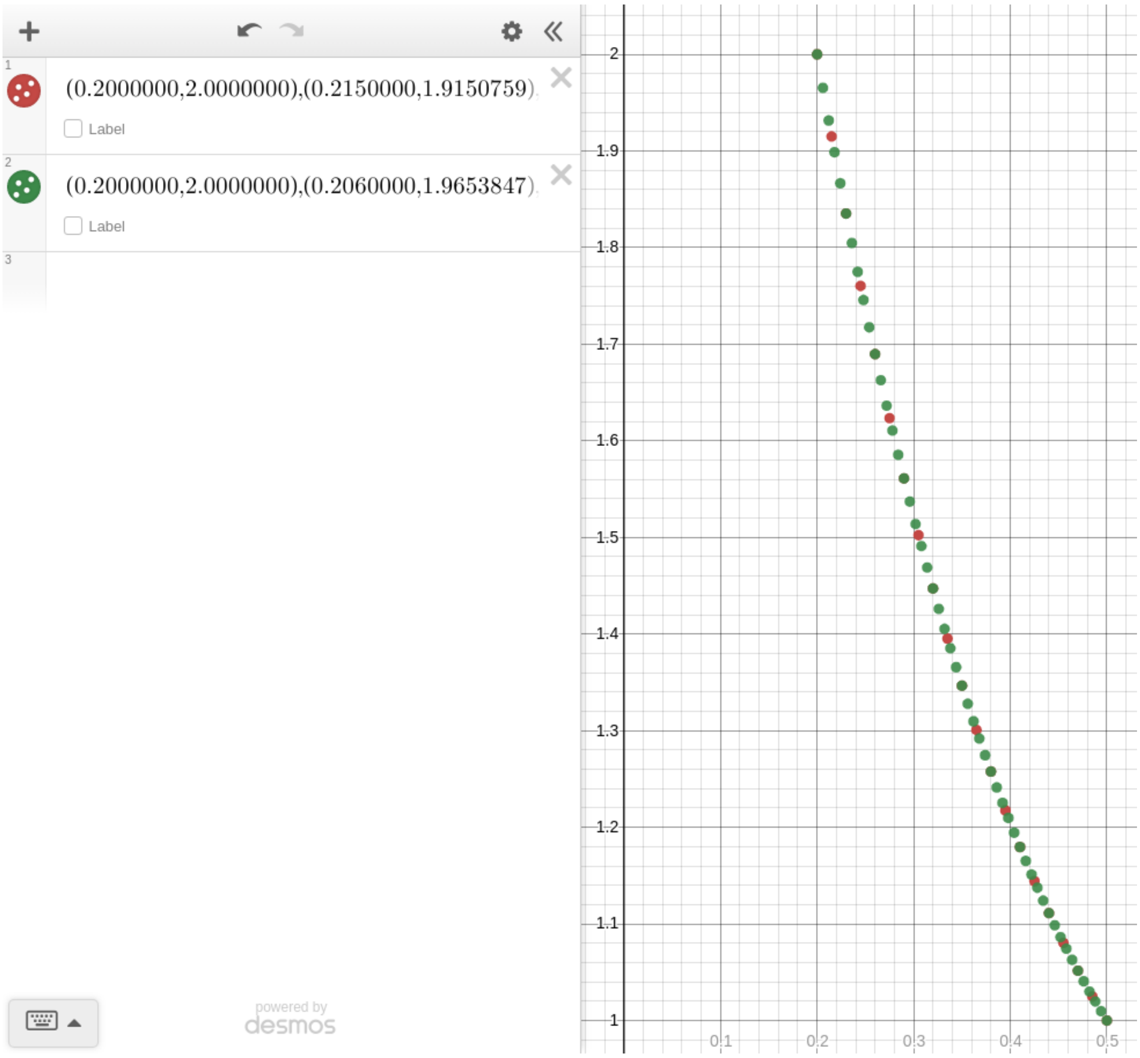


Рис. 1

Тест №2 (Дополнительный тест)

$$\begin{cases} y'' + y' = 1 \\ y'(0) = 0 \\ y(1) = 1 \end{cases}$$

Аналитическое решение: $y(x) = e^{-x} - e^{-1} + x$

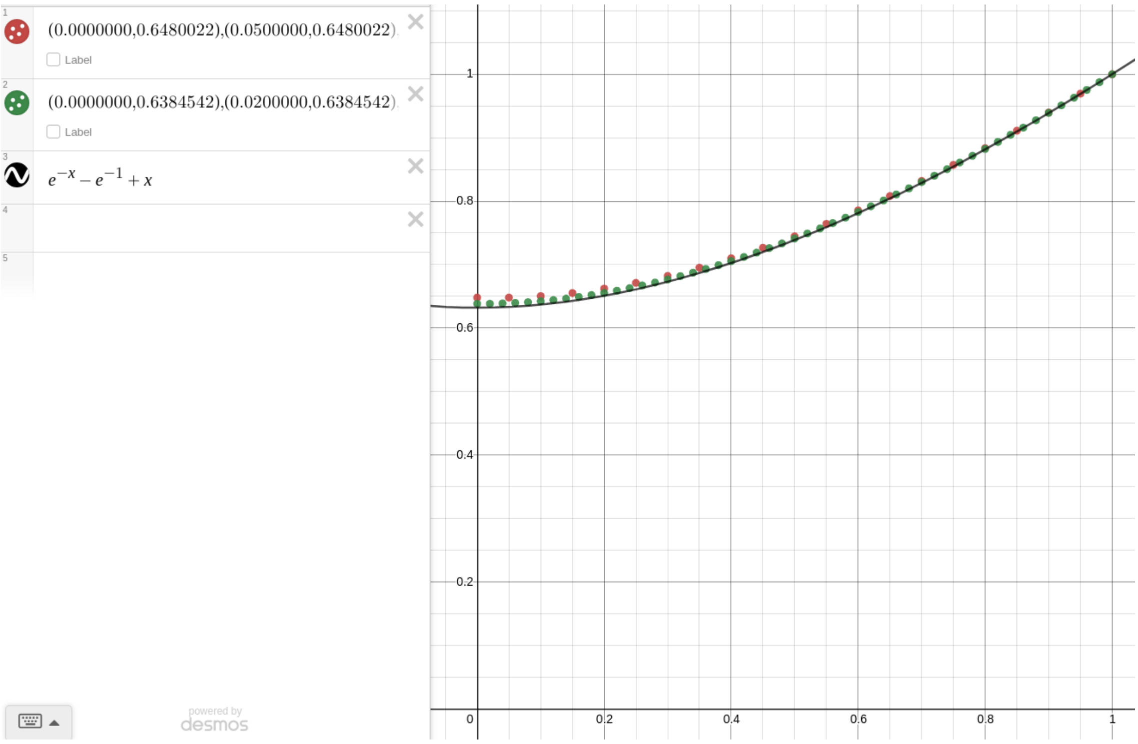


Рис. 2

Тест №3 (Дополнительный тест)

$$\begin{cases} y'' + y' = 1 \\ y'(0) = 0 \\ y(1) = 1 \end{cases}$$

Аналитическое решение: $y(x) = \frac{1}{4}(2x - e^{2-2x} + e^2)$

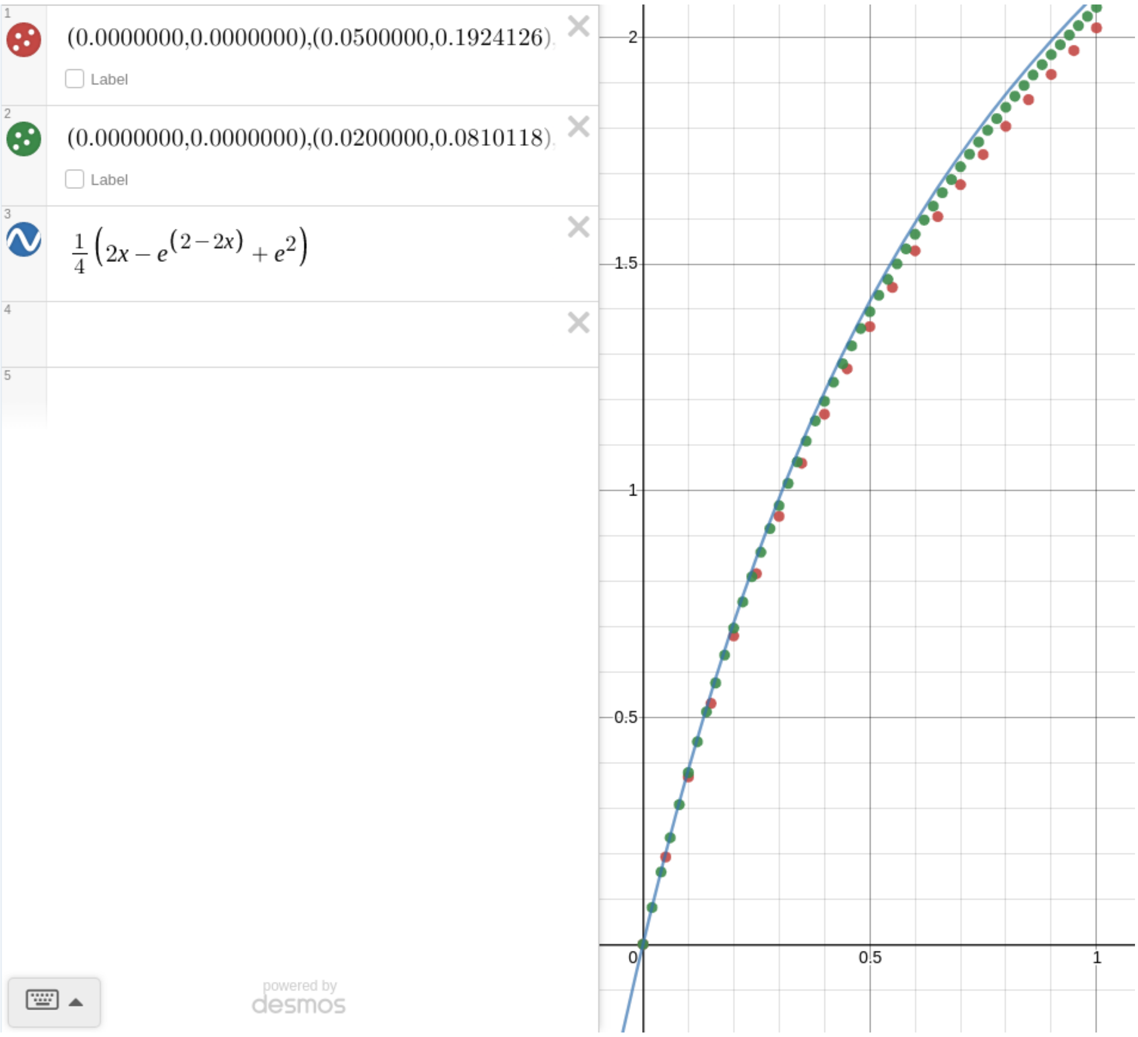


Рис. 3

Тест №4 (Дополнительный тест)

$$\begin{cases} y'' + 2y' - xy = x^2 \\ y'(0.6) = 0.7 \\ y(0.9) - 0.5y'(0.9) = 1 \end{cases}$$

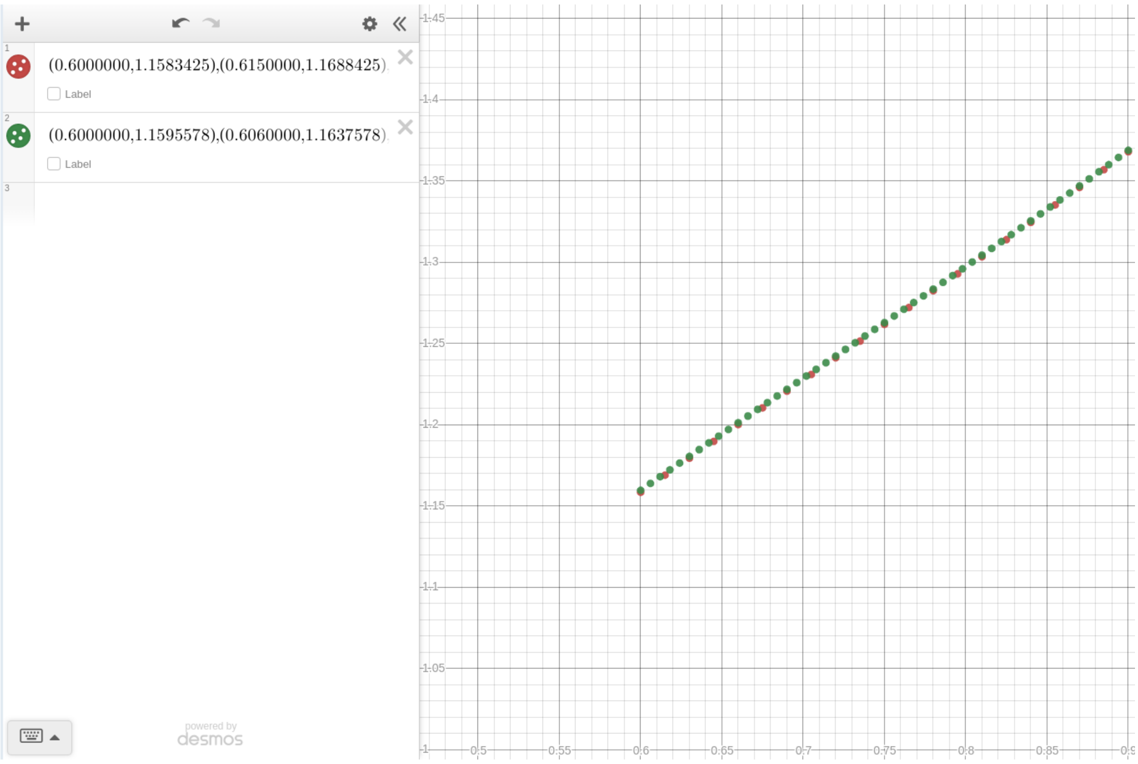


Рис. 4

Выводы

В ходе работы освоен метод решения краевой задачи для дифференциального уравнения второго порядка.

Краевая задача была сведена методом конечных разностей, аппроксимировав её разностной схемой второго порядка точности на равномерной сетке, к система конечно-разностных уравнений. Полученная система была решена методом прогнки.

Был построен график разностного решения задачи, который сопоставлен с графиком точного решения.