**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**имени М.В.Ломоносова**

**Факультет вычислительной математики и кибернетики**

## Компьютерный практикум по учебному курсу

## «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

# ЗАДАНИЕ № 2_2

## Численные методы решения дифференциальных уравнений

## ОТЧЕТ

**о выполненном задании**

студента 203-ей учебной группы факультета ВМК МГУ

Кудисова Артёма Аркадьевича

гор. Москва

2020 г.

## Цель работы

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

## Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида

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

с дополнительными условиями в граничных точках

$$\begin{cases} \sigma_1 y(0) + \gamma_1 y'(0) & = & \delta_1, \\ \sigma_2 y(1) + \gamma_2 y'(1) & = & \delta_2. \end{cases} \tag{2}$$

## Основные цели

1. Решить краевую задачу (1)-(2) 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 (подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций, при проверке можно использовать ресурсы on-line системы http://www.wolframalpha.com или пакета Maple и т.п.)

## Решение краевой задачи

Рассмотрим линейное дифференциальное уравнение второго порядка

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

на отрезке $(a, b)$ с дополнительными условиями в граничных точках

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

Зададим на $[a, b]$ равномерную сетку с шагом $h = \frac{b-a}{n}$. Сетка состоит из $n+1$ узла вида $x_i = a + ih$, $i = \overline{0, n}$.

Аппроксимируем наши производные через центральные разностные производные:

$$y_i' \approx \frac{y_{i+1} - y_{i-1}}{2h},\ y_i'' \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2},\ i = \overline{1, n-1}$$

Тогда наше дифференциальное уравнение примет следующий вид

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

Теперь перегруппируем коэффициенты при соответствующих $y$ и получим

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

где $A_i = h^{-2} - \frac{p(x_i)}{2h},\ B_i = -2h^{-2} + q(x_i),\ C_i = h^{-2} + \frac{p(x_i)}{2h},\ D_i = -f(x_i)$

Вернемся к условиям в граничных точках и аппроксимируем $y'(a)$ справа и $y'(b)$ слева:

$$y'(a) \approx \frac{y_1 - y_0}{h},\ y'(b) \approx \frac{y_n - y_{n-1}}{h}$$

Тогда мы имеем следующую систему

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

И, перегруппировав слагаемые, получим

$$\begin{cases} B_0 y_0 + C_0 y_1 = D_0 \\ A_n y_{n-1} + B_n y_n = D_n \end{cases}$$

$$\begin{aligned} \text{где}\quad & B_0 = \sigma_1 - \gamma_1/h,\ C_0 = \gamma_1/h,\ D_0 = \delta_1 \\ & A_n = -\gamma_2/h,\ B_n = \sigma_2 + \gamma_2/h,\ D_n = \delta_2 \end{aligned}$$

В итоге у нас есть n + 1 неизвестных $y_0, y_1, ..., y_n$ и СЛАУ с трехдиагональной матрицей, которую можно решить методом прогонки.

$$\begin{bmatrix} B_0 & C_0 & 0 & 0 & ... & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & ... & 0 & 0 & 0 \\ 0 & A_2 & B_2 & C_2 & ... & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & ... & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & 0 & 0 & 0 & ... & 0 & A_n & B_n \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ ... \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ ... \\ D_{n-1} \\ D_n \end{bmatrix}$$

## Метод прогонки

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

$$\begin{cases} B_0 y_0 + C_0 y_1 = D_0 \\ A_n y_{n-1} + B_n y_n = D_n \end{cases}$$

Суть метода прогонки заключена в предположении, что искомые неизвестные $y_i,\ y_{i+1}$ связаны несложным рекуррентным соотношением

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1},\ i = \overline{0, n-1}$$

Выразим $y_{i-1}$ через $y_{i+1}$

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

И подставим $y_i,\ y_{i-1}$, выраженные через $y_{i+1}$, в исходное уравнение. В результате получим

$$(A_i \alpha_i \alpha_{i+1} + B_i \alpha_{i+1} + C_i) y_{i+1} + A_i \alpha_i \beta_{i+1} + A_i \beta_i + B_i \beta_{i+1} - D_i = 0,$$
$$i = \overline{1, n-1}$$

Мы можем добиться заведомого выполнения данного соотношения (причем независимо от решения), если потребуем, чтобы при $i = \overline{1, n-1}$ выполнялись следующие равенства:

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

Тогда мы получаем рекуррентные соотношения для наших прогоночных коэффициентов $\alpha_{i+1},\ \beta_{i+1}$

$$\alpha_{i+1} = \tfrac{-C_i}{A_i \alpha_i + B_i},\ \beta_{i+1} = \tfrac{D_i - A_i \beta_i}{A_i \alpha_i + B_i},\ i = \overline{1, n-1}$$

Вернемся к левому граничному условию и рекуррентному соотношению

$$B_0 y_0 + C_0 y_1 = D_0$$
$$y_0 = \alpha_1 y_1 + \beta_1$$

Получаем, что $\alpha_1 = -C_0/B_0,\ \beta_1 = D_0/B_0$. Теперь мы можем вычислить остальные прогоночные коэффициенты $\alpha_2, ..., \alpha_n$ и $\beta_2, ..., \beta_n$.

Из правого граничного условия, выразив $y_{n-1}$ через $y_n$ мы получаем, что

$$A_n(\alpha_n y_n + \beta_n) + B_n y_n = D_n$$
$$(A_n\alpha_n + B_n)y_n = D_n - A_n\beta_n$$
$$y_n = (D_n - A_n\beta_n)/(A_n\alpha_n + B_n)$$

Вычислив таким образом $y_n$ мы можем найти и остальные неизвестные $y_0, y_1, ..., y_{n-1}$ через рекуррентные соотношения.

## Описание программы

Программа написана на языке python и состоит из следующих функций:

1. def forward_pass(test: dict, n: int)

   Вычисляет прогоночные коэффициенты $\alpha_1, \alpha_2, ..., \alpha_n$ и $\beta_1, \beta_2, ..., \beta_n$ и возвращает 2 списка $\alpha$ и $\beta$.

   Аргументы:

   - test - словарь, содержащий в себе всю информацию о примере: его описание, границы, краевые условия
   - n - количество частей, на которые поделен интервал $[a, b]$

2. def backward_pass(test: dict, n: int, alpha: list, beta: list) -> list

   Вычисляет значения неизвестных $y_0, y_1, ..., y_n$ и возвращает список $y$.

   Аргументы:

   - test - словарь, содержащий в себе всю информацию о примере: его описание, границы, краевые условия
   - n - количество частей, на которые поделен интервал $[a, b]$
   - alpha - список прогоночных коэффициентов $\alpha_1, \alpha_2, ..., \alpha_n$
   - beta - список прогоночных коэффициентов $\beta_1, \beta_2, ..., \beta_n$

3. def main()

   Главная функция, считывающая пользовательский ввод, запускающая остальные функции для вычислений и выводящая ответ.

## Код программы

```
import math

test1 = {
    "description": "y’’ - xy’ + 2y = x - 1\n"
                   "y(0.9) - 0.5y’(0.9) = 2\ny(1.2) = 1\n",
    "p": lambda x: -x,
    "q": lambda x: 2,
    "f": lambda x: 1 - x,
    "a": 0.9, "b": 1.2,
    "s1": 1, "g1": -0.5, "d1": 2,
    "s2": 1, "g2": 0, "d2": 1
}

test2 = {
    "description": "y’’ - 0.5y’ - 3y = 2x^2\n"
                   "y(1) - 2y’(1) = 0.6\ny(1.3) = 1\n",
    "p": lambda x: -0.5,
    "q": lambda x: -3,
    "f": lambda x: -2 * x * x,
    "a": 1, "b": 1.3,
    "s1": 1, "g1": -2, "d1": 0.6,
    "s2": 1, "g2": 0, "d2": 1
}

test3 = {
    "description": "y’’ + y’ = 1\ny’(0) = 0\ny(1) = 1\n",
    "p": lambda x: 1,
    "q": lambda x: 0,
    "f": lambda x: -1,
    "a": 0, "b": 1,
    "s1": 0, "g1": 1, "d1": 0,
    "s2": 1, "g2": 0, "d2": 1
}

test4 = {
    "description": "y’’ + y = 1\ny’(0) = 0\n"
                   "y(0.5 * pi) - y’(0.5 * pi) = 2\n",
    "p": lambda x: 0,
    "q": lambda x: 1,
    "f": lambda x: -1,
    "a": 0, "b": math.pi/2,
    "s1": 0, "g1": 1, "d1": 0,
    "s2": 1, "g2": -1, "d2": 2
}
```

```
def forward_pass(test: dict, n: int):
    h = (test["b"] - test["a"]) / n

    C = test["g1"] / h
    B = test["s1"] - C
    D = test["d1"]

    alpha = [-C/B]
    beta = [D/B]

    x = test["a"] + h
    for i in range(n - 1):
        A = 1 / h / h - test["p"](x) / 2 / h
        B = -2 / h / h + test["q"](x)
        C = 1 / h / h + test["p"](x) / 2 / h
        D = -test["f"](x)

        alpha.append(-C / (A * alpha[i] + B))
        beta.append((D - A * beta[i]) / (A * alpha[i] + B))
        x += h

    return alpha, beta


def backward_pass(test: dict, n: int, alpha: list, beta: list) -> list:
    h = (test["b"] - test["a"]) / n

    A = -test["g2"] / h
    B = test["s2"] - A
    D = test["d2"]

    y = [(D - A * beta[n-1]) / (A * alpha[n-1] + B)]
    for i in range(n):
        y.append(alpha[n - i - 1] * y[i] + beta[n - i - 1])

    y.reverse()
    return y


def main():
    tests_dict = [test1, test2, test3, test4]

    for ind, test in enumerate(tests_dict):
        print(f'Test {ind + 1}:')
        print(test["description"])

    test_num = int(input("\nChoose the test (1, 2, 3 or 4): "))
    if test_num > len(tests_dict):
```

```
        print("Wrong input")
        return

    test = tests_dict[test_num - 1]

    try:
        n = int(input("Enter the count of steps: "))
        if n <= 0:
            raise ValueError
    except ValueError:
        print("Wrong input")
        return

    alpha, beta = forward_pass(test, n)
    y = backward_pass(test, n, alpha, beta)

    h = (test["b"] - test["a"]) / n
    x = test["a"]

    print("x\t\t\ty")
    for i in y:
        print(f'{x:.5f}\t\t{i:.10f}')
        x += h


if __name__ == '__main__':
    main()
```

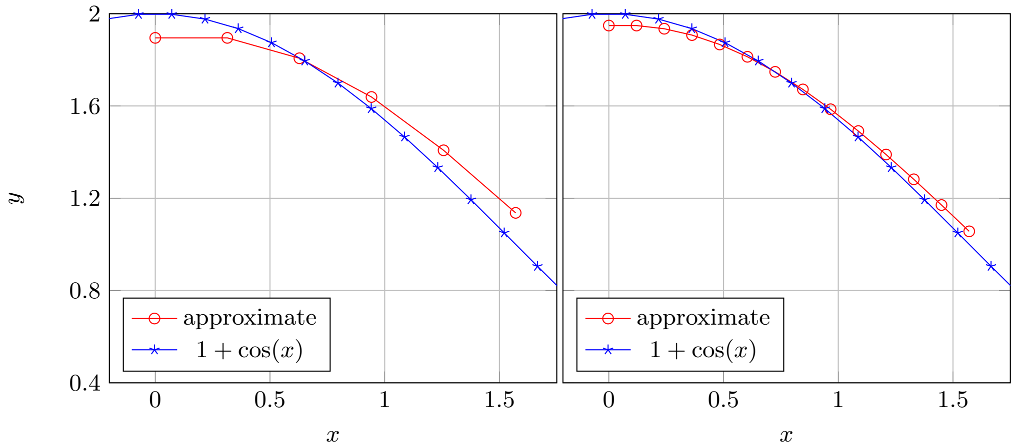
## Тестирование

Вариант 9

Для проверки решений использовался ресурс www.wolframalpha.com

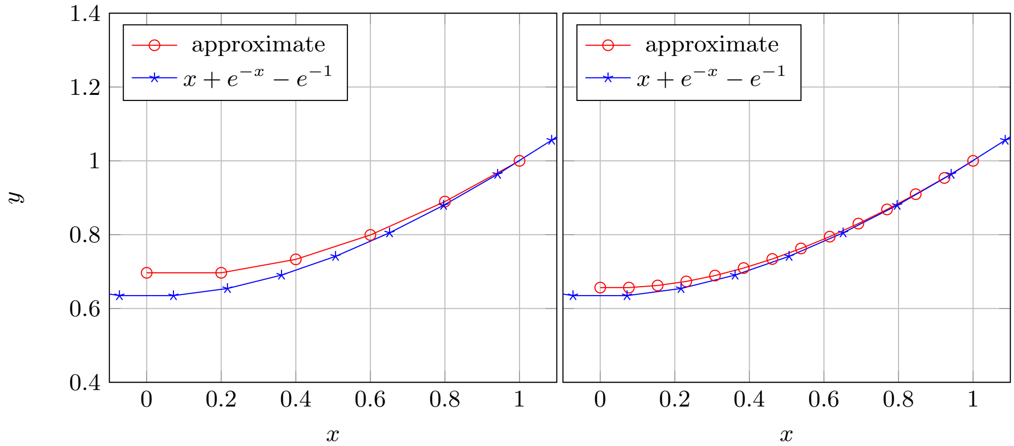
1. $$\begin{cases} y'' + y = 1 \\ y(0) = 0 \\ y(\frac{\pi}{2}) - y'(\frac{\pi}{2}) = 2 \end{cases}$$

   Аналитическое решение: $y(x) = 1 + \cos(x)$



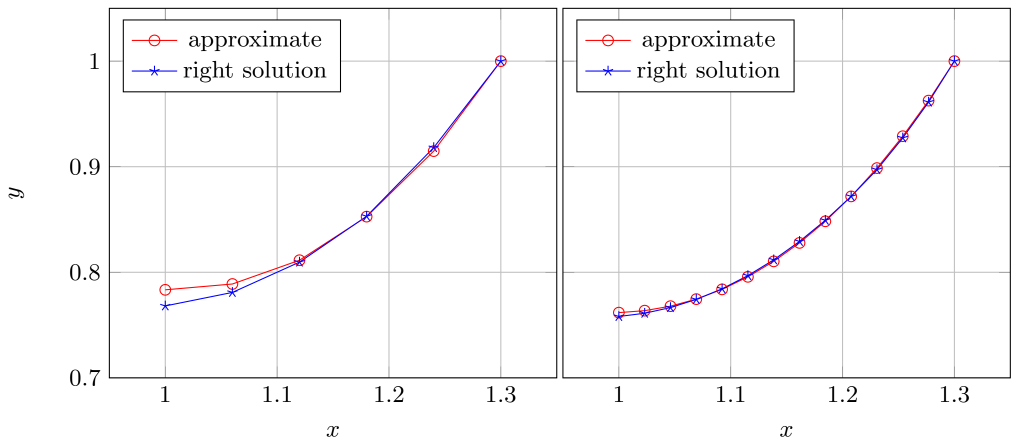

2. $$\begin{cases} y'' + y' = 1 \\ y'(0) = 0 \\ y(1) = 1 \end{cases}$$

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




3. $$\begin{cases} y'' + 0.5y' - 3y = 2x^2 \\ y(1) - 2y'(1) = 0.6 \\ y(1.3) = 1 \end{cases}$$

Аналитическое решение: $y(x) \approx -\frac{2}{3}x^2 - \frac{2}{9}x + 3.31053e^{-2x} + 0.37719e^{1.5x} - 0.481481$




## Выводы

В ходе работы был рассмотрен с теоретической точки зрения и реализован на практике метод прогонки, применяемый для решения краевой задачи ОДУ 2-ого порядка, разрешенного относительно старшей производной. На конкретных примерах была показана корректность реализации данного метода.

Стоит заметить, что точность вычислений напрямую зависит от числа разбиений отрезка, так с увеличением числа разбиений точность увеличивается. Реализация метода на практике не является сложной, однако требует крайней внимательности из-за достаточно большого числа промежуточных значений.