**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 201 учебной группы факультета ВМК МГУ

Бутылкина Андрея Сергеевича

гор. Москва

2022 г.

## Содержание

# Подвариант 1

## Постановка задачи

Рассматривается обыкновенное дифференциальное уравнение первого порядка, разрешенное относительно производной с дополнительным начальным условием, заданным в точке $x = x_0$ и имеющее вид:

$$\begin{cases} \frac{dy}{dx} = f(x, y),\ x_0 < x \\ y(x_0) = y_0 \end{cases}$$

Или рассматривается система обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производных неизвестных функций, с дополнительными начальными условиями, заданными в точке $x = x_0$ и имеет вид:

$$\begin{cases} \frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_n),\ x_0 < x \\ \cdots \\ \frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n),\ x_0 < x \\ y_1(x_0) = y_{01} \\ \cdots \\ y_n(x_0) = y_{0n} \end{cases}$$

Предполагается, что параметры задач таковы, что выполнены условия существования и единственности решения.

## Цели и задачи практической работы

- Решить 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), представляющее фактически некоторую рекуррентную формулу, просчитать численно;
- Найти численное решение задачи и построить его график;
- Найденное численное решение сравнить с точным решением дифференциального уравнения;

## Описание алгоритма решения

Рассмотрим ОДУ первого порядка, разрешенное относительно производной с дополнительным начальным условием, заданным в точке $x = x_0$ и имеющее вид:

$$\begin{cases} \frac{dy}{dx} = f(x, y),\ x_0 < x \\ y(x_0) = y_0 \end{cases}$$

Найдем численное решение задачи на отрезке $[x_0,\ x_0 + l]$.
Воспользуемся алгоритмом Рунге-Кутта второго порядка точности и четвёртого порядка точности.
Определим шаг $h = \frac{l}{n}$, где $n$— количество точек, в которых мы хотим получить значение функции. Равномерная сетка при этом: $x_{i+1} = x_i + h$. Для нее имеем следующее удобное для расчетов рекуррентное соотношение:

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

Это метод Рунге-Кутта второго порядка точности. Для метода Рунге-Кутта четвёртого порядка точности имеем следующее рекуррентное соотношение:

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

где

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

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

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

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

Рассмотрим систему ОДУ первого порядка, разрешенных относительно производных неизвестных функций, с дополнительными начальными условиями, заданными в точке $x = x_0$ и имеющую вид:

$$\begin{cases} \frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_m),\ x_0 < x \\ \frac{dy_2}{dx} = f_2(x, y_1, y_2, \ldots, y_m),\ x_0 < x \\ \cdots \\ \frac{dy_m}{dx} = f_n(x, y_1, y_2, \ldots, y_m),\ x_0 < x \\ y_1(x_0) = y_{01} \\ y_2(x_0) = y_{02} \\ \cdots \\ y_m(x_0) = y_{0m} \end{cases}$$

Для этой задачи метод Рунге-Кутта второго порядка точности переписывается следующем рекуррентным соотношением:

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

Для метода Рунге-Кутта четвёртого порядка имеем:

$$y_{j,i+1} = y_{j,i} + h\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

где

$$k_1 = f_j(x_i, y_{j,i})$$

$$k_2 = f_j(x_i + \frac{h}{2}, y_{j,i} + \frac{h}{2}k_1)$$

$$k_3 = f_j(x_i + \frac{h}{2}, y_{j,i} + \frac{h}{2}k_2)$$

$$k_4 = f_j(x_i + h, y_{j,i} + hk_3)$$

$j$ — номер решения, $i$ — номер итерации.

# Тесты

## Тест 1

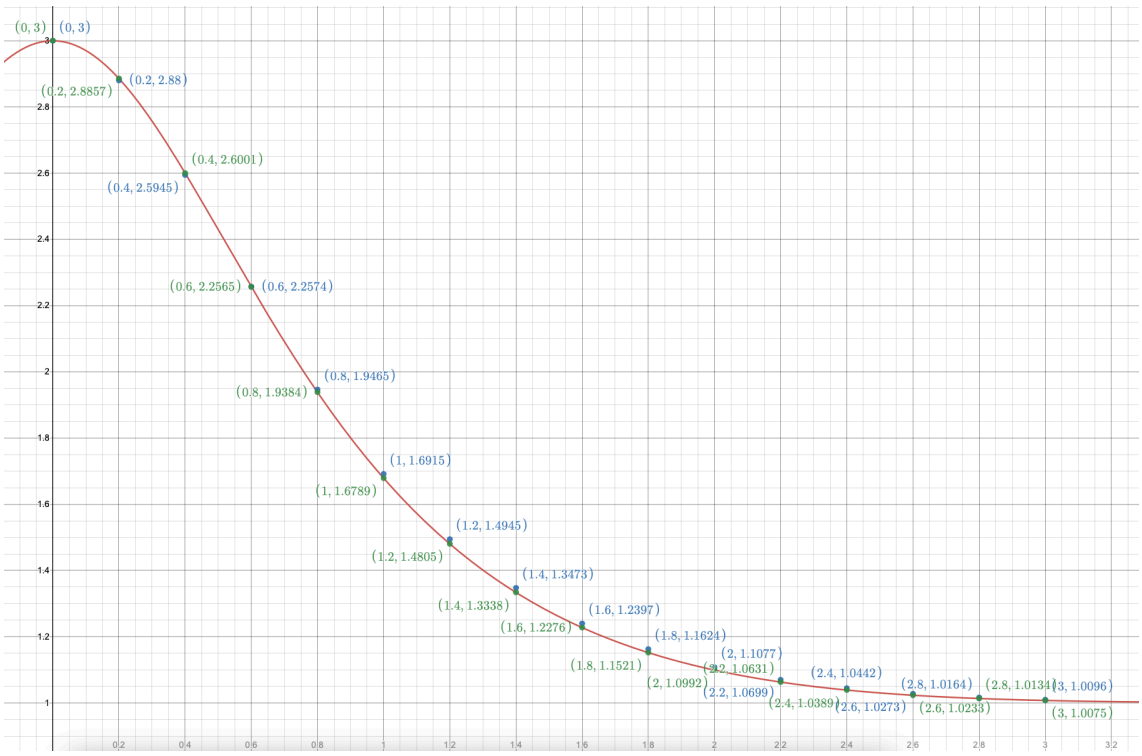
Задача Коши:

$$\begin{cases} \frac{dy}{dx} = (y - y^2)x \\ y(0) = 3 \end{cases}$$

Точное аналитическое решение:

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

Рассмотрим эту задачу на отрезке [0, 3]:




Красная линия — точное аналитическое решение, синие точки — решение второго порядка точности, зелёные точки — четвёртого порядка точности. Значения искомой функции, в искомых точках
(0,3), (0.2,2.28857), (0.4,2.6002), (0.6,2.2566), (0.8,1.9384), (1,1.6788), (1.2,1.4804), (1.4,1.3337), (1.6,1.2275), (1.8,1.152), (2,1.0992), (2.2,1.063), (2.4,1.0389), (2.6,1.0232), (2.8,1.0134), (3,1.0075)
Оба метода дают довольно точную аппроксимацию данной функции, но решения четвёртого порядка несколько точнее второго порядка.

# Тест 2

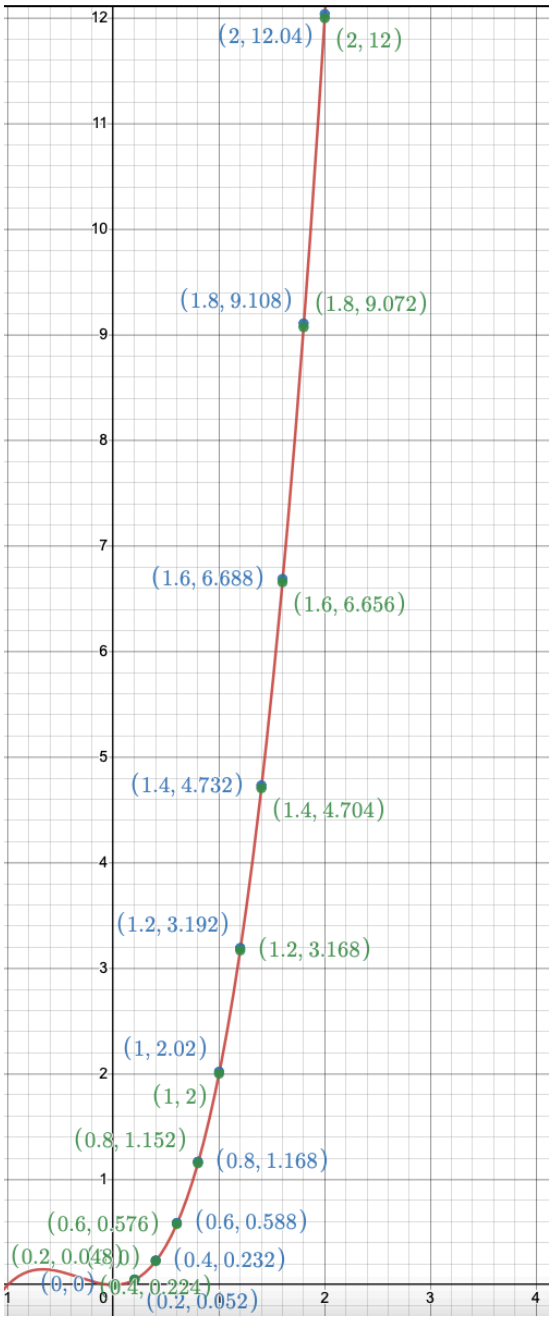
Задача Коши:

$$\begin{cases} \frac{dy}{dx} = 2x + 3x^2 \\ y(0) = 0 \end{cases}$$

Точное аналитическое решение:

$$y = x^2 + x^3$$

Рассмотрим эту задачу на отрезке [0, 2]:



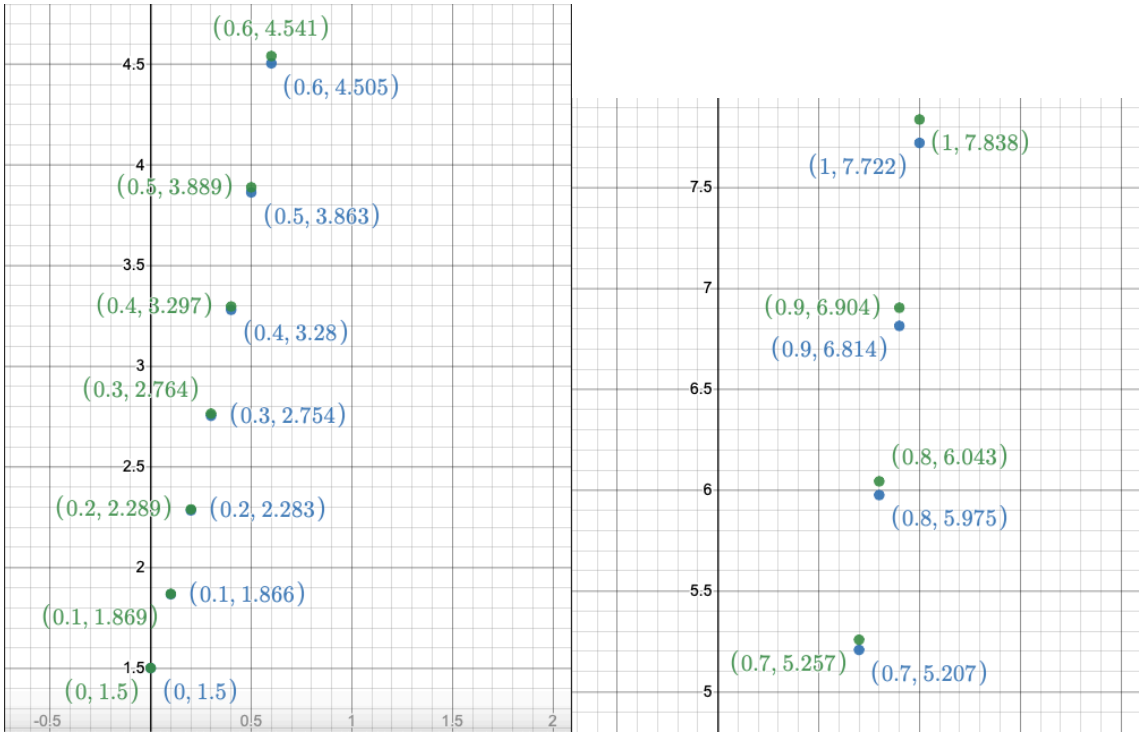
Красная линия — точное аналитическое решение, синие точки — решение второго порядка точности, зелёные точки — четвёртого порядка точности. Значения искомой функции, в искомых точках (0,0), (0.2,0.48), (0.4,0.224), (0.6,0.576), (0.8,1.152), (1,2), (1.2,3.168), (1.4,4.704), (1.6,6.656), (1.8,9.072), (2,12)
Метод Рунге-Кутта второго порядка точности даёт довольно точную аппроксимацию данной функции. Метод Рунге-Кутта четвертого порядка точности в данной задаче даёт численное решение, совпадающее с аналитическим.

## Тест 3

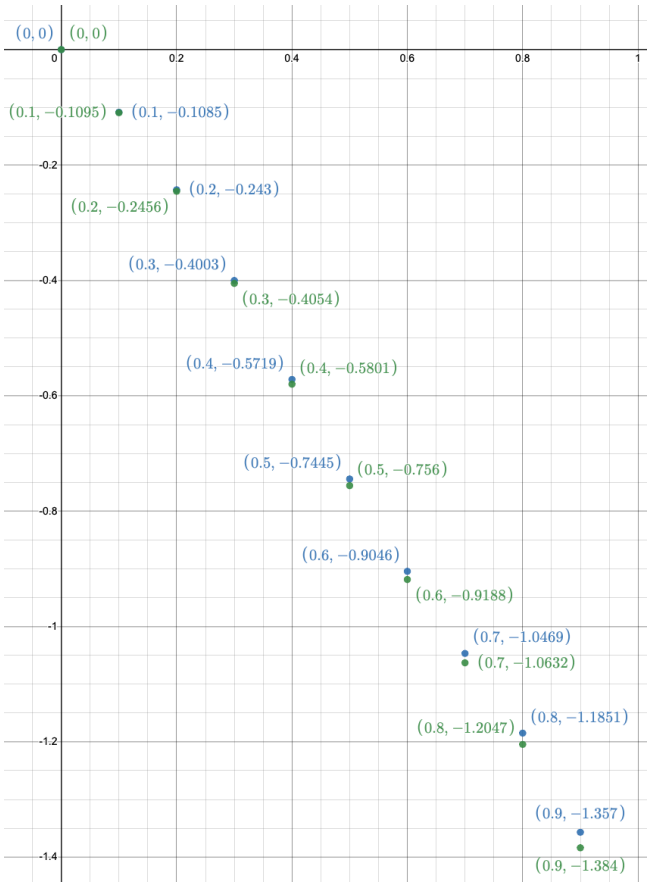
Задача Коши:

$$\begin{cases} \frac{dy_1}{dx} = x + y_1 - y_2^2 + 2 \\ \frac{dy_2}{dx} = sin(x - y_1) + 2.1y_2 \\ y_1(0) = 1.5 \\ y_2(0) = 0 \end{cases}$$

Рассмотрим эту задачу на отрезке [0, 2].
Приближение для $y_1$:



Приближения для $y_2$:




Синие точки — решение второго порядка точности, зелёные точки — четвёртого порядка точности. На этом тесте более видна разница между порядками точности, четвертый порядок точности даёт лучший результат. Несмотря на это, оба метода дают довольно точную аппроксимацию функций.

# Выводы

После изучения и практического применения методов Рунге-Кутта второго и четвертого порядка точности для задач Коши для ОДУ первого порядка, разрешенное относительно производной и системы ОДУ первого порядка, разрешенной относительно производных неизвестных функций; проведения тестов, можно сделать вывод, что оба метода дают довольно точную аппроксимацию функций, но метод Рунге-Кутта четвертого порядка точности оказывался более точным, чем второго порядка. Однако и реализация у четвёртого порядка сложнее, количество вычислений тоже увеличивается. Это может понизить скорость работы в случае сложных функций. Стоит добавить, что повысить точность можно с помощью увеличения количества искомых точек.

# Код программы

Использованный язык — Python

```
from math import *
from copy import *

def function_pull (uk, x, y):              # задача уравнений
    if uk == 1:
        return (y - y ** 2) * x      # уравнение по варианту
    elif uk == 2:
        return 2 * x + 3 * x ** 2              # свой пример
    else:
        print("error")

def function_pull_sys (uk, x, y):   # уравнения для системы
    if uk == 1:
        return x + y[0] - y[1] ** 2 + 2
    elif uk == 2:
        return sin(x - y[0]) + 2.1 * y[1]
    else:
        print("error")

def runge_kutt_2 (uk, len_ab, n, x, y): # метод Рунге-Кутта
    h = len_ab / n                # второго порядка точности
    h_p = h / 2                                      # для ОДУ
    ans = []
    ans.append([x, y])
    for i in range(n):
        f_prev = function_pull(uk, x, y)
        f_now = function_pull(uk, x + h, y + h * f_prev)
        y = y + h_p * (f_prev + f_now)
        x = x + h
```

```
        ans.append([x, y])
    return ans

def runge_kutt_4 (uk, len_ab, n, x, y): # метод Рунге-Кутта
    h = len_ab / n            # четвертого порядка точности
    h_p = h / 2                                  # для ОДУ
    ans = []
    ans.append([x, y])
    for i in range(n):
        k1 = function_pull(uk, x, y)
        k2 = function_pull(uk, x + h_p, y + h_p * k1)
        k3 = function_pull(uk, x + h_p, y + h_p * k2)
        k4 = function_pull(uk, x + h, y + h * k3)
        y = y + h_p * (k1 + 2 * k2 + 2 * k3 + k4) / 3
        x = x + h
        ans.append([x, y])
    return ans

def runge_kutt_2_sys (uk_ar, len_ab, n, x, y):      # метод
    h = len_ab / n            # Рунге-Кутта второго порядка
    h_p = h / 2                  # точности для системы ОДУ
    ans = []
    ans.append([x, deepcopy(y)])
    kol_func = len(uk_ar)
    for i in range(n):
        y_prev = deepcopy(y)
        for j in range(kol_func):
            f_prev = function_pull_sys(uk_ar[j], x,
[y_prev[k] for k in range(kol_func)])
            f_now = function_pull_sys(uk_ar[j], x + h,
[y_prev[k] + h * f_prev for k in range(kol_func)])
            y[j] = y[j] + h_p * (f_prev + f_now)
        x = x + h
```

```
        ans.append([x, deepcopy(y)])
    return ans


def runge_kutt_4_sys (uk_ar, len_ab, n, x, y):      # метод
    h = len_ab / n                  # Рунге-Кутта четвертого
    h_p = h / 2                   # порядка точности для ОДУ
    ans = []
    ans.append([x, deepcopy(y)])
    kol_func = len(uk_ar)
    for i in range(n):
        y_prev = deepcopy(y)
        for j in range(kol_func):
            k1 = function_pull_sys(uk_ar[j], x, [y_prev[k]
for k in range(kol_func)])
            k2 = function_pull_sys(uk_ar[j], x + h_p,
[y_prev[k] + h_p * k1 for k in range(kol_func)])
            k3 = function_pull_sys(uk_ar[j], x + h_p,
[y_prev[k] + h_p * k2 for k in range(kol_func)])
            k4 = function_pull_sys(uk_ar[j], x + h,
[y_prev[k] + h * k3 for k in range(kol_func)])
            y[j] = y[j] + h_p * (k1 + 2 * k2 + 2 * k3 + k4)
/ 3
        x = x + h
        ans.append([x, deepcopy(y)])
    return ans


# основная программа
print("ODE solution according to the variant:")
print("~Second order~")
print(*runge_kutt_2(1, 3, 15, 0, 3), sep='\n')
print()
print("~Fourth order~")
print(*runge_kutt_4(1, 3, 15, 0, 3), sep='\n')
print()
```

```
print("ODE solution own example:")

print("~Second order~")

print(*runge_kutt_2(2, 2, 10, 0, 0), sep='\n')

print()

print("~Fourth order~")

print(*runge_kutt_4(2, 2, 10, 0, 0), sep='\n')

print()


print("Solution of the ODE system according to the
variant:")

print("~Second order~")

print(*runge_kutt_2_sys([1, 2], 1, 10, 0, [1.5, 0]),
sep='\n')

print()

print("~Fourth order~")

print(*runge_kutt_4_sys([1, 2], 1, 10, 0, [1.5, 0]),
sep='\n')
```

# Подвариант 2

## Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида:

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

с дополнительными условиями в граничных точках:

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

## Цели и задачи практической работы

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

- Найти разностное решение задачи и построить его график;

- Найденное разностное решение сравнить с точным решением дифференциального уравнения.

## Описание алгоритма решения

Рассмотрим ОДУ второго порядка вида:

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

с дополнительными условиями в граничных точках:

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

Найдем численное решение задачи на отрезке $[a, b]$.
Воспользуемся следующим алгоритмом.
Определим шаг $h = \dfrac{b-a}{n}$, где $n$— количество точек, в которых мы хотим получить значение функции. Равномерная сетка при этом: $x_{i+1} = x_i + h$, $x_0 = a$. Имеем следующие конечно разностные сотношения:

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

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

Тогда исходное ОДУ и дополнительные условия можно переписать:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p(x) \cdot \frac{y_{i+1} - y_{i-1}}{2h} + q(x) \cdot y = -f(x),$$

$$\begin{cases} \sigma_1 \frac{y_0 + y_1}{2} + \gamma_1 \frac{y_1 - y_0}{2h} = \delta_1, \\ \sigma_2 \frac{y_n + y_{n+1}}{2} + \gamma_2 \frac{y_{n+1} - y_n}{2h} = \delta_2 \end{cases}$$

В полученных уравнениях перегруппируем слагаемые, чтобы они имели вид:

$$\begin{cases} B_0 y_0 + C_0 y_1 = D_0 \\ A_i y_{i-1} + B_i y_i + C_i y_{i+1} = D_i \\ A_{n+1} y_n + B_{n+1} y_{n+1} = D_{n+1} \end{cases}$$

$i = 1...n$

Таким образом, мы получили СЛАУ с трёхдиагональной матрицей коэффициентов. Для нахождения решения воспользуемся методом прогонки. Имеем следующее рекуррентное соотношение:

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}$$

Подставляя в нашу СЛАУ и требуя выполнения равенств независимо от $y_i, y_{i+1}$, получаем:

$$\begin{cases} A_i \alpha_i \alpha_{i+1} + C_i \alpha_{i+1} + B_i = 0 \\ A_i \alpha_i \beta_{i+1} + A_i \beta_i + C_i \beta_{i+1} - D_i = 0 \end{cases}$$

Из этой системы следуют следующие формулы:

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

Из дополнительных условий в граничных точках получаем:

$$\begin{cases} \alpha_0 = -\frac{\gamma_1}{\sigma_1 h - \gamma_1} \\ \beta_0 = \frac{\delta_1}{\sigma_1 - \frac{\gamma_1}{h}} \end{cases}$$

Теперь можем итерационно получить все коэффициенты $\alpha_i, \beta_i$. Получив все значения для $\alpha_i, \beta_i$, можем итерационно получить $y_i$, с помощью рекуррентного соотношения описанного выше. Для начала итерации имеем:

$$y_n = \frac{\delta_2 h + \gamma_2 \beta_{n-1}}{\sigma_2 h + \gamma_2 (1 - \alpha_{n-1})}$$

Формулы для расчета $A_i$, $B_i$, $C_i$ :

$$A_i = h^{-2} - \frac{p(x_i)}{2h}$$

$$B_i = h^{-2} + \frac{p(x_i)}{2h}$$

$$C_i = -2h^{-2} + q(x_i)$$

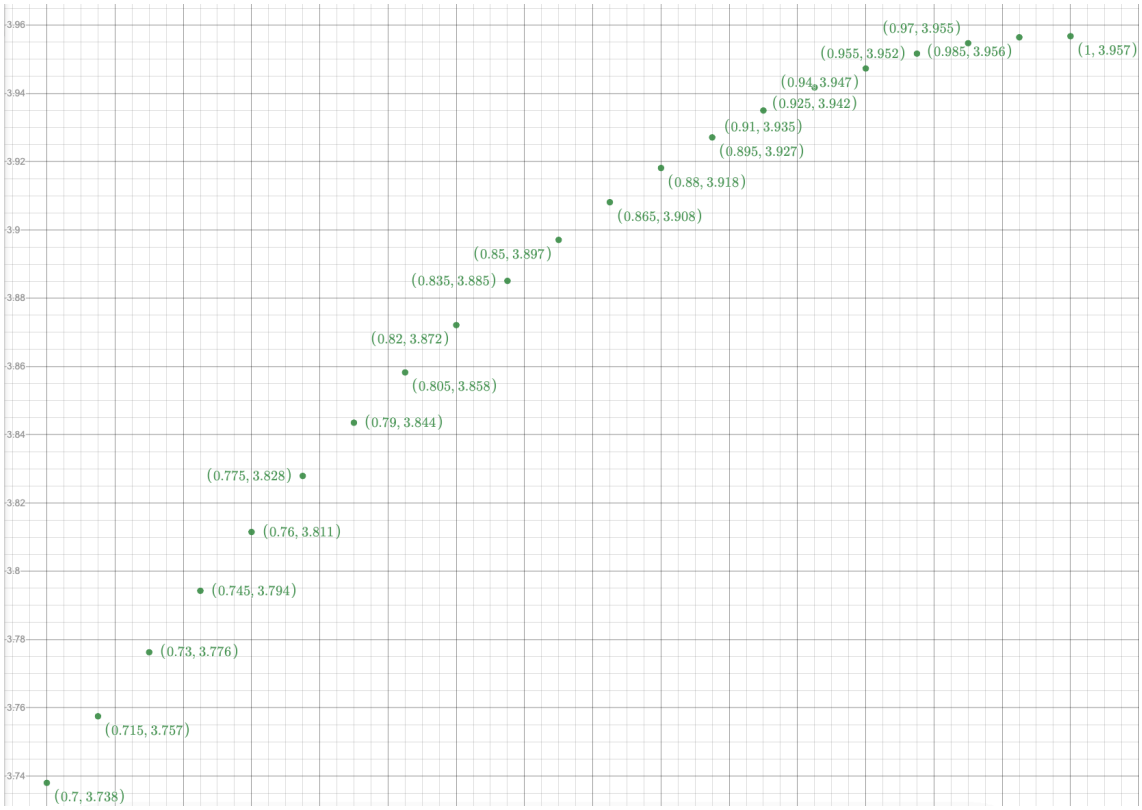
# Тесты

## Тест 1

Краевая задача:

$$\begin{cases} y'' - 3xy' + 2y = 1.5 \\ y'(0.7) = 1.3 \\ 0.5y(1) + y'(1) = 2 \end{cases}$$

График численного решения:




Численное решение представлено на графике зелёными точками и соответствующими координатами.

## Тест 2

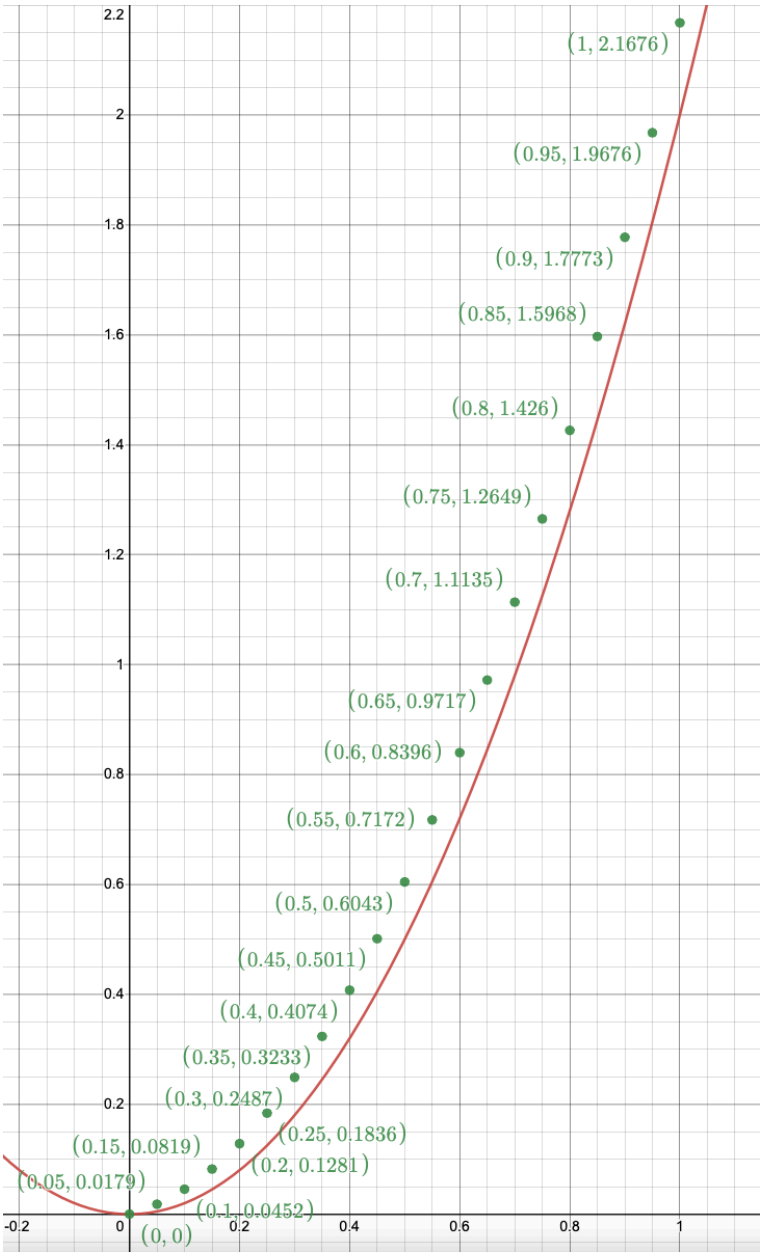
Краевая задача:

$$\begin{cases} y'' + y' = 4x + 4 \\ y(0) = 0 \\ y'(1) = 4 \end{cases}$$

Точное аналитическое решение:

$$y = 2x^2$$

График:




Численное решение представлено на графике зелёными точками и соответствующими координатами, точное аналитическое решение — красная линия. Как можно видеть, получили довольно хорошее приближение. В искомых точках точное решение следующее:
(0, 0), (0.05, 0.005), (0.1, 0.02), (0.15, 0.045), (0.2, 0.08), (0.25, 0.125), (0.3, 0.18), (0.35, 0.245), (0.4, 0.32), (0.45, 0.405), (0.5, 0.5), (0.55, 0.605), (0.6, 0.72), (0.65, 0.845), (0.7, 0.98), (0.75, 1.125), (0.8, 1.28), (0.85, 1.445), (0.9, 1.62), (0.95, 1.805), (1, 2) .

## Тест 3

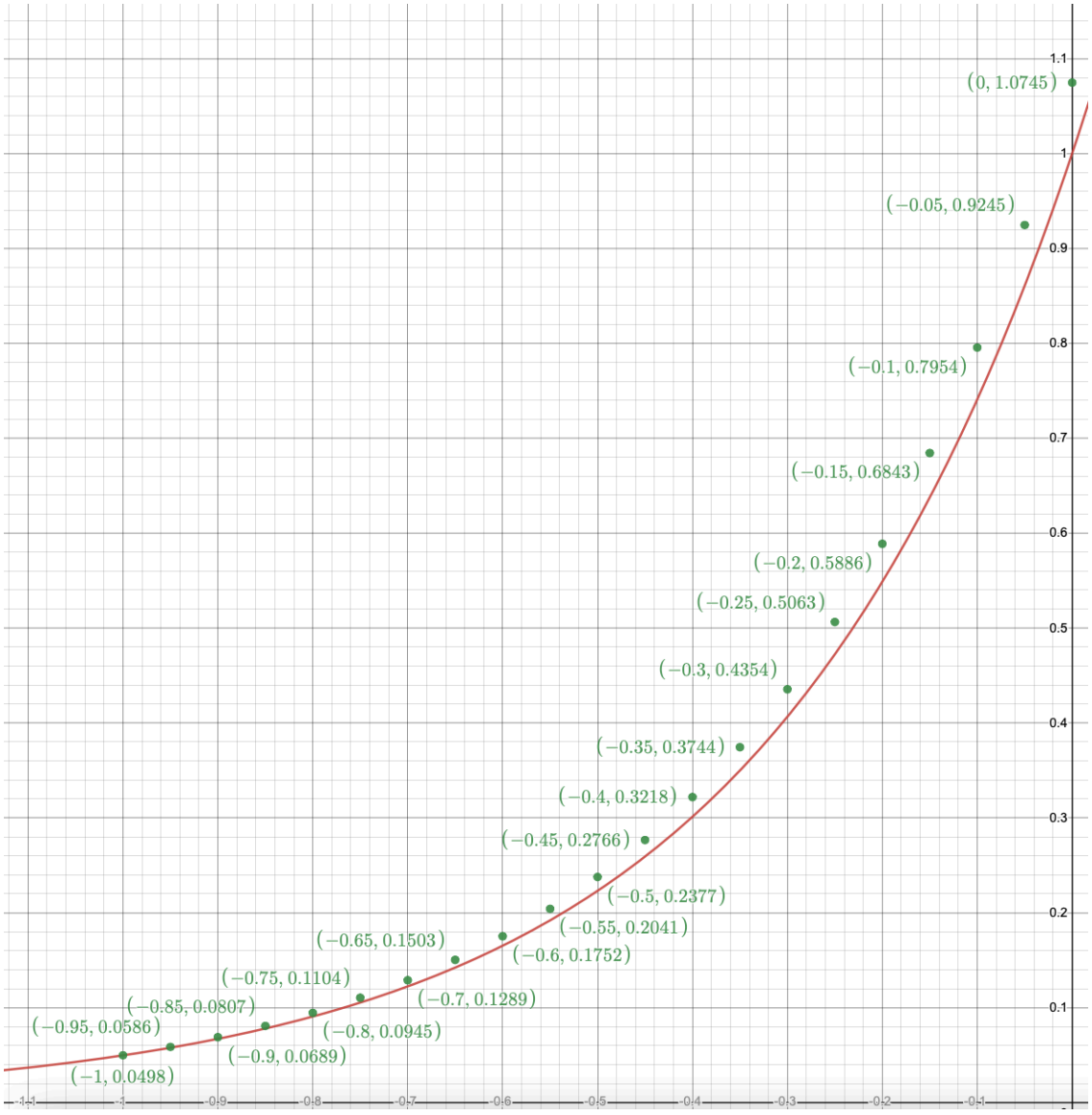
Краевая задача:

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

Точное аналитическое решение:

$$y = e^{3x}$$

График:



Численное решение представлено на графике зелёными точками и соответствующими координатами, точное аналитическое решение — красная линия. Как можно видеть, получили довольно хорошее приближение. В искомых точках точное решение следующее: $(-1, 0.04978)$, $(-0.95, 0.0578)$, $(-0.9, 0.0672)$, $(-0.85, 0.0780)$, $(-0.8, 0.0907)$, $(-0.75, 0.1053)$, $(-0.7, 0.1224)$, $(-0.65, 0.1422)$, $(-0.6, 0.1652)$, $(-0.55, 0.1920)$, $(-0.5, 0.2231)$, $(-0.45, 0.2592)$, $(-0.4, 0.3011)$, $(-0.35, 0.3499)$, $(-0.4, 0.4065)$, $(-0.25, 0.4723)$, $(-0.3, 0.5488)$, $(-0.15, 0.6376)$, $(-0.1, 0.7408)$, $(-0.05, 0.8607)$, $(0, 1)$.

## Выводы

После изучения и практического применения метода прогонки для решения краевой задачи для ОДУ второго порядка; проведения тестов, можно сделать вывод, что метод даёт довольно точную аппроксимацию функций. Стоит добавить, что повысить точность можно с помощью увеличения количества искомых точек.

# Код программы

Использованный язык — Python

```
from math import *

def p1(x):
    return -3 * x

def q1(x):
    return 2

def f1(x):
    return 1.5

def p2(x):
    return 1

def q2(x):
    return 0

def f2(x):
    return 4 * x + 4

def p3(x):
    return -2

def q3(x):
    return -3

def f3(x):
    return 0
```

```
def sweep_bound(p, q, f, a, b, sigma1, gamma1, delta1,
sigma2, gamma2, delta2, n):
    h = (b - a) / n
    x = [0] * (n + 1)
    y = [0] * (n + 1)
    alpha = [0] * n
    beta = [0] * n
    ans = []
    x[0] = a
    x[n] = b
    alpha[0] = -gamma1 / (h * sigma1 - gamma1)
    beta[0] = delta1 / (sigma1 - gamma1 / h)

    for i in range(n - 1):
        x[i + 1] = x[i] + h
        vr_h = h ** (-2)
        vr_p = p(x[i + 1]) / (2 * h)
        A = vr_h - vr_p
        B = vr_h + vr_p
        C = -2 * vr_h + q(x[i + 1])
        alpha[i + 1] = -B / (A * alpha[i] + C)
        beta[i + 1] = (f(x[i + 1]) - A * beta[i]) / (A *
alpha[i] + C)

    y[n] = (h * delta2 + gamma2 * beta[n - 1]) / (h *
sigma2 + gamma2 * (1 - alpha[n - 1]))

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

    for i in range(n + 1):
        ans.append([x[i], y[i]])

    return ans
```

```
#основная программа

print("Solution of the boundary value problem by the sweep
method according to the variant:")

print(*sweep_bound(p1, q1, f1, 0.7, 1, 0, 1, 1.3, 0.5, 1,
2, 20), sep='\n')

print()


print("Solution of the boundary value problem by the sweep
method own version number 1:")

print(*sweep_bound(p2, q2, f2, 0, 1, 1, 0, 0, 0, 1, 4, 20),
sep='\n')

print()


print("Solution of the boundary value problem by the sweep
method own version number 2:")

print(*sweep_bound(p3, q3, f3, -1, 0, 1, 0, e ** (-3), 0,
1, 3, 20), sep='\n')
```

# Список литературы

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