МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 2.

Подвариант № 2.

РЕШЕНИЕ КРАЕВОЙ ЗАДАЧИ ДЛЯ ОБЫКНОВЕННОГО ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ ВТОРОГО ПОРЯДКА, РАЗРЕШЕННОГО ОТНОСИТЕЛЬНО СТАРШЕЙ ПРОИЗВОДНОЙ

ОТЧЕТ

о выполненном задании

студента 202 учебной группы факультета ВМК МГУ

Плеханова Антона Дмитриевича

гор. Москва

2020 г.

Содержание

Цель работы

Освоить метод прогонки решения краевой задачи дифференциального уравнения второго порядка.

Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида

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

с дополнительными условиями в крайних точках

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

Цели и задачи практической работы

1. Решить краевую задачу (1)-(2) методом конечных разностей, аппроксимировав ее разностной схемой второго порядка точности (на равномерной сетке); полученну. систему конеяно-разностных уравнений решить методом прогонки;
2. Найти разностное решение задачи и построить его график;
3. Найденное решение сравнить с точным решением дифференциального уравнения.

Алгоритм

Рассматриваем задачу уравнение (1) с граничными условиями (2). Введем на отрезке $[a, b]$ равномерную сетку $x_i = a + ih, h = \frac{b-a}{n}, i = \overline{0, n}$, где n - количество шагов сетки. Решение задачи (1)-(2) сведем к вычислению сеточной функции в узловых точках x_i. Для внутренних узлов ($i = \overline{1, n-1}$) аппроксимируем первую ивторую производную разностной производной второго порядка точности:

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} + \underline{O}(h^2),$$

$$y''(x_i) = \frac{y_{i+1} - 2y_i + y_{i+1}}{h^2} + \underline{O}(h^2).$$

Рассмотрим уравнение (1) в точках x_i, $i = \overline{1, n-1}$ с учетом вышестоящих соотношений и получим систему линейных уравнений:

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

Приведем подобные слагаемые:

$$\left(1 + \frac{p_i h}{2}\right) y_{i+1} + \left(q_i h^2 - 2\right) y_i + \left(1 - \frac{p_i h}{2}\right) y_{i-1} = -f_i h^2, \;\; i = \overline{1, n-1}$$

Положим $A_i = \left(1 - \frac{p_i h}{2}\right), B_i = \left(q_i h^2 - 2\right), C_i = \left(1 + \frac{p_i h}{2}\right), F_i = -f_i h^2, i = \overline{1, n-1}$.
Для решения этой системы не хватает еще двух уравнений, которые можно получить из краевых условий.
В краевых условиях аппроксимируем первую производную разностной производной первого порядка точности:

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

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

Таким образом, положив $B_0 = \sigma_1 h - \gamma_1$, $C_0 = \gamma_1, F_0 = \delta 1 h, A_n = -\gamma_2, B_n = \sigma_2 h + \gamma_2, F_n = \delta_2 h$, получим линейную систему уравнений относитено n+1 компонент сеточной функции y:

$$\begin{cases} \mathrm{B}_0 y_0 + C_0 y_1 = F_0, \\ \mathrm{A}_i y_{i-1} + B_i y_i + C_i y_{i+1} = F_i, \quad \mathrm{i} = \overline{1, n-1} \\ \mathrm{A}_n y_{n-1} + B_n y_n = F_n. \end{cases}$$

Матрица коэффициентов этой системы имеет трехдиагональный вид:

$$\begin{bmatrix} B_0 & C_0 & & & & \\ A_1 & B_1 & C_1 & & & 0 \\ & A_2 & B_2 & C_2 & & \\ & & \dots & \dots & \dots & \\ & & & A_{n-1} & B_{n-1} & C_{n-1} \\ & 0 & & & A_n & B_n \end{bmatrix}$$

Системы с трехдиагональной матрицей удобно решать методом прогонки. Компонентны решения системы уравнений ищятся в рекуррентном виде:

$$y_i = \alpha_i y_{i+1} + \beta_i,\ i = \overline{0, n-1}$$

где α_k и β_k - прогоночные коэффициенты.
Подставим эту рекуррентную формулу в уравнение системы и получим:

$$A_i(\alpha_{i-1}y_i + \beta_{i-1}) + B_i y_i + C_i y_{i+1} = F_i,\ i = \overline{1, n-1}$$

Выразим y_i:

$$y_i = -\frac{C_i}{A_i\alpha_{i-1} + B_i}y_{i+1} + \frac{F_i - A_i\beta_{i-1}}{A_i\alpha_{i-1} + B_i}$$

Отсюда имеем:

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

Из первого уравнения системы можем вычислить α_0 и β_0:

$$\alpha_0 = -\frac{C_0}{B_0},\ \beta_0 = \frac{F_0}{B_0}$$

Из уравнения $y_{n-1} = \alpha_{n-1}y_n + \beta_{n-1}$ и последнего уравнения системы находим:

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

Подводя итог, можем разделить метод прогонки на прямую прогонку, в ходе которой последовательно от 0-ого до (n-1)-ого вычисляются прогоночные коэффициенты, и обратную прогонку, в ходе которой в обратном порядке вычисляются компоненты решения.

Описание программы

Программа написана на языке Python3. Для ее работы необходимо наличие установленной библиотеки numpy.
При запуске программа предложит выбрать номер теста (1, 2, 3, 4). Затем пользователь должен ввести n - количество шагов сетки для выбранной краевой задачи.
Программа выведет на стандартный поток вывода n пар чисел - координаты точек графика вычисленной сеточной функции. Для вариантов 3, 4 доступна опция проверки решения, поскольку для этих задач найдено аналитическое решение. Для проверки решения необходимо ввести 1, иначе - 0.
В программе реализованы следующие функции:

- $SweepMethod(A, B, C, F)$ - в этой функции реализован метод прогонки решения линейной системы с трехдиагональной матрицей. Аргументы A, B, C - одномерные массивы - соответсвенно нижняя, главная и верхняя диагонали матрицы системы, аргумент F - вектор-столбец свободых членов системы. Реализация функции полностью совпадает с описанным выше алгоритмом. Возвращает решение заданной системы;

- $BoundaryValueSolution(p, q, f, a, b, n, sigma1, sigma2, gamma1, gamma2, delta1, delta2)$ - в этой функции реализован сам алгоритм решения краевой разностными схемами. Аргументы p, q, f - функции из уравнения (1), аргументы a, b задают краевые точки, n - количество шагов сетки, аргументы $sigma1, sigma2, gamma1, gamma2, delta1, delta2$ задают краевые условия (см. уравнение (2)). Работа функции заключается в формировании системы с трехдиагональной матрицей и последующем вызове функции $SweepMethod$ для ее решения. Возвращает кортеж из массива точек сетки и значений вычисленной сеточной функции.

Реализацию вышеперечисленных функций с комментариями можно посмотреть в листинге программы.

Код программы

```
1 # coding: utf-8
2
3 import numpy as np
4 import math
5
6 # ## Метод прогонки
7 # A, B, C, F - одномерные массивы трех главных диагоналей и столбца
      свободных членов соответственно
8 def SweepMethod(A, B, C, F):
9     m = A.shape[0]
10
11     # векторрешение-
12     y = np.zeros(m)
13
14     # коэффициенты альфа и бета
15     alpha = np.zeros(m)
16     beta = np.zeros(m)
17
18     # вычисляем нулевые коэффициенты
19     alpha[0] = -C[0] / B[0]
20     beta[0] = F[0] / B[0]
21
22     # прямая прогонка:
23     # вычисляем по рекуррентной формуле оставшиеся прогоночные коэффициенты
24     for i in range(1, m):
25         alpha[i] = -C[i] / (A[i] * alpha[i-1] + B[i])
26         beta[i] = (F[i] -  A[i] * beta[i-1]) / (A[i] * alpha[i-1] +
       B[i])
27
28     # вычисляем последнюю компоненту решения
29     y[m-1] = (F[m-1] - A[m-1] * beta[m-2]) / (A[m-1] * alpha[m-2] +
       B[m-1])
30
31     # обратная прогонка
32     for i in range(m-1, 0, -1):
33         y[i-1] = alpha[i-1] * y[i] + beta[i-1]
34
35     return y
36
37
38 # ## Решение краевой задачи
39
40 # p(x), q(x), f(x) - функции из дифференциального уравнения
41 # [a, b] - отрезок, на котором решается краевая задача
42 # n - количество узлов сеточной функции
43 # sigma1, sigma2, gamma1, gamma2, delta1, delta2 задают граничные
      условия
44 def BoundaryValueSolution(p, q, f, a, b, n,
45                           sigma1, sigma2, gamma1,
46                           gamma2, delta1, delta2):
47     # вычисляем размер шага, формируем сетку
48     h = (b-a) / n
49     grid = np.linspace(a, b, n+1)
```

```
50
51      # вычисляем значения функций p(x), q(x), f(x) в точках сетки
52      grid_p = p(grid)
53      grid_q = q(grid)
54      grid_f = f(grid)
55
56      # главные диагонали матрицы снизу( вверх)
57      A = np.zeros(n+1)
58      B = np.zeros(n+1)
59      C = np.zeros(n+1)
60
61      # веткорстолбец- свободных членов
62      F = np.zeros(n+1)
63
64      # учитываем граничные условия
65      B[0] = sigma1 * h - gamma1
66      C[0] = gamma1
67      F[0] = delta1 * h
68      A[n] = -gamma2
69      B[n] = sigma2 * h + gamma2
70      F[n] = delta2 * h
71
72      # вычисляем диагональные элементы матрицы системы
73      for i in range(1, n):
74          A[i] = 1 - grid_p[i] * h/2
75          B[i] = grid_q[i] * h**2 - 2
76          C[i] = 1 + grid_p[i] * h/2
77          F[i] = -grid_f[i] * h**2
78
79      # находим решение методом прогонки
80      grid_y = SweepMethod(A, B, C, F)
81
82      return (grid, grid_y)
83
84
85  # ##  Тестирование Вариант( 14 - основной)
86
87  # y'' + 2*x^2*y' + y = x
88  # 2*y(0.5) - y'(0.5) = 1
89  # y(0.8) = 3
90
91  def p1(x):
92      return 2 * x**2
93
94  def q1(x):
95      return np.full(x.shape, 1)
96
97  def f1(x):
98      return -x
99
100 # ## Тестирование Вариант( 5 - дополнительный)
101
102 # y'' + 2 y' - x y = x^2
103 # y'(0.6) = 0.7
104 # y(0.9) - 0.5 y'(0.9) = 1
105
```

```
106 def p2(x):
107     return np.full(x.shape, 2)
108
109 def q2(x):
110     return -x
111
112 def f2(x):
113     return -x**2
114
115 # ## Дополнительный тест 1
116
117 # y'' + y = 1
118 # y(0)=0
119 # y'(1)=1
120
121 def solution_3(x):
122     return -np.cos(x) + (1 - math.sin(1)) / math.cos(1) * np.sin(x)
        + 1
123
124 def p3(x):
125     return np.full(x.shape, 0)
126
127 def q3(x):
128     return np.full(x.shape, 1)
129
130 def f3(x):
131     return np.full(x.shape, -1)
132
133
134 # ## Дополнительный тест 2
135
136 # y'' +  2y' = x,
137 # y(0)=0,
138 # y'(1)=1
139
140 # 1/8 (3 e^2 - 3 e^(2 - 2 x) + 2 (-1 + x) x)
141
142 def solution_4(x):
143     return 1/8 * (2*x*(x-1) - 3 * np.exp(2 - 2*x) + 3 * np.exp(np.
       full(x.shape, 2)))
144
145 def p4(x):
146     return np.full(x.shape, 2)
147
148 def q4(x):
149     return np.full(x.shape, 0)
150
151 def f4(x):
152     return -x
153
154
155 def main():
156     print("Выберите вариант задания (1, 2, 3, 4):")
157
158     var = int(input())
159
```

```
    p = {1: p1, 2: p2, 3: p3, 4: p4}
    q = {1: q1, 2: q2, 3: q3, 4: q4}
    f = {1: f1, 2: f2, 3: f3, 4: f4}

    if var == 1:
        # задаем параметры краевых условий
        a = 0.5; b = 0.8
        sigma1 = 2; sigma2 = 1
        gamma1 = -1; gamma2 = 0
        delta1 = 1; delta2 = 3
    elif var == 2:
        # задаем параметры краевых условий
        a = 0.6; b = 0.9
        sigma1 = 0; sigma2 = 1
        gamma1 = 1; gamma2 = -0.5
        delta1 = 0.7; delta2 = 1
    elif var == 3:
        # задаем параметры краевых условий
        a = 0; b = 1
        sigma1 = 1; sigma2 = 0
        gamma1 = 0; gamma2 = 1
        delta1 = 0; delta2 = 1
    elif var == 4:
        # задаем параметры краевых условий
        a = 0; b = 1
        sigma1 = 1; sigma2 = 0
        gamma1 = 0; gamma2 = 1
        delta1 = 0; delta2 = 1
    else:
        print("Такого варианта нет")
        return

    n = int(input("Введите количество шагов сетки: "))

    res = BoundaryValueSolution(p[var], q[var], f[var], a, b, n,
                                sigma1, sigma2, gamma1,
                                gamma2, delta1, delta2)

    print("Вычисленное решение:")
    for tpl in zip(res[0], res[1]):
        print(f"({tpl[0]}, {tpl[1]})")

    if (var >= 3):
        print("Для выбранного варианта найдено аналитическое решение.")
        print("Для сравнения аналитического и вычисленного решения введите
    1, иначе - 0.")

        check = int(input())
        if check == 1:
            solution = {3: solution_3, 4: solution_4}
            print("Аналитическое решение, вычисленное в точках сетки:")
            for tpl in zip(res[0], solution[var](res[0])):
                print(f"({tpl[0]}, {tpl[1]})")

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

Тестирование программы

Тест 1 (Вариант 14)

Уравнение:

$$y'' + 2x^2y' + y = x$$

Краевые условия: $\begin{cases} 2y(0.5) - y'(0.5) = 1 \\ y(0.8) = 3 \end{cases}$

На графике приведен результат работы программы с количеством шагов сетки 20 (оранжевым цветом) и 40 (синим цветом):

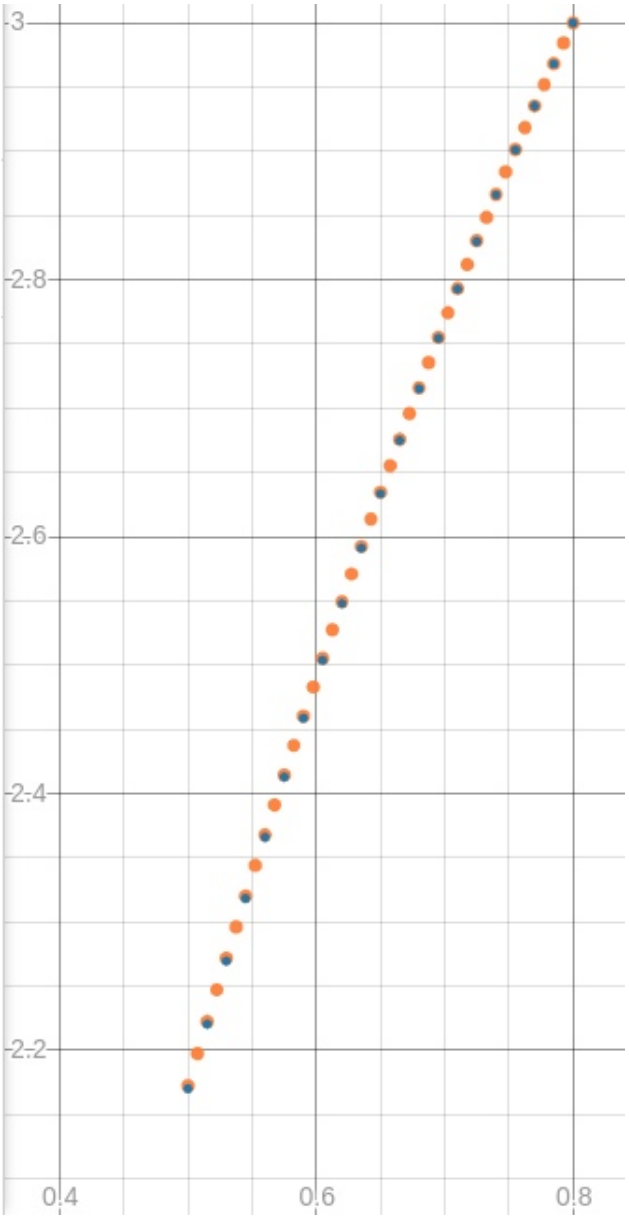

Рис. 1:

Тест 2 (Вариант 15)

Уравнение:

$$y'' + 2y' - xy = x^2$$

Краевые условия: $\begin{cases} \text{y'}(0.6) = 0.7 \\ \text{y}(0.9) - 0.5\text{y'}(0.9) = 1 \end{cases}$

На графике приведен результат работы программы с количеством шагов сетки 20 (оранжевым цветом) и 40 (синим цветом):

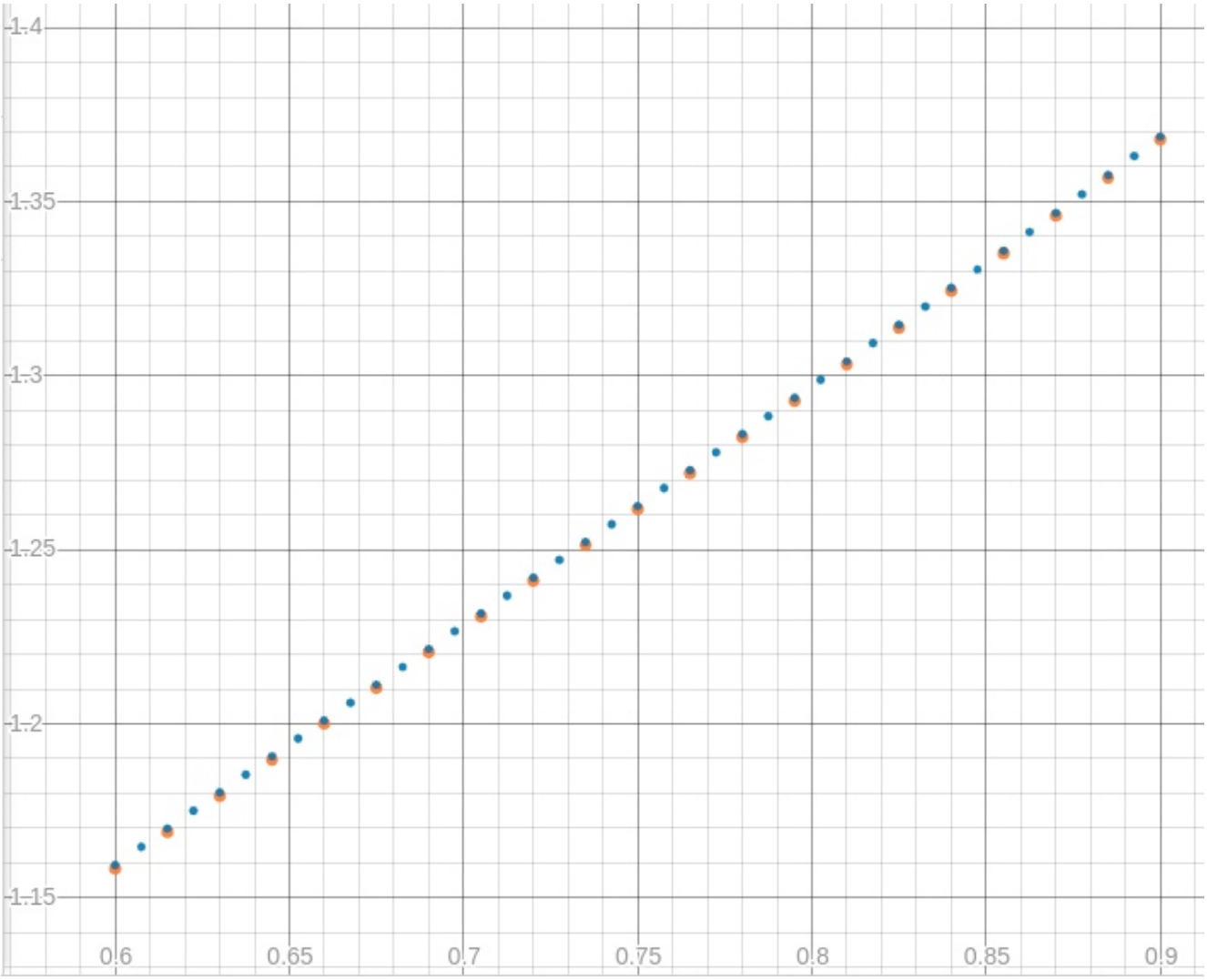

Рис. 2:

Тест 3

Уравнение:

$$y'' + y' = 1$$

Краевые условия: $\begin{cases} \text{y}(0) = 0 \\ \text{y'}(1) = 1 \end{cases}$

Аналитическое решение:

$$y = -\cos x + \frac{1 - \sin 1}{\cos 1} \sin x + 1$$

На графике приведен результат работы программы с количеством шагов сетки 20 (оранжевым цветом) и 40 (синим цветом), а также аналитическое решение (зеленым цветом):

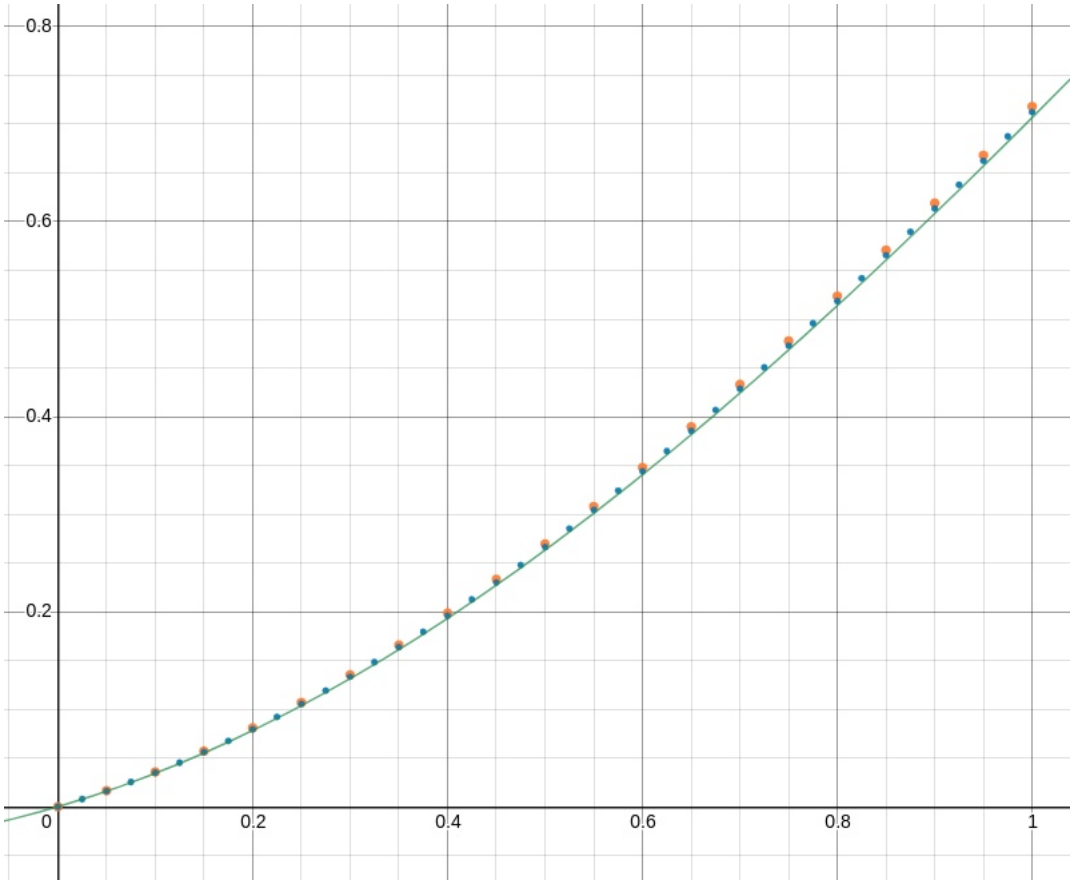

Рис. 3:

Тест 4

Уравнение:

$$y'' + 2y' = x$$

Краевые условия: $\begin{cases} \text{y}(0) = 0 \\ \text{y'}(1) = 1 \end{cases}$

Аналитическое решение:

$$y = \frac{1}{8}(3e^2 - 3e^{2-2x} + 2x(x-1)$$

На графике приведен результат работы программы с количеством шагов сетки 20 (оранжевым цветом) и 40 (синим цветом), а также аналитическое решение (зеленым цветом):

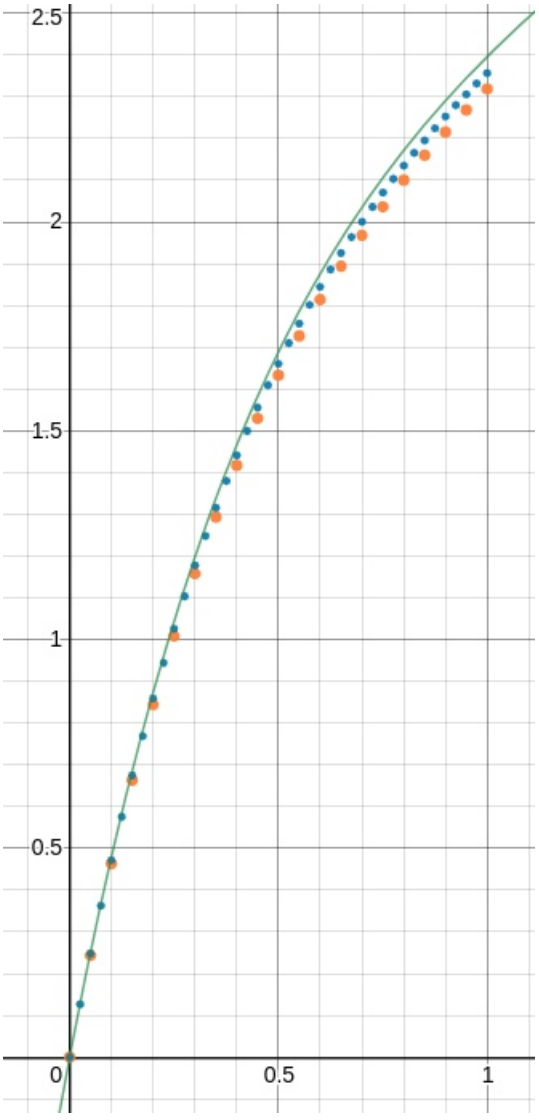

Рис. 4:

Выводы

В ходе работы был изучен метод прогонки решения СЛАУ с трехдиагональной матрицей в приложении к решению краевой задачи дифференциального уравнения второго порядка. Была разработана и протестирована программа, осуществляющая решение краевой задачи. Тесты, в которых есть возможность найти аналитическое решение, доказывают работоспособность метода.