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 204 учебной группы факультета ВМК МГУ

Митрофанова Андрея Александровича

(фамилия, имя, отчество)

гор. Москва

2022 г.

Оглавление

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

Постановка задачи

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

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

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

$$y(x_0) = y_0 \ (2)$$

Предполагается, что правая часть уравнения, функция $f = f(x, y)$, такова, что гарантирует существование и единственность решения задачи Коши (1)-(2). В 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):

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

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

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

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 работы

1. Научиться находить решения задачи Коши (1)-(2) (или (3)-(4)) методами Рунге-Кутта второго и четвертого порядка точности.
2. Реализовать метод Рунге-Кутта второго и четвертого порядка точности, аппроксимировав дифференциальную задачу соответствующей разностной схемой (на равномерной сетке).
3. Найти численное решение задачи и построить его график.
4. Подтвердить корректность работы программы с использованием набора тестов.

Описание метода решения

Метод Рунге-Кутта второго порядка точности

Метод Рунге-Кутта для численного решения задачи Коши на отрезке [x_0,x_0+ l] :

$$\begin{cases} u'(x) = f(x, u(x)) \\ u(x_0) = u_0 \end{cases}$$

Сначала реализуем метод Рунге-Кутта второго порядка точности. Результатом работы алгоритма будет являться сеточная функция y(x_i), $x_i = x_0 + ih$, где $i \in \overline{0, \dots, n}$, а h -фиксированный шаг. В нашем случае сетка равномерная и равна $\mathrm{b} = \frac{1}{\mathrm{n}}$, где n - число шагов, которое подается на вход алгоритму. Метод Рунге-Кутта второго порядка точности предоставляет нам рекуррентные формулы для вычисления значения сеточной функции y_i :

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

В приведенном ниже решении $\alpha = \frac{1}{2}$. В этом случае формулы принимают вид: $y_{i+1} = y_i + \frac{\mathrm{h}}{2}(f(x_i, y_i) + f(x_i + h, y_i + hf(x_i, y_i)))$.

Метод Рунге-Кутта четвертого порядка точности

Второй порядок точности лучше, чем первый, однако практика показывает, что этой точности также недостаточно. Наиболее часто при проведении реальных расчетов используется схема Рунге-Кутта четвертого порядка точности. Метод определяется формулами

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

где

$$k_1 = f(x_i, y_i), \quad k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1)$$
$$k_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2), \quad k_4 = f(x_i + h, y_i + hk_3)$$

Отсюда получаем рекуррентную формулу

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

Метод Рунге-Кутта четвертого порядка точности для решения систем

Рассмотрим задачу Коши для нормальной системы дифференциальных уравнений

$$\begin{cases} y_1'(x) = f_1(x, y_1(x), y_2(x), ..., y_n(x)) \\ y_2'(x) = f_2(x, y_1(x), y_2(x), ..., y_n(x)) \\ ... \\ y_n'(x) = f_n(x, y_1(x), y_2(x), ..., y_n(x)) \\ y_1(x_0) = y_{1,0} \\ y_2(x_0) = y_{2,0} \\ ... \\ y_n(x_0) = y_{n,0} \end{cases}$$

В данном случае метод Рунге-Кутта также применим, однако теперь считаем y, f, k_i векторами размерности n, где n - число функций в системе. Формулы остаются прежними:

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

где

$$k_1 = f(x_i, y_i), \quad k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1)$$
$$k_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2), \quad k_4 = f(x_i + h, y_i + hk_3)$$

Отсюда получаем рекуррентную формулу

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

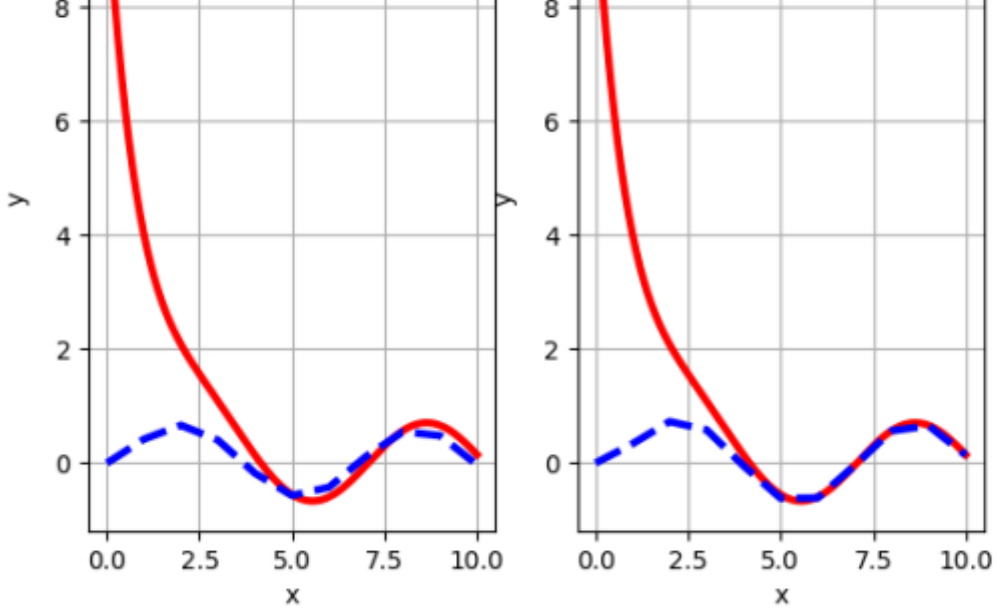
Тестирование и результаты эксперимента

Для решения этой задачи была написана программа, текст которой приведен в соответствующем разделе. Результаты программы проверялись на правильность с помощью сравнения графиков точного решения (если оно выражалось в элементарных функциях) и полученного численного решения. Приведем результаты работы программы для некоторых уравнений и систем.

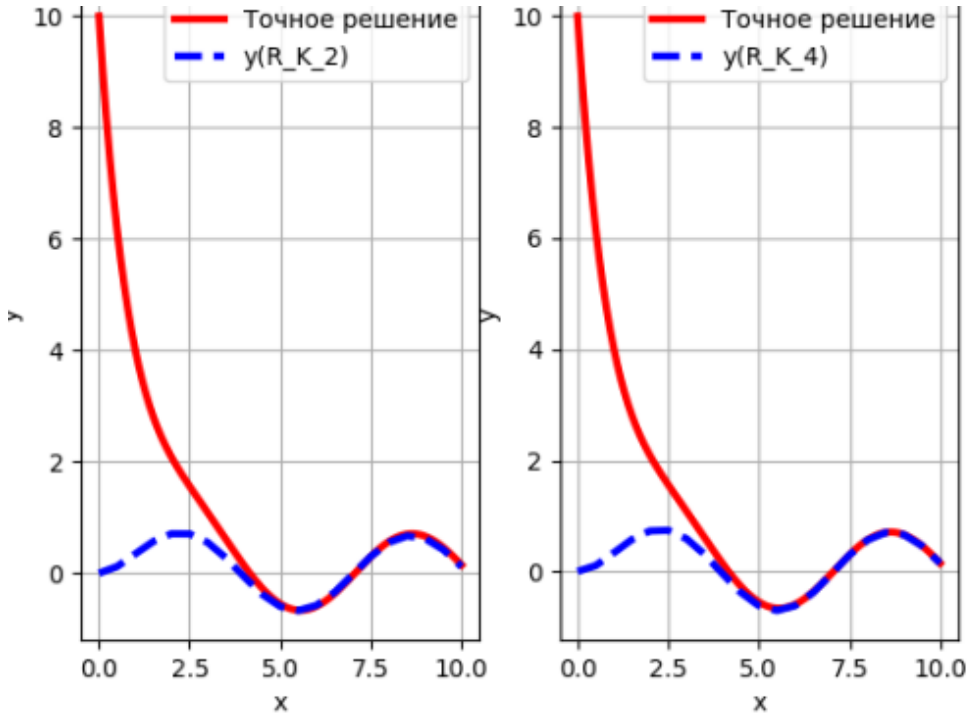
Тест 1

$f(x,y) = sin(x) - y, \qquad y(0) = 10$

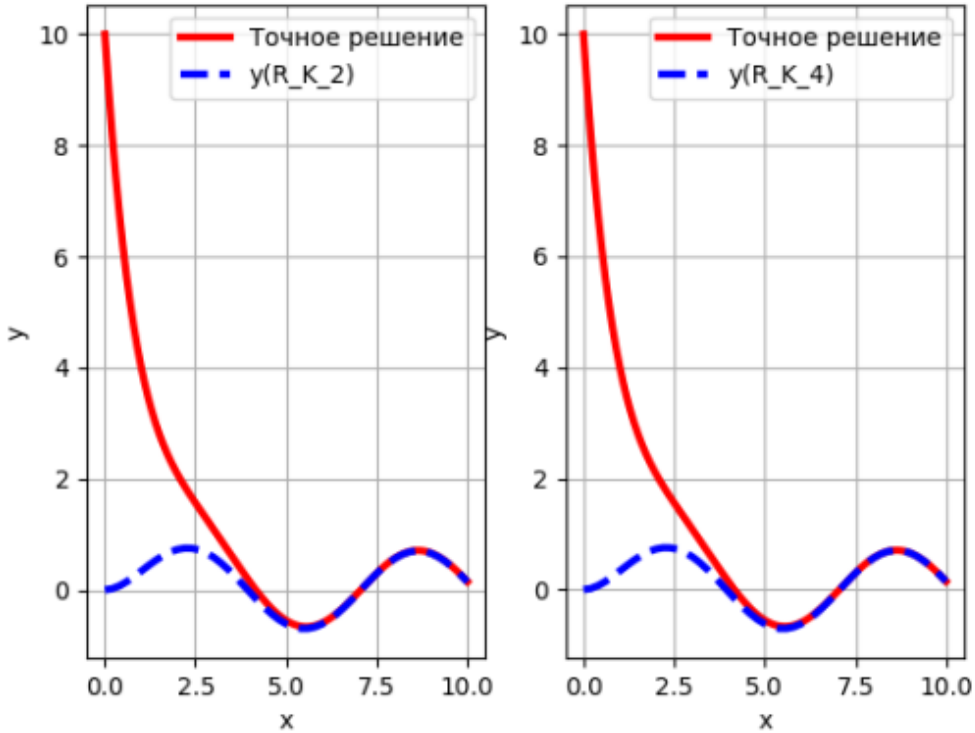
Рассмотрим отрезок [0, 5] и сравним результат с точным решением дифференциального уравнения при разных значения шага h:


y
x
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8

$h = 1$


Точное решение
y(R_K_2)
Точное решение
y(R_K_4)
y
x
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10

$h = 0.5$

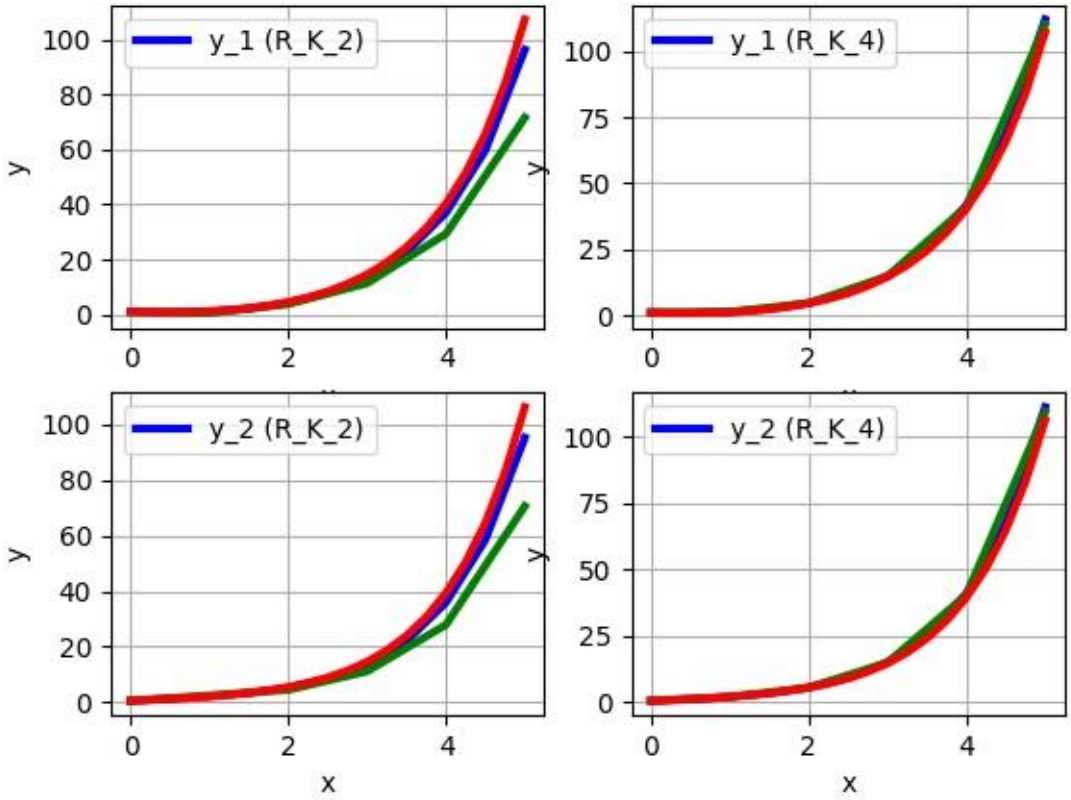

Точное решение
y(R_K_2)
Точное решение
y(R_K_4)
y
x
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10

$h = 0.2$

Тест 2

$$\begin{cases} f1(x,u,v) = v - cos(\mathrm{x}) \\ f2(x,u,v) = u + sin(x) \end{cases}$$

Для данного теста не удалось найти аналитическое решение.



На данной схеме графики синего цвета соответствуют решениям, построенным с шагом h = 0.25; зелёного цвета - с шагом h = 0.5; красного цвета - с шагом h = 0.125.

Программа

```python
import grid as grid
import numpy as np
import math
import matplotlib.pyplot as plt


# y' = sin(x) - y, y(0) = 10

def f1(x, y):
    return math.sin(x) - y
    # Точное решение теста выше
    # y = -0.5cos(x) + 0.5sin(x) + 21/2*exp(-x)


def sol1(x):
    y = list()
    for i in x:
        y.append(-0.5 * math.cos(i) + 0.5 * math.sin(i) + 10.5 * math.exp(-i))
    return y


# Первая система (решения, представимого в элементарных функциях, не сущ ествует):
# Первая система (решения, представимого в элементарных функциях, не сущ ествует):
# (y_1)' = y_2 - cos(x), y_1(0) = 0
# (y_2)' = y_1 + sin(x), y_2(0) = 0

# Вычисление производной y1
def f1_1(x, y):
    return y[1] - math.cos(x)


# Вычисление производной y2
def f1_2(x, y):
    return y[0] + math.sin(x)


# Метод Рунге-Кутта 2-ого порядка
def R_K_2(func, x_0, y_0, func_cnt, n, len):
    # Используется схема "предиктор - корректор"
    h = len / n  # шаг
    if func_cnt == 1:
        # Одно уравнение
        grid = dict()  # Сеточная функция
        grid[x_0] = y_0
        x_i = x_0
        y_i = y_0
        for i in range(0, n):
            x_new = x_i + h
            grid[x_new] = y_i + (func(x_i, y_i) + func(x_new, y_i + h * func(x_i, y_i))) * h / 2
            y_i = grid[x_new]
            x_i = x_new
        return grid
    else:
        # Система уравнений
        grid = dict()
        grid[x_0] = y_0
        x_i = x_0
```

```
            for j in range(0, func_cnt):
                y_vals1[j] = func[j](x_i, y_i)
                y_vals2[j] = func[j](x_new, y_i + h * y_vals1[j])
            grid[x_new] = y_i + (y_vals1 + y_vals2) * h / 2
            y_i = grid[x_new]
            x_i = x_new
        return grid

# Метод Рунге-Кутта 4-ого порядка
def R_K_4(func, x_0, y_0, func_cnt, n, len):
    h = len / n  # шаг
    grid = dict()  # Сеточная функция
    grid[x_0] = y_0
    x_i = x_0
    y_i = y_0
    if func_cnt == 1:
        # Одно уравнение
        for i in range(0, n):
            x_new = x_i + h
            k1 = func(x_i, y_i)
            k2 = func(x_i + h / 2, y_i + (h / 2) * k1)
            k3 = func(x_i + h / 2, y_i + (h / 2) * k2)
            k4 = func(x_new, y_i + h * k3)
            grid[x_new] = y_i + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
            y_i = grid[x_new]
            x_i = x_new
        return grid
    else:
        # Система уравнений
        for i in range(0, n):
            x_new = x_i + h
            k1 = np.zeros(func_cnt)
            k2 = np.zeros(func_cnt)
            k3 = np.zeros(func_cnt)
            k4 = np.zeros(func_cnt)
            for j in range(0, func_cnt):
                k1[j] = func[j](x_i, y_i)
            for j in range(0, func_cnt):
                k2[j] = func[j](x_i + h / 2, y_i + (h / 2) * k1)
            for j in range(0, func_cnt):
                k3[j] = func[j](x_i + h / 2, y_i + (h / 2) * k2)
            for j in range(0, func_cnt):
                k4[j] = func[j](x_new, y_i + h * k3)
            grid[x_new] = y_i + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
            y_i = grid[x_new]
            x_i = x_new
        return grid
```

Выводы

Мы реализовали метод Рунге-Кутта решения ОДУ, разрешённой относительно производной, с заданными начальными условиями (нормальной системы с заданными начальными условиями). Мы показали, что точность аппроксимации увеличивается с ростом числа разбиений n. Тестирование программы показало достаточно высокую эффективность метода Рунге-Кутта 4-го порядка (даже при относительно небольшом числе n он показывал приемлемую точность). В то же время метод Рунге-Кутта 2-го порядка точности часто работал с гораздо большей погрешностью даже при больших n.

Поэтому, когда нам нужна большая точность аппроксимации, следует использовать метод 4-го порядка. Метод 2-го порядка же подходит лишь в случаях, когда требуется оценить результат только примерно.

Решение краевой задачи для обыкновенного дифференциального уравнения второго порядка, разрешенного относительно старшей производной

Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида:

$$y''(x) + p(x)y'(x) + q(x)y(x) = -f(x)(5),$$

где p, q, f заданы, y(x) – искомая функция, $x \in [0,1]$.

Дополнительные условия в граничных точках

$$\begin{cases} \sigma_1 y(0) + \gamma_1 y'(0) = \delta_1 \\ \sigma_2 y(1) + \gamma_2 y'(1) = \delta_2 \end{cases}, (6)$$

Цели работы

1. Изучить метод конечных разностей для решения краевой задачи (5)-(6) и метод прогонки.
2. Реализовать метод конечных разностей и метод прогонки.
3. Решить краевую задачу (5)-(6) методом конечных разностей, аппроксимировав ее разностной схемой второго порядка точности (на равномерной сетке).
4. Полученную на предыдущем этапе систему конечно-разностных уравнений решить методом прогонки.
5. Найти разностное решение задачи и построить его график.
6. Найденное разностное решение сравнить с точным решением дифференциального уравнения.

Описание алгоритма

Опишем процесс нахождения численного решения краевой задачи для одного дифференциального уравнения $y'' + p(x)y' + q(x)y = -f(x)$ с дополнительными условиями в граничных точках:

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

Рассмотрим отрезок [a, b] и разобьем его на n частей: $x_i = a + ih$, где h = $\frac{b-a}{n}$, 0≤i≤n. Обозначим $y_i = y(x_1)$, $p_i = p(x_i)$, $q_i = q(x_i)$, $f_i = f(x_i)$.

Заменяя производные в исходном дифференциальном уравнении конечно-разностными отношениями, получаем:

$$\frac{y_{i+1}-2y_i+y_{i-1}}{h^2} + p_i\frac{y_{i+1}-y_{i-1}}{2h} + q_i y_i = -f_i, \quad 1 \le i \le n-1 \quad (3)$$

Аппроксимируем также производные в дополнительных условиях

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

Собирая коэффициенты при y_i, y_{i+1}, y_{y-1} в уравнении (3) и при y_0, y_1, y_n, $y_n + 1$ получим следующую СЛАУ с неизвестными $y_0, y_1, \ldots, y_n$:

$$\begin{cases} \mathrm{y}_0(\sigma_1 - \frac{\gamma_1}{h}) + y_1\frac{\gamma_1}{h} = \delta_1 \\ \mathrm{A}_i y_{i-1} - C_i y_i + B_i y_{i+1} = F_i, \quad 1 \le i \le n-1 \\ \mathrm{y}_n(\sigma_2 - \frac{\gamma_2}{h}) + y_{n+1}\frac{\gamma_2}{h} = \delta_2 \\ \mathrm{A}_i = \frac{1}{h^2} - \frac{p_i}{2h} \\ \mathrm{B}_i = \frac{1}{h^2} - \frac{p_i}{2h} \\ \mathrm{C}_i = \frac{2}{h^2} - q_i \\ \mathrm{F}_i = f_i. \end{cases}$$

Данная система имеет трехдиагональную матрицу коэффициентов. Следовательно, она может быть решена методом прогонки. Решение будем искать в виде $y_i = \alpha_i y_{i+1} + \beta_i$. Так как $y_0 = \alpha_0 y_1 + \beta_0$, то, преобразовав первое уравнение дополнительных условий, получим, что 0 = $-\frac{\gamma_1}{\sigma_1 h - \gamma_1}$, $\beta_0 = \frac{\delta_1 h}{\sigma_1 h - \varphi_1}$.

Преобразуя уравнения (3) и учитывая, что $y_i = \alpha_i y_{i+1} + \beta_i$, получим рекуррентные формулы для α_1 и β_i:

$$\alpha_{i+1} = \frac{B_i}{C_i - \alpha_i A_i}, \quad \beta_{i+1} = \frac{A_i\beta_i - F_i}{C_i - \alpha_i A_i}, \quad 1 \le i \le n-1.$$

Из второго уравнения дополнительных условий с учетом того, что $y_i = \alpha_i y_{i+1} + \beta_i$, получим, что $y_n = \frac{\delta_2 h + \gamma_2 \beta_{n-1}}{\sigma_2 h + \gamma_2(1 - \alpha_{n-1})}$.

С помощью данной формулы мы сможем найти численное решение в правой точке сетки. Зная его, по формуле $y_{i-1} = \alpha_{i-1} y_i + \beta_{i-1}$ можно высчитать значение в предыдущей точке. И так далее. Данный алгоритм как раз и позволяет найти численное решение краевой задачи.

Тестирование и результаты эксперимента

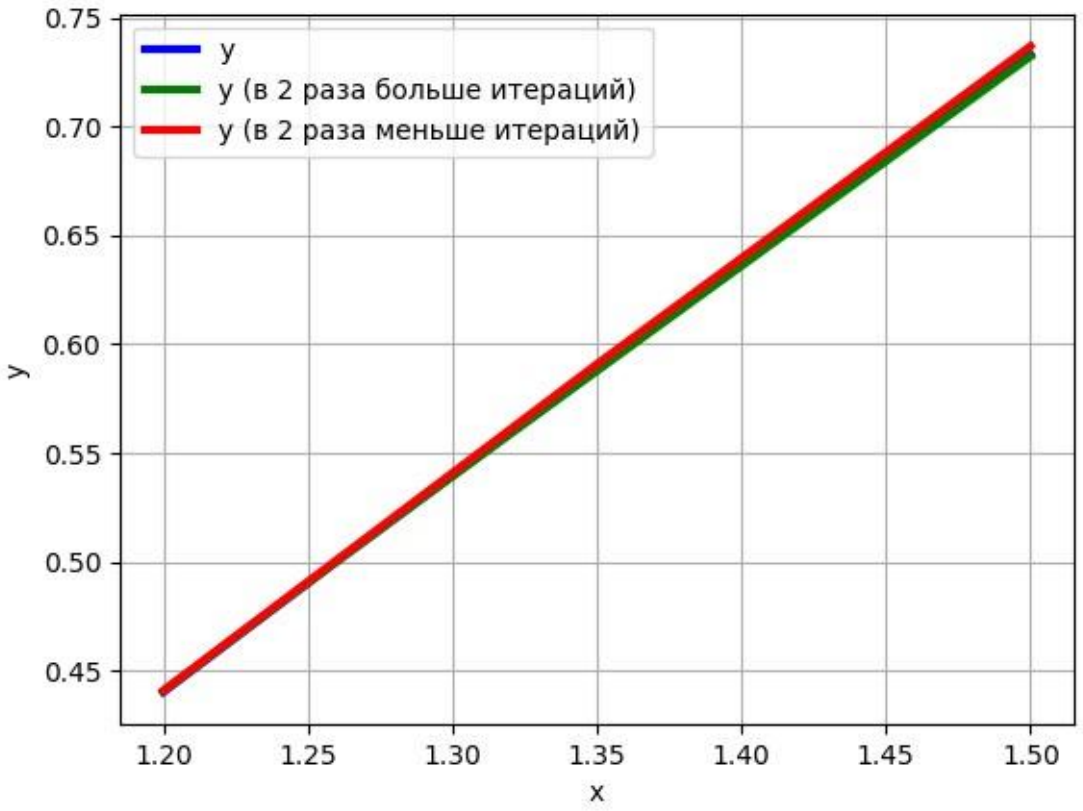
Для решения этой задачи была написана программа, текст которой приведен соответствующем разделе. Результаты программы проверялись на правильность с помощью сравнения графиков.

Тест 1

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

Для этого уравнения не существует решения, представимого в виде элементарных функций. Поэтому вместо сравнения с графиками точных решений покажем с помощью графиков, как изменяется решение при изменении размера шага (кол-ва итераций).

Рассматривается отрезок [1.2, 1.5]. Приведём полученные численные решения для шага h = 0.06 (5 итераций):



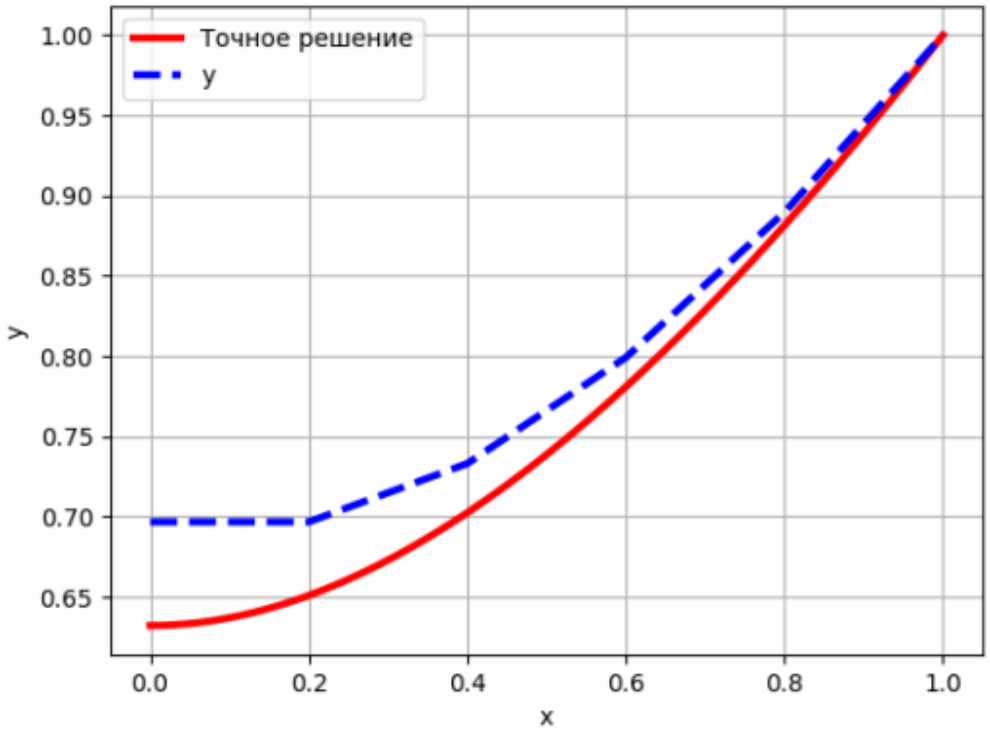
Тест 2

$$y'' + y' = 1$$
$$y'(0) = 0$$
$$y(1) = 1$$

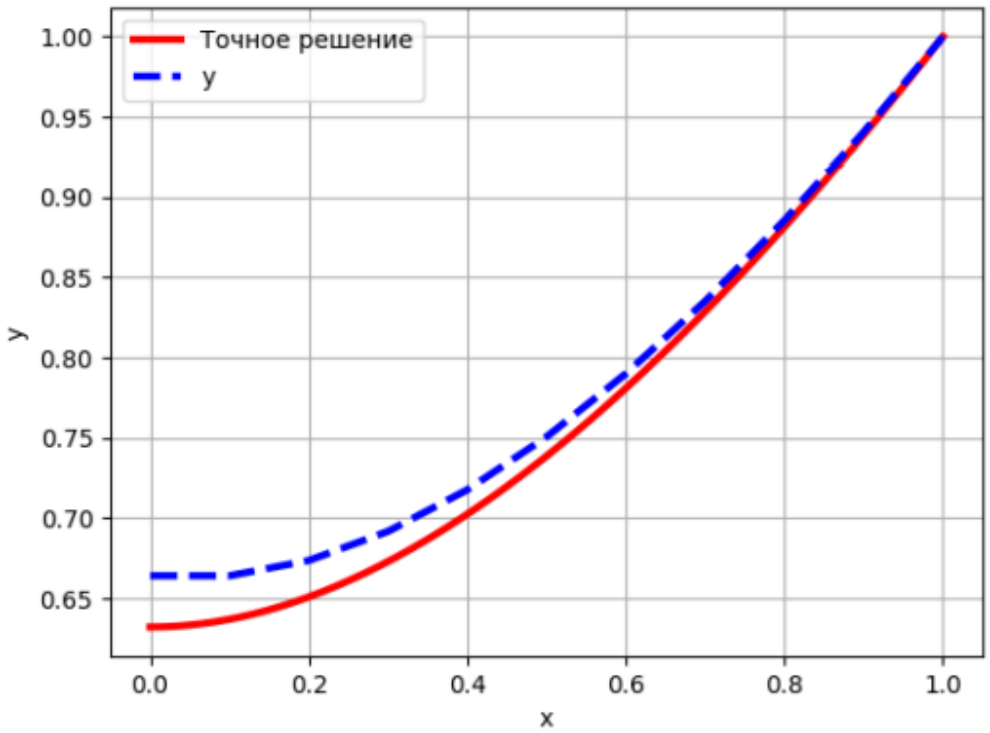
Решением данного дифференциального уравнения является функция

$$y = x + e^{-x} - \frac{1}{e}$$

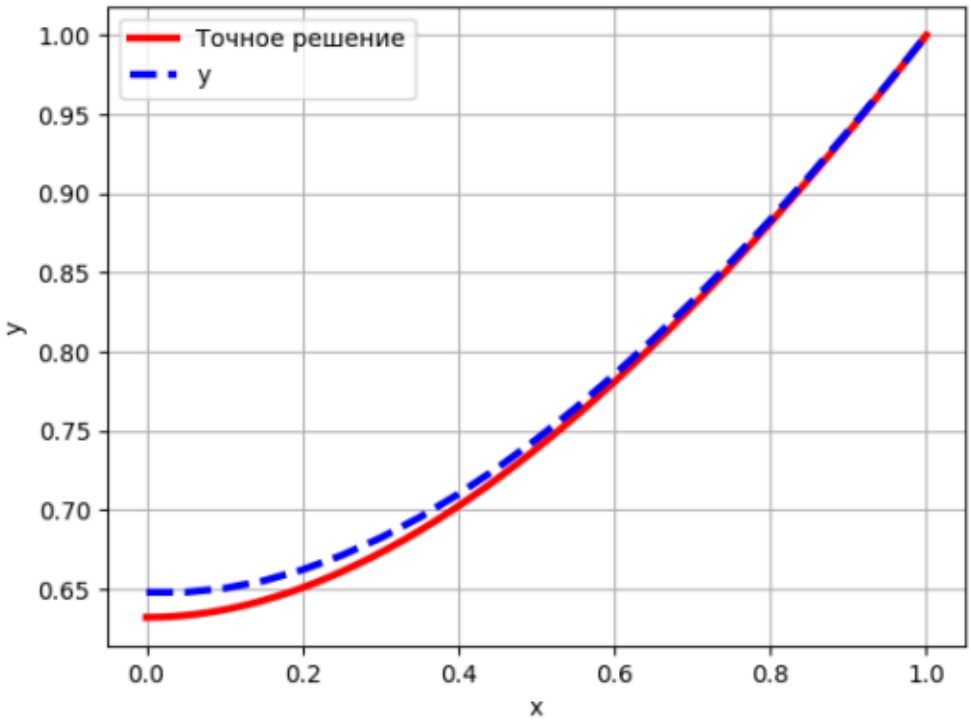
Рассматривается отрезок [0, 1]. Сравним результат с точными решениями системы при разном количестве итераций h:



$\mathrm{h} = 0.2$



$\mathrm{h} = 0.1$


Точное решение
y
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
y
0.0
0.2
0.4
0.6
0.8
1.0
x

$h = 0.05$

Программа

```python
import numpy as np
import math
import matplotlib.pyplot as plt

global f, p, q, a_1, a_2, a, c_1, b_1, b_2, b, c_2, x_g, y_g
# p = p(x) - функция при первой производной
# q = q(x) - функция при y
# f - правая часть уравнения
# a_1, a_2 - коэффициенты в 1 дополнительном условии
# b_1, b_2 - коэффициенты в 2 дополнительном условии
# a, b - левая и правая границы
# c_1, c_2 - правая часть доп. условий
# x_g - сетка
# y_g - сеточная функция
# y'' - 0.5xy' + y = 2, y(0.4) = 1.2, y(0.7) + 2y'(0.7) = 1.4
# Решения, представимого в элементарных функциях, не существует


def p_1(x):
    return 3


def q_1(x):
    return -1 / x


def f_1(x):
    return x + 1


def probl_1():
    global f, p, q, a_1, a_2, a, c_1, b_1, b_2, b, c_2
    f = f_1
    p = p_1
    q = q_1
    a_1 = 0
    a_2 = 1
    a = 1.2
    c_1 = 1
    b_1 = 2
    b_2 = -1
    b = 1.5
    c_2 = 0.5
```

```python
# y'' + y' = 1, y'(0) = 0, y(1) = 1
# Решение: y = x + exp(-x) - 1 / exp(1)
def sol_2(x):
    y = list()
    for i in x:
        y.append(i + math.exp(-i) - 1 / math.exp(1))
    return y


def p_2(x):
    return 1


def q_2(x):
    return 0


def f_2(x):
    return 1


def probl_2():
    global f, p, q, a_1, a_2, a, c_1, b_1, b_2, b, c_2
    f = f_2
    p = p_2
    q = q_2
    a_1 = 0
    a_2 = 1
    a = 0
    c_1 = 0
    b_1 = 1
    b_2 = 0
    b = 1
    c_2 = 1


def solution_task(n):
    # Решение задачи с помощью метода прогонки
    global x_g, y_g
    h = (b - a) / n
    x_i = a
```

```
    x_i = a
    x_g = np.zeros(n + 1)
    x_g[0] = x_i
    y_g = np.zeros(n + 1)
    # Прогоночные коэффициенты
    alpha = np.zeros(n + 1)
    beta = np.zeros(n + 1)
    alpha[1] = -a_2 / (a_1 * h - a_2)
    beta[1] = c_1 * h / (a_1 * h - a_2)
    for i in range(1, n):
        x_i += h
        x_g[i] = x_i
        # Коэффициенты трёхдиагональной СЛАУ
        A_i = 1 / (h * h) - p(x_i) / (2 * h)
        C_i = 2 / (h * h) - q(x_i)
        B_i = 1 / (h * h) + p(x_i) / (2 * h)
        F_i = f(x_i)
        # Вычисление прогоночных коэффициентов по реккурентным формулам
        alpha[i + 1] = B_i / (C_i - A_i * alpha[i])
        beta[i + 1] = (beta[i] * A_i - F_i) / (C_i - A_i * alpha[i])
    # Обратная прогонка
    y_g[n] = (b_2 * beta[n] + c_2 * h) / (b_2 * (1 - alpha[n]) + b_1 * h)
    for i in range(n, 0, -1):
        y_g[i - 1] = y_g[i] * alpha[i] + beta[i]
    x_g[n] = b

```

Выводы

В данной работе был освоен и реализован метод прогонки решения краевой задачи для дифференциального уравнения второго порядка. Данный метод оказался простым в реализации. Экспериментально было показано, что метод достаточно точно вычисляет решения при малом кол-ве итераций. Также было показано, что при увеличении числа итераций, точность решения задачи увеличивается.

Список литературы

[1] Костомаров Д.П., Фаворский А.П. Вводные лекции по численным методам: Учеб. Пособие. – М.: Университетская книга, Логос, 2006

[2] Самарский АА. Введение в численные методы. Москва: Издательство «Наука», 1982