МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

Численные методы решения дифференциальных уравнений

Задание №2

Отчет

о выполненном задании

студента 201 учебной группы факультета ВМК МГУ
Долгушева Глеба Дмитриевича

гор. Москва
2023

Содержание

Подвариант 1

Постановка задачи

Пусть дана задача Коши для обыкновенного дифференциального уравнения первого порядка, разрешенным относительно производной:

$$\begin{cases} y' = f(x, y), x > x_0 \\ y(x_0) = y_0 \end{cases}$$

Или задача Коши для системы обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производной:

$$\begin{cases} {y_1}' = f_1(x, y_1, y_2), x > x_0 \\ {y_2}' = f_2(x, y_1, y_2), x > x_0 \\ y_1(x_0) = y_{01} \\ y_2(x_0) = y_{02} \end{cases}$$

Во всех случаях предполагается, что параметры задачи таковы, что выполнены условия существования и единственности решения.
Требуется численно решить эти задачи.

Задачи практической работы

- Решить задачу Коши 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 уравнения в случае системы), представляющее фактически некоторую рекуррентную формулу, просчитать численно с использованием программных средств (Для данной задачи был выбран язык Python 3.10)
- Найти численное решение задачи и построить его график
- Подобрать тесты, на которых точное решение выражается в элементарных функциях
- Найденное численное решение сравнить с точным решением дифференциального уравнения

Описание алгоритма решения

Рассмотрим задачу Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно старшей производной:

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

с начальным условием

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

Тогда для нахождения численного решения задачи Коши (1) (2) на отрезке $[x_0, x_0 + a]$ воспользуемся следующим алгоритмом.
Зафиксируем шаг $h := \frac{a}{n}$, где n - количество точек или же количество шагов алгоритма, значения функции в которых мы хотим посчитать для приблизительного нахождения функции (например, затем можем интерполировать ее, допустим, сплайнами). Тогда для аргумента построим следующую равномерную сетку: $x_{i+1} = x_i + h$. И ей сопоставим следующую рекуррентную формулу:

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

Данный алгоритм является методом Рунге-Кутта второго порядка точности. Для получения метода Рунге-Кутта четвертого порядка точности нужно использовать следующую формулу:

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

Где коэффициенты можно найти из следующих соотношений

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1) \\ k_3 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2) \\ k_4 &= f(x_i + h, y_i + k_3 h) \end{aligned}$$

Пусть теперь дана задача Коши для системы обыкновенных дифференциальных уравнений первого порядка, разрешенного относительно производной:

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

И начальные условия:

$$\begin{cases} y_1(x_0) = y_{01} \\ y_2(x_0) = y_{02} \\ ... \\ y_n(x_0) = y_{0n} \end{cases} \tag{4}$$

Тогда метод Рунге-Кутта второго порядка для решения задачи Коши (3) (4) переписывается следующим образом:

$$y_{j,i+1} = y_{j,i} + \frac{h}{2}(f_j(x_i, y_{j,i}) + f_j(x_i + h, y_{j,i} + hf_j(x_i, y_{j,i}))$$

Где i - номер итерации, j - номер решения. Метод Рунге-Кутта четвертого порядка точности переписывается аналогично.

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

Соотношения для коэффициентов

$$\begin{aligned}
k_1^{(j)} &= f_j(x_i, y_i^{(j)}) \\
k_2^{(j)} &= f_j(x_i + \frac{h}{2}, y_i^{(j)} + \frac{h}{2}k_1^{(j)}) \\
k_3^{(j)} &= f_j(x_i + \frac{h}{2}, y_i^{(j)} + \frac{h}{2}k_2^{(j)}) \\
k_4^{(j)} &= f_j(x_i + h, y_i^{(j)} + k_3^{(j)}h)
\end{aligned}$$

Тесты

Все тесты были проверены с помощью вычислительных алгоритмов, предоставляемых сайтом `https://www.wolframalpha.com/`

Тест 1

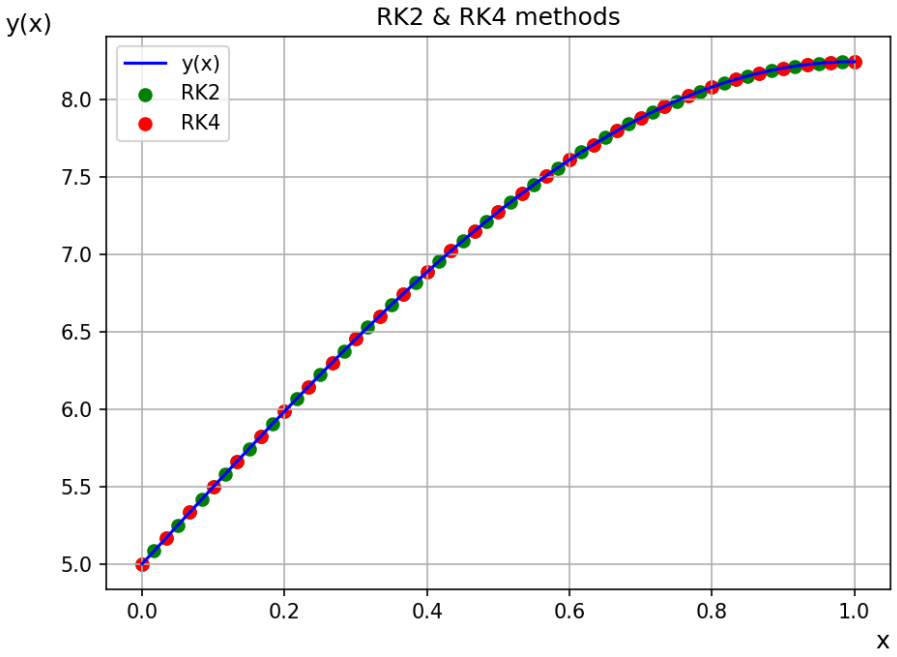
Задача Коши (в соответствии с вариантом):

$$\begin{cases} y' = y - yx \\ y(0) = 5 \end{cases}$$

Точное аналитическое решение:

$$y = 5e^{-\frac{1}{2}x(x-2)}$$

Рассмотрим данную задачу на отрезке $[0, 1]$:



Синяя линия - точное аналитическое решение, зеленые точки - решение, полученное методом Рунге-Кутта второго порядка точности, красные точки - решение, полученное методом Рунге-Кутта четвертого порядка точности.

Тест 2

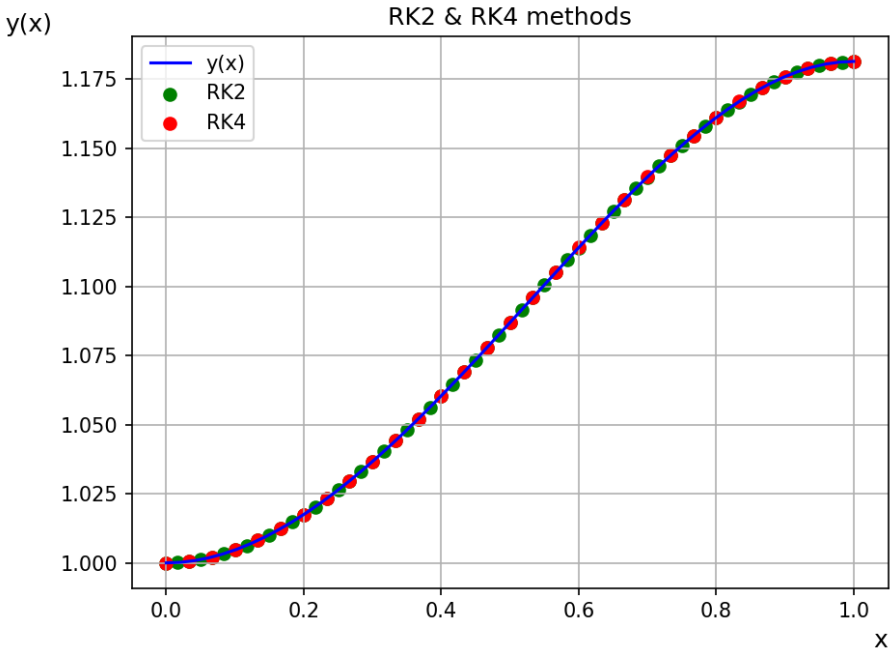
Задача Коши:

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

Решение:

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

Рассмотрим данную задачу на отрезке $[0, 1]$:



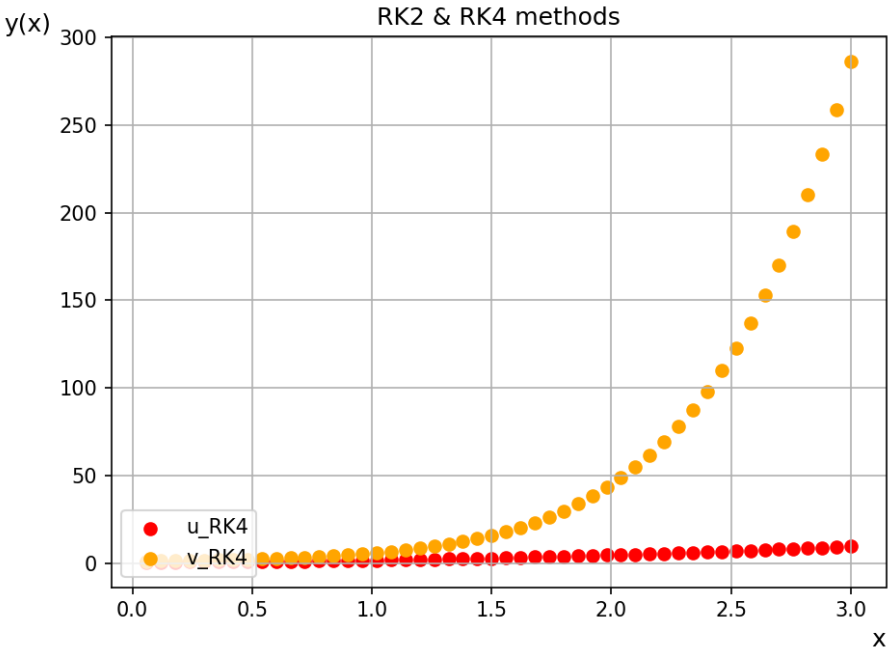
Синяя линия - точное аналитическое решение, зеленые точки - решение, полученное методом Рунге-Кутта второго порядка точности, красные точки - решение, полученное методом Рунге-Кутта четвертого порядка точности.

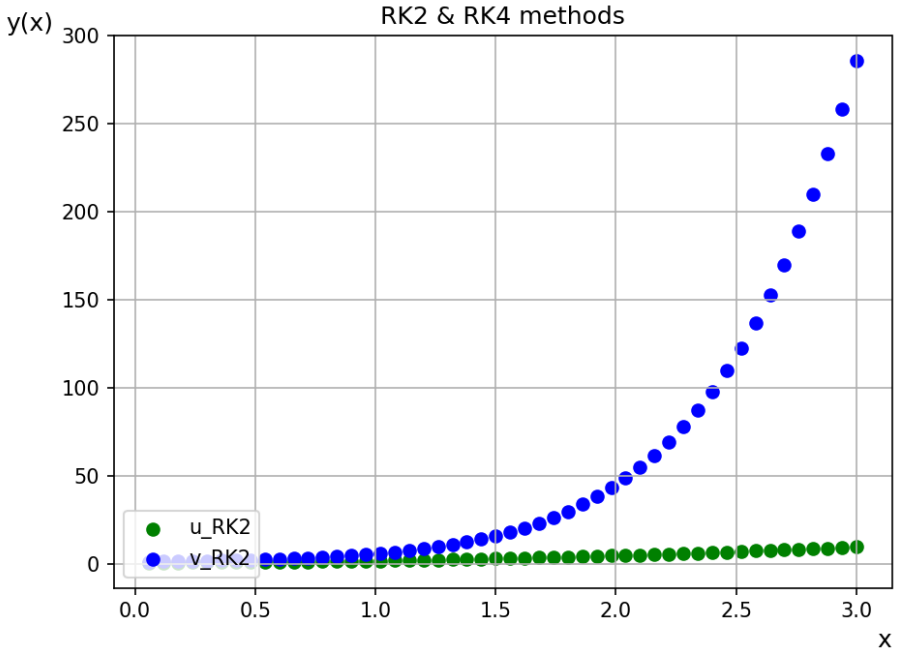
Тест 3

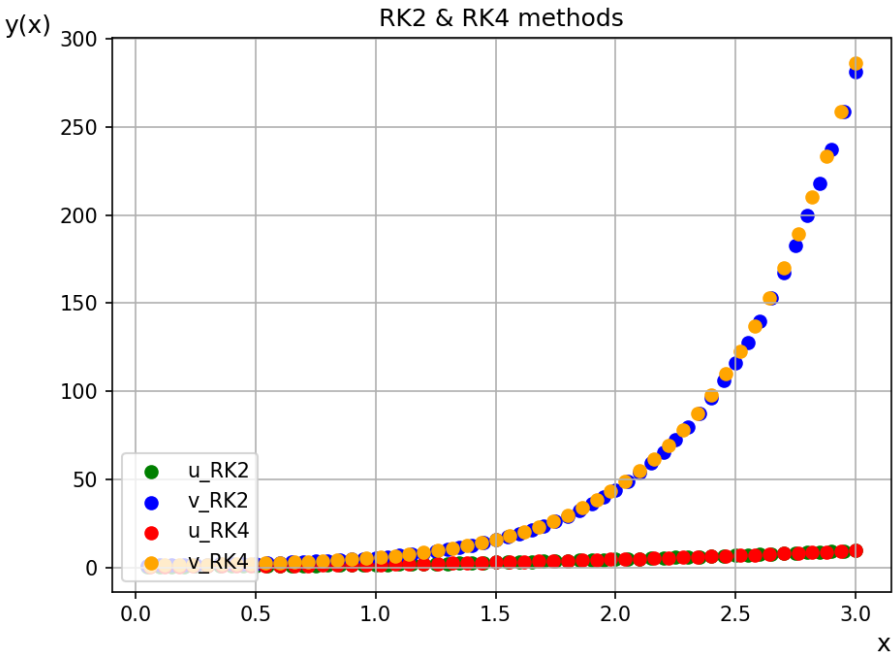
Задача Коши (в соответствии с вариантом):

$$\begin{cases} y_1' = exp^{-(y_1^2+y_2^2)} + 2x \\ y_2' = 2y_1^2 + y_2 \\ y_1(0) = 0.5 \\ y_2(0) = 1 \end{cases}$$

Рассмотрим данную задачу на отрезке $[0, 3]$. Приближение для функции y_1 (зеленые точки - второй порядок точности, красные - четвертый). Приближение для функции y_2 (синие точки - второй порядок точности, оранжевые - четвертый)


RK2 & RK4 methods
y(x)
300
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x
u_RK4
v_RK4


RK2 & RK4 methods
y(x)
300
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x
u_RK2
v_RK2



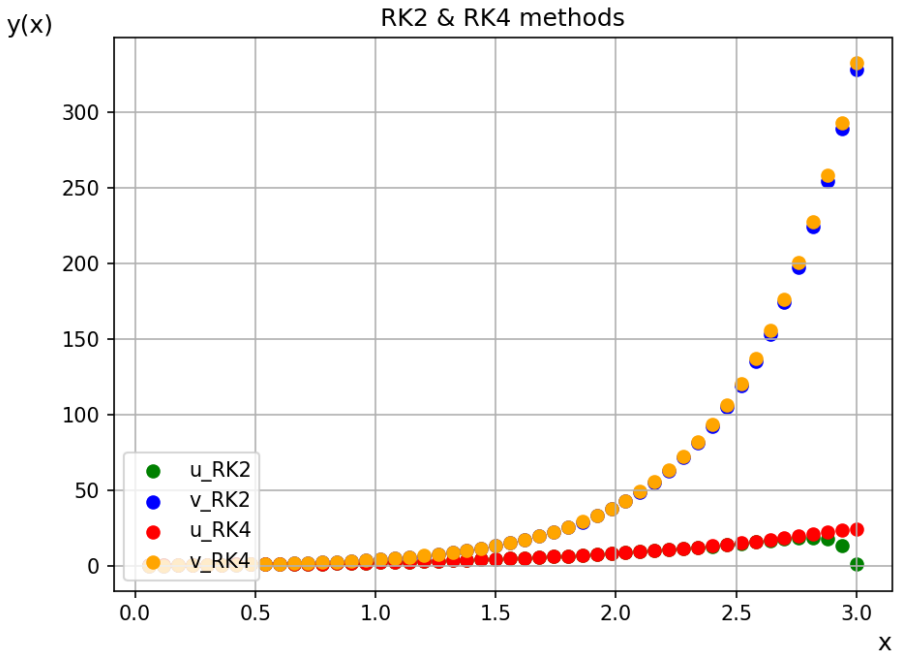
Как можно заметить, графики мало отличаются друг от друга, оба метода дают хорошее приближение искомой функции, поэтому в следующем тесте попробуем показать иную ситуацию.

Тест 4

Задача Коши:

$$\begin{cases} y_1' = 2.1y_2 - y_1^2 \\ y_2' = e^{-y_1} + x + 2.1y_2 \\ y_1(0) = 1 \\ y_2(0) = 0.25 \end{cases}$$

Рассмотрим данную задачу на отрезке $[0, 3]$. Приближение для функции y_1 (зеленые точки - второй порядок точности, красные - четвертый). Приближение для функции y_2 (синие точки - второй порядок точности, оранжевые - четвертый)



В данном случае можно легко увидеть, что при недостаточно малом разбиении функция, найденная методом Рунге-Кутта второго порядка точности, значительно разнится с найденной методом Рунге-Кутта четвертого порядка точности, что показывает преимущество второго метода.

Выводы

В ходе практической работы были изучены и реализованы методы Рунге-Кутта второго и четвертого порядка точности для задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной и для системы обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производной. Кроме того в ходе тестов было показано, что второй алгоритм действительно может показывать лучшую сходимость по сравнению с первым, но в то же время далеко не всегда эта точность критична. При этом метод Рунге-Кутта четвертого порядка точности имеет более сложную реализацию и несколько большие расходы накладных ресурсов.

Подвариант 2

Постановка задачи

Рассмотрим обыкновенное дифференциальное уравнение второго порядка вида:

$$y'' + p(x)y' + q(x) + f(x) = 0, 0 < x < 1$$

С дополнительными (краевыми) условиями:

$$\begin{cases} \alpha_1 y(0) + \beta_1 y'(0) = \delta_1 \\ \alpha_2 y(0) + \beta_2 y'(0) = \delta_2 \end{cases}$$

Требуется решить данную задачу численно

Задачи практической работы

- Реализовать алгоритм решения поставленной краевой задачи методом конечных разностей, аппроксимируя ее разностной схемой второго порядка точности (на равномерной сетке). Полученную систему конечно-разностных уравнений решить методом прогонки (Для данной задачи был выбран язык Python 3.10)
- Найти разностное решение задачи и построить его график
- Подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций
- Найденное разностное решение сравнить с точным решением дифференциального уравнения

Описание алгоритма решения

Рассмотрим обыкновенное дифференциальное уравнение второго порядка вида:

$$y'' + p(x)y' + q(x) + f(x) = 0, 0 < x < 1$$

С дополнительными (краевыми) условиями:

$$\begin{cases} \alpha_1 y(0) + \beta_1 y'(0) = \delta_1 \\ \alpha_2 y(0) + \beta_2 y'(0) = \delta_2 \end{cases}$$

Тогда для нахождения численного решения задачи на отрезке $[a, b]$ воспользуемся следующим алгоритмом.
Зафиксируем шаг

$$h := \frac{b-a}{n},$$

где n - количество точек (количество шагов алгоритма), значения функции в которых мы хотим посчитать для приблизительного нахождения функции (например, затем можем интерполировать ее, допустим, сплайнами). Тогда для аргумента построим следующую равномерную сетку: $x_{i+1} = x_i + h, x_0 = a$. Заменим в имеющемся уравнении производные конечно-разностными отношениями:

$$y' = \frac{y_{i+1} - y_{i-1}}{2h}$$
$$y'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

Тогда первоначальное уравнение примет вид:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p(x_i)\frac{y_{i+1} - y_{i-1}}{2h} + q(x_i)y_i + f(x_i) = 0$$

Применим аналогичное преобразование для краевых условий:

$$\alpha_1 \frac{y_0 + y_1}{2} + \beta_1 \frac{y_1 - y_0}{h} = \delta_1$$
$$\alpha_2 \frac{y_n + y_{n+1}}{2} + \beta_2 \frac{y_{n+1} - y_n}{h} = \delta_2$$

Соберем слагаемые в полученных уравнениях так, чтобы привести их к виду:

$$\begin{cases} B_0 y_0 + C_0 y_1 = F_0 \\ A_i y_{i-1} + B_i y_i + C_i y_{i+1} = F_i, i = 1 \dots n \\ A_{n+1} y_n + B_{n+1} y_{n+1} = F_{n+1} \end{cases} \quad (5)$$

Система уравнений (5) имеет ненулевые элементы матрицы коэффициентов только на трех диагоналях, то есть является трехдиагональной матрицей. Для такого типа матриц, конечно, можно применять стандартные методы решения систем линейных алгебраических уравнений, такие как метод Гаусса или метод верхней релаксации, но куда эффективнее будет воспользоваться методом прогонки. Запишем рекуррентную формулу для $y_i, i = 1 \dots n$

$$y_i = \phi_{i+1} y_{i+1} + \psi_{i+1}.$$

Подставим его в нашу систему и потребуем, чтобы равенство выполнялось независимо от значений x_i и x_{i+1}. Тогда получим:

$$\begin{cases} A_i\phi_i\phi_{i+1} + C_i\phi_{i+1} + B_i = 0 \\ A_i\phi_i\psi_{i+1} + A_i\psi i + C_i\phi_{i+1} - D_i = 0 \end{cases}$$

Решив эту систему руками, получим рекуррентную формулу:

$$\phi_{i+1} = \frac{-B_i}{A_i\phi_i + C_i}$$

$$\psi_{i+1} = \frac{D_i - A_i\psi_i}{A_i\phi_i + C_i}$$

То есть теперь мы можем итеративно найти все коэффициенты α_i, β_i, идя в сторону возрастания i. Тогда для завершенного алгоритма необходимо задать начальные значения. Их можно получить из преобразования начальных условий:

$$\phi_0 = \frac{-\alpha_1}{\beta_1 h - \alpha_1}$$

$$\psi_0 = \frac{\delta_1}{\beta_1 - \frac{\alpha_1}{h}}$$

Затем, идя в сторону убывания i, найдем решение $y_{n+1}, ..., y_1$. Для начала итерации используем, что:

$$y_n = \frac{\delta_2 h + \alpha_2\psi_{n-1}}{\beta_2 h + \alpha_2(1 - \phi_{n-1})}$$

Таким образом, мы получили решение краевой задачи методом прогонки.
Формулы для расчета A_i, B_i, C_i:

$$\begin{aligned} A_i &= \frac{1}{h^2} - \frac{p(x_i)}{2h} \\ B_i &= \frac{1}{h^2} + \frac{p(x_i)}{2h} \\ C_i &= -\frac{2}{h^2} + q(x_i) \end{aligned} \tag{6}$$

и для B_0, C_0, A_n, B_n

$$\begin{aligned} B_0 &= \alpha_1 h - \beta_1 \\ C_0 &= \beta_1 \end{aligned} \tag{7}$$

$$\begin{aligned} A_n &= -\beta_2 \\ B_n &= \alpha_2 h + \beta_2 \end{aligned} \tag{8}$$

Тесты

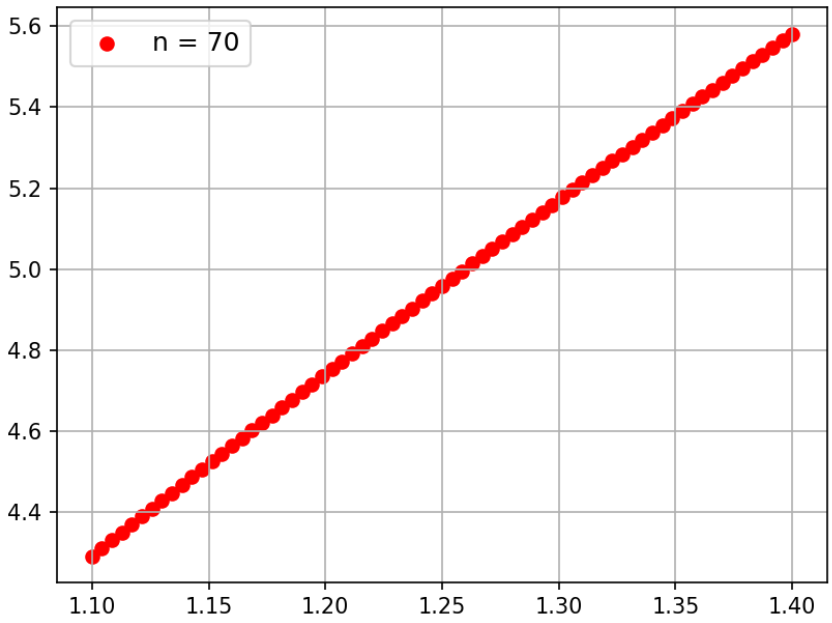
Все тесты были проверены с помощью вычислительных алгоритмов, предоставляемых сайтом `https://www.wolframalpha.com/`

Тест 1

Краевая задача (в соответствии с вариантом):

$$\begin{cases} y'' - y' + 2yx = x + 0.4 \\ y(1.1) - 0.5y'(1.1) = 2 \\ y(1.1) - 0.5y'(1.1) = 2 \end{cases}$$

Точное решение: Не предоставлено. Построим график численного решения



Из аналитических рассуждений можно сделать вывод, что было получено хорошее приближение.

Тест 2

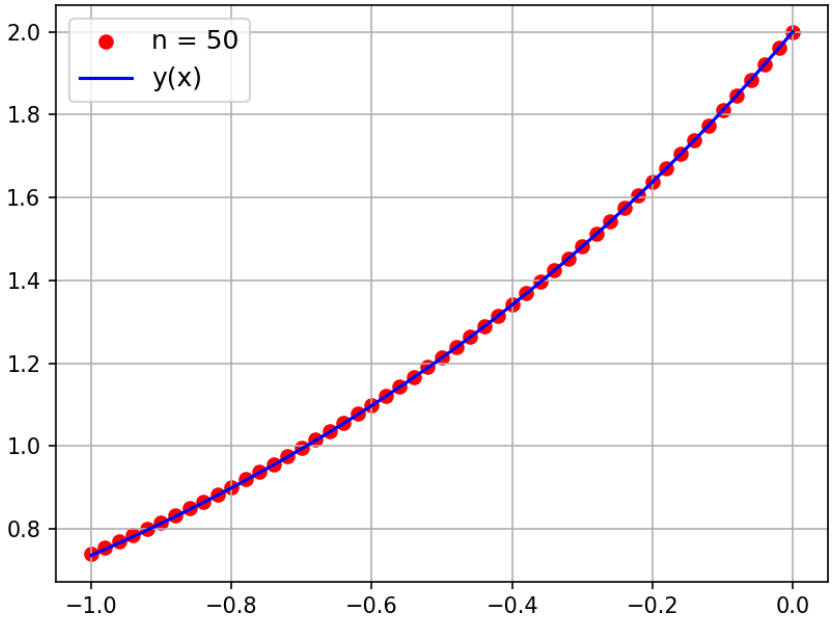
Краевая задача:

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

Точное решение:

$$y = 2e^x$$

Построим графики численного решения и точного аналитического решения (точки - численное решение, непрерывная линия - аналитическое):



Можно увидеть, что получено хорошее приближение.

Тест 3

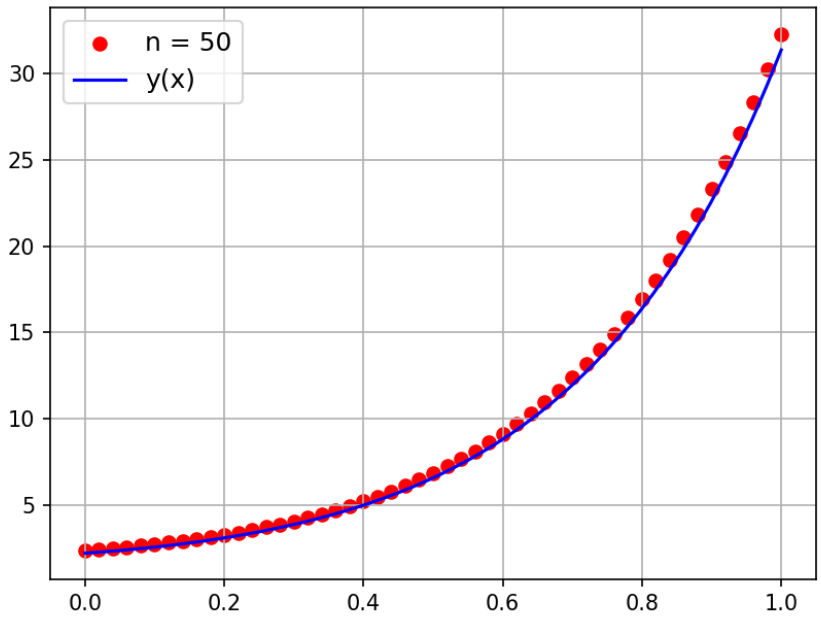
Краевая задача:

$$\begin{cases} y'' - 2y' - 3y = e^{4x} \\ y'(0) - y(0) = 0.6 \\ y'(1) + y(1) = 4e^3 + e^4 \end{cases}$$

Точное решение:

$$y = e^{-x} + e^{3x} + 0.2e^{4x}$$

Построим графики численного решения и точного аналитического решения (точки - численное решение, непрерывная линия - аналитическое):



Можно увидеть, что получено хорошее приближение.

Выводы

В ходе практической работы был изучен и реализован метод прогонки для решения краевой задачи для обыкновенного дифференциального уравнения второго порядка. Данный метод показал хорошую точность решения данного класса задач. Для увеличения точности решения можно увеличивать количество шагов алгоритма, так как это увеличит количество точек в решении.

Код программы

Для решения поставленных задач был выбран язык программирования Python 3.10

Код для вспомогательных классов

Так как в ходе практической работы приходилось работать с большим количеством различных объектов, объединенных названием "дифференциальные уравнения то было принято решение спроектировать специальные классы для унифицированного представления. Приведем их реализацию.

Файл **tasks.py** из библиотеки **equations**:

```
class DiffEqType2:
    def __init__(self, p, q, f, cc1, cc2, left, right):
        self.p = p
        self.q = q
        self.f = f
        self.cc1 = cc1
        self.cc2 = cc2
        self.left = left
        self.right = right

class DiffEqType1:
    def __init__(self, f, cc, left, right):
        self.f = f
        self.cc = cc
        self.left = left
        self.right = right

class DiffEqSystem:
    def __init__(self, f1, f2, cc, left, right):
        self.f1 = f1
        self.f2 = f2
        self.left = left
        self.right = right
        self.cc = cc
```

Файл **tests.py** из библиотеки **equations**:

```
from equations import tasks

class TestDiffEqType2(tasks.DiffEqType2):
    def __init__(self, p, q, f, cc1, cc2, solution, left, right):
        super().__init__(p, q, f, cc1, cc2, left, right)
        self.solution = solution
```

```
class TestDiffEqType1(tasks.DiffEqType1):
    def __init__(self, f, cc, left, right, solution):
        super().__init__(f, cc, left, right)
        self.solution = solution

class TestDiffEqSystem(tasks.DiffEqSystem):
    def __init__(self, f1, f2, cc, left, right, u, v):
        super().__init__(f1, f2, cc, left, right)
        self.u = u
        self.v = v
```

Код для первого подзадания

В этом разделе содержатся реализации методов Рунге-Кутта второго и четвертого порядков точности, а также необходимый набор функций для взаимодействия с пользователем. Все дифференциальные уравнения задаются пользователем в описанных выше классах в файлах **tests.py** и **tasks.py** из пользовательской библиотеки **equtions**

```
import matplotlib.pyplot as plt
import numpy as np
from equations import tests, tasks


def rk2(f, length, iterations, x0, y0):
    x_lst, y_lst = list(), list()
    x, y = x0, y0

    x_lst.append(x)
    y_lst.append(y)

    h = length / iterations
    for i in range(iterations):
        y = y + (f(x, y) + f(x + h, y + h * f(x, y))) * h / 2
        x += h
        x_lst.append(x)
        y_lst.append(y)
    return x_lst, y_lst


def rk4(f, length, iterations: int, x0: float, y0: float):
    x_lst, y_lst = list(), list()
    x, y = x0, y0

    x_lst.append(x)
    y_lst.append(y)

    h = length / iterations
    for i in range(iterations):
        k1 = f(x, y)
        k2 = f(x + h / 2, y + h / 2 * k1)
        k3 = f(x + h / 2, y + h / 2 * k2)
        k4 = f(x + h, y + h * k3)

        y = y + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
        x += h
        x_lst.append(x)
        y_lst.append(y)
```

```
    return x_lst, y_lst


def system_rk2(eq, length, iterations, x0, y0):
    h = length / iterations
    size = len(eq)
    x_lst, y_lst = np.zeros((iterations,)), np.zeros((2, iterations))
    x, y = x0, y0.copy()

    k1 = np.zeros((size,))
    k2 = np.zeros((size,))
    for i in range(iterations):
        for j in range(size):
            k1[j] = eq[j](x, y[0], y[1])
            k2[j] = eq[j](x + h, y[0] + h * k1[0], y[1] + h * k1[1])
            y[j]= y[j] + (k1[j] + k2[j]) * h / 2
            y_lst[j][i] = y[j]

        x += h
        x_lst[i] = x
    return x_lst, y_lst[0], y_lst[1]


def system_rk4(eq, length, iterations, x0, y0):
    size = len(eq)
    x_lst, y_lst = np.zeros((iterations, )), np.zeros((2, iterations))
    x, y = x0, y0.copy()

    h = length / iterations
    k1 = np.zeros((size,))
    k2 = np.zeros((size,))
    k3 = np.zeros((size,))
    k4 = np.zeros((size,))
    for i in range(iterations):
        for j in range(size):
            k1[j] = eq[j](x, y[0], y[1])
            k2[j] = eq[j](x + h / 2, y[0] + h / 2 * k1[0], \
            y[1] + h / 2 * k1[1])
            k3[j] = eq[j](x + h / 2, y[0] + h / 2 * k2[0], \
            y[1] + h / 2 * k2[1])
            k4[j] = eq[j](x + h, y[0] + h * k3[0], y[1] + h * k3[1])

            y[j] = y[j] + h / 6 * (k1[j] + 2 * k2[j] + 2 * k3[j] + k4[j])
            y_lst[j][i] = y[j]

        x += h
```

```
        x_lst[i] = x
    return x_lst, y_lst[0], y_lst[1]

def test(eq: tests.TestDiffEqType1, iterations=30):
    fig, ax = plt.subplots()

    test_x = np.linspace(eq.left, eq.right)
    test_y = [eq.solution(test_x[i]) for i in range(test_x.size)]
    ax.plot(test_x, test_y, c='blue', label='y(x)')
    ax.set_xlabel(r'x', fontsize=12, loc="right")
    ax.set_ylabel(r'y(x)', fontsize=12, loc="top", rotation=0)
    x, y = rk2(eq.f, eq.right - eq.left, iterations * 2, eq.left, eq.cc)
    ax.scatter(x, y, c='green', label='RK2')
    x, y = rk4(eq.f, eq.right - eq.left, iterations, eq.left, eq.cc)
    ax.scatter(x, y, c='red', label='RK4')

    ax.set_title('RK2 & RK4 methods')
    ax.legend()
    ax.grid()
    fig.canvas.manager.set_window_title("Results")
    fig.tight_layout()
    plt.show()


def test_system(eq: tests.TestDiffEqSystem, iterations=30):
    diff_eq = [eq.f1, eq.f2]
    fig, ax = plt.subplots()

    test_t = np.linspace(0, 3)
    test_u = [eq.u(t) for t in test_t]
    test_v = [eq.v(t) for t in test_t]
    ax.plot(test_t, test_u, c='brown', label='u(x)')
    ax.plot(test_t, test_v, c='purple', label='v(x)')
    ax.set_xlabel(r'x', fontsize=12, loc="right")
    ax.set_ylabel(r'y(x)', fontsize=12, loc="top", rotation=0)
    x, y1, y2 = system_rk2(
    diff_eq, eq.right - eq.left, iterations, eq.left, eq.cc)
    ax.scatter(x, y1, c='green', label='u_RK2')
    ax.scatter(x, y2, c='blue', label='v_RK2')
    x, y1, y2 = system_rk4(
    diff_eq, eq.right - eq.left, iterations, eq.left, eq.cc)
    ax.scatter(x, y1, c='red', label='u_RK4')
    ax.scatter(x, y2, c='orange', label='v_RK4')

    ax.set_title('RK2 & RK4 methods')
    ax.legend(loc="lower left")
```

```
    ax.grid()
    fig.canvas.manager.set_window_title("Results for system")
    fig.tight_layout()
    plt.show()

def solve_system(eq: tasks.DiffEqSystem, iterations=30):
    diff_eq = [eq.f1, eq.f2]
    fig, ax = plt.subplots()

    ax.set_xlabel(r'x', fontsize=12, loc="right")
    ax.set_ylabel(r'y(x)', fontsize=12, loc="top", rotation=0)
    x, y1, y2 = system_rk2(
    diff_eq, eq.right - eq.left, iterations, eq.left, eq.cc)
    ax.scatter(x, y1, c='green', label='u_RK2')
    ax.scatter(x, y2, c='blue', label='v_RK2')
    x, y1, y2 = system_rk4(
    diff_eq, eq.right - eq.left, iterations, eq.left, eq.cc)
    ax.scatter(x, y1, c='red', label='u_RK4')
    ax.scatter(x, y2, c='orange', label='v_RK4')

    ax.set_title('RK2 & RK4 methods')
    ax.legend(loc="lower left")
    ax.grid()
    fig.canvas.manager.set_window_title("Results for system")
    fig.tight_layout()
    plt.show()

if __name__ == "__main__":
    test(tests.type1_test2)
    test_system(tasks.system_eq2, 50)
```

Код для второго подзадания

В этом разделе содержатся реализации методов конечных разностей и прогонки, а также необходимый набор функций для взаимодействия с пользователем. Все дифференциальные уравнения задаются пользователем в описанных выше классах в файлах **tests.py** и **tasks.py** из пользовательской библиотеки **equtions**

```
import matplotlib.pyplot as plt
import numpy as np
from equations import tests, tasks

def finite_diff(p, q, f, left, right, certain_cond1, certain_cond2, n=20):
    h = (right - left) / n
    A = np.zeros((n + 1,))
    B = np.zeros((n + 1,))
    C = np.zeros((n + 1,))
    F = np.zeros((n + 1,))
    y = np.zeros((n + 1,))
    aa = np.zeros((n + 1,))
    bb = np.zeros((n + 1,))

    x = np.array([left + h * i for i in range(n + 1)])
    for i in range(n):
        A[i] = 1 / h ** 2 - p(x[i]) / (2 * h)
        C[i] = 1 / h ** 2 + p(x[i]) / (2 * h)
        B[i] = - 2 / h ** 2 + q(x[i])
        F[i] = f(x[i])

    B[0] = certain_cond1[1] * h - certain_cond1[0]
    C[0] = certain_cond1[0]
    F[0] = certain_cond1[2] * h

    B[n] = certain_cond2[1] * h + certain_cond2[0]
    A[n] = -certain_cond2[0]
    F[n] = certain_cond2[2] * h

    aa[0] = -C[0] / B[0]
    bb[0] = F[0] / B[0]
    for i in range(1, n + 1):
        aa[i] = -C[i] / (A[i] * aa[i - 1] + B[i])
        bb[i] = (F[i] - A[i] * bb[i - 1]) / (A[i] * aa[i - 1] + B[i])

    y[n] = (F[n] - bb[n - 1] * A[n]) / (B[n] + aa[n - 1] * A[n])
    for i in range(n, 0, -1):
        y[i - 1] = aa[i - 1] * y[i] + bb[i - 1]
    return x, y
```

```
def test(eq: tests.TestDiffEqType2, n: int) -> None:
    fig, ax = plt.subplots()

    x, y = finite_diff(
    eq.p, eq.q, eq.f, eq.left, eq.right, eq.cc1, eq.cc2, n)
    plt.scatter(x, y, c='red', label='n = {}'.format(n))

    test_x = np.linspace(eq.left, eq.right)
    test_y = [eq.solution(test_x[i]) for i in range(test_x.size)]
    ax.plot(test_x, test_y, c='blue', label='y(x)')
    plt.legend(fontsize=12)
    plt.grid()
    plt.show()


def solve(eq: tasks.DiffEqType2, n: int) -> None:
    fig, ax = plt.subplots()

    x, y = finite_diff(
    eq.p, eq.q, eq.f, eq.left, eq.right, eq.cc1, eq.cc2, n)
    ax.scatter(x, y, c='red', label='n = {}'.format(n))

    plt.legend(fontsize=12)
    plt.grid()
    plt.show()


if __name__ == "__main__":
    ans = input("Выберите режим работы:\
    \n- \(1) тестирование\
    \n- (2) показать решения\n > ")
    n_loc = input("Введите n; (Enter) для значения по умолчанию\n > ")
    if n_loc:
        n_loc = int(n_loc)
    else:
        n_loc = 50
    if ans == '1':
        print("Для завершения программы закройте выплывшее окно")
        test(tests.type2_test, n_loc)
    elif ans == '2':
        print("Для завершения программы закройте выплывшее окно")
        solve(tasks.type2_eq, n_loc)
    else:
        print("Ошибка ввода")
        exit(0)
```

Список литературы

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

[2] Самарский А.А., Гулин А.В. Численные методы, 1989