Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по курсе

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ №2
Подвариант №2

Решение систем линейных алгебраических уравнений методом гаусса и методом гаусса с выбором главного элемента

ОТЧЕТ
о выполненном задании

студента 202 учебной группы факультета ВМК МГУ
Кузнецова Михаила Константиновича

Москва
2020

Содержание

Цель работы

Изучить классический метод Гаусса (а также модифицированный метод Гаусса), применяемый для решения системы линейных алгебраических уравнений.

Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида:

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

с дополнительными условиями в граничных точках

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

Задачи практической работы

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.

Описание метода

Рассмотрим на $[a;b]$ равномерную сетку из $n+1$ узлов с шагом $h = \frac{b-a}{n}$ и узлами $x_i = a+ih, i = 0, 1, ..., n$. Обозначим $y_i = y(x_i), p_i = p(x_i), q_i = q(x_i), f_i = f(x_i)$. Аппроксимировав производные разностными функциями второго порядка точности. Получим:

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

Перегруппировав коэффициенты при соответствующих y_i получим:

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

Для получения еще двух уравнений, аппроксимируем производные в граничных точках:

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

Опять перегруппируем слагаемые, получим:

$$\begin{cases} B_0 y_0 + C_0 y_1 = D_0 \\ A_{n+1} y_n + B_{n+1} y_{n+1} = D_{n+1} \end{cases}$$

Получили СЛАУ с трехдиагональной матрицей с $n+2$ неизвестными. Ее можно решить методом прогонки. Он основан на предположении, что искомые неизвестные связаны реккурентным соотношением

$$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1}, 0 \le i \le n-1$$

С помощью этой формулы выразим x_{i-1} через x_{i+1} и подставим выраженные x_{i-1} и x_i в исходное уравнения. Получим

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

Это равенство будет выполняться независимо от решения, если потребовать, чтобы:

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

Отсюда следуют реккурентные соотношения для прогоночных коэффициентов:

$$\alpha_{i+1} = \frac{-B_i}{A_i\alpha_i + C_i}, \beta_{i+1} = \frac{D_i - A_i\beta_i}{A_i\alpha_i + C_i}, i = 1, 2, ..., n+1$$

Находим из первого уравнения α_1, β_1, а далее все остальные $\alpha_2, ..., \alpha_n, \beta_2, ..., \beta_n$. Далее из последнего уравнения находим $y_{n+2} = \beta_{n+2}$. Отсюда находим остальные неизвестные $y_{n+1}, ..., y_1$ в процессе обратной прогонки с помощью реккурентной формулы.

Описание программы

- При запуске программы необходимо ввести число n - количество итераций
- затем ввести номер теста

  Результатом работы программы являются строки вида $(x_i; y(x_i))$ - точки x_i с заданным шагом и посчитанные в них значения.

Код программы

```
def test1():
    a = 1
    b = 1.3
    p = lambda x: -3 / 2 * x
    q = lambda x: 0
    f = lambda x: 2 * x ** 2
    s1, g1, t1 = 1, -2, 0.6
    s2, g2, t2 = 1, 0, 1
    return a, b, p, q, f, s1, g1, t1, s2, g2, t2


def test2():
    a = 0
    b = 1
    p = lambda x: 1
    q = lambda x: 0
    f = lambda x: 1
    s1, g1, t1 = 0, 1, 0
    s2, g2, t2 = 1, 0, 1
    return a, b, p, q, f, s1, g1, t1, s2, g2, t2


def rt2(x):
    return x + np.exp(-x) - 1/np.exp(1)


def test3():
    a = 0
    b = np.log(3)
    p = lambda x: -5
    q = lambda x: 6
    f = lambda x: 2*np.exp(x)
    s1, g1, t1 = 1, 0, 2
    s2, g2, t2 = 1, 0, -6
    return a, b, p, q, f, s1, g1, t1, s2, g2, t2


def rt3(x):
    return np.exp(x) + 2*np.exp(2*x) - np.exp(3*x)



#begining of the program
n = int(input())
nt = int(input())
tst = test1
if (nt == 2):
    tst = test2
else:
    tst = test3

a, b, p, q, f, s1, g1, t1, s2, g2, t2 = tst()

h = (b - a) / n
x = a
alpha = np.zeros(n+1)
betta = np.zeros(n+1)
y = np.zeros(n+1)

alpha[1] = -g1 / (s1 * h - g1)
betta[1] = t1 * h / (s1 * h - g1)

for i in range(1, n):
    x += h
    al = 1.0 / (h * h) - p(x) / (2.0 * h)
    bl = 2.0 / (h * h) - q(x)
    cl = 1.0 / (h * h) + p(x) / (2.0 * h)
```

```
    dl = f(x)
    alpha[i+1] = cl / (bl - al * alpha[i])
    betta[i+1] = (betta[i] * al - dl)/(bl - al * alpha[i])

y[n] = (g2 * betta[n] + t2 * h) / (g2 * (1.0 - alpha[n]) + s2 * h)

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

x = np.arange(a, b + h, h).tolist()
for i in range(x.shape[0]):
    print('({},{})'.format(x[i], y[i]))
```

Тестирование программы

Вариант 9

$$y'' - 3/2xy' = 2x^2$$

$$y(1) - 2y'(1.1) = 0.6, y(1.3) = 1$$

По результатам работы программы построен график, где отмеченно полученное решение с различным количеством итераций
Количество итераций - 10, 50, 100

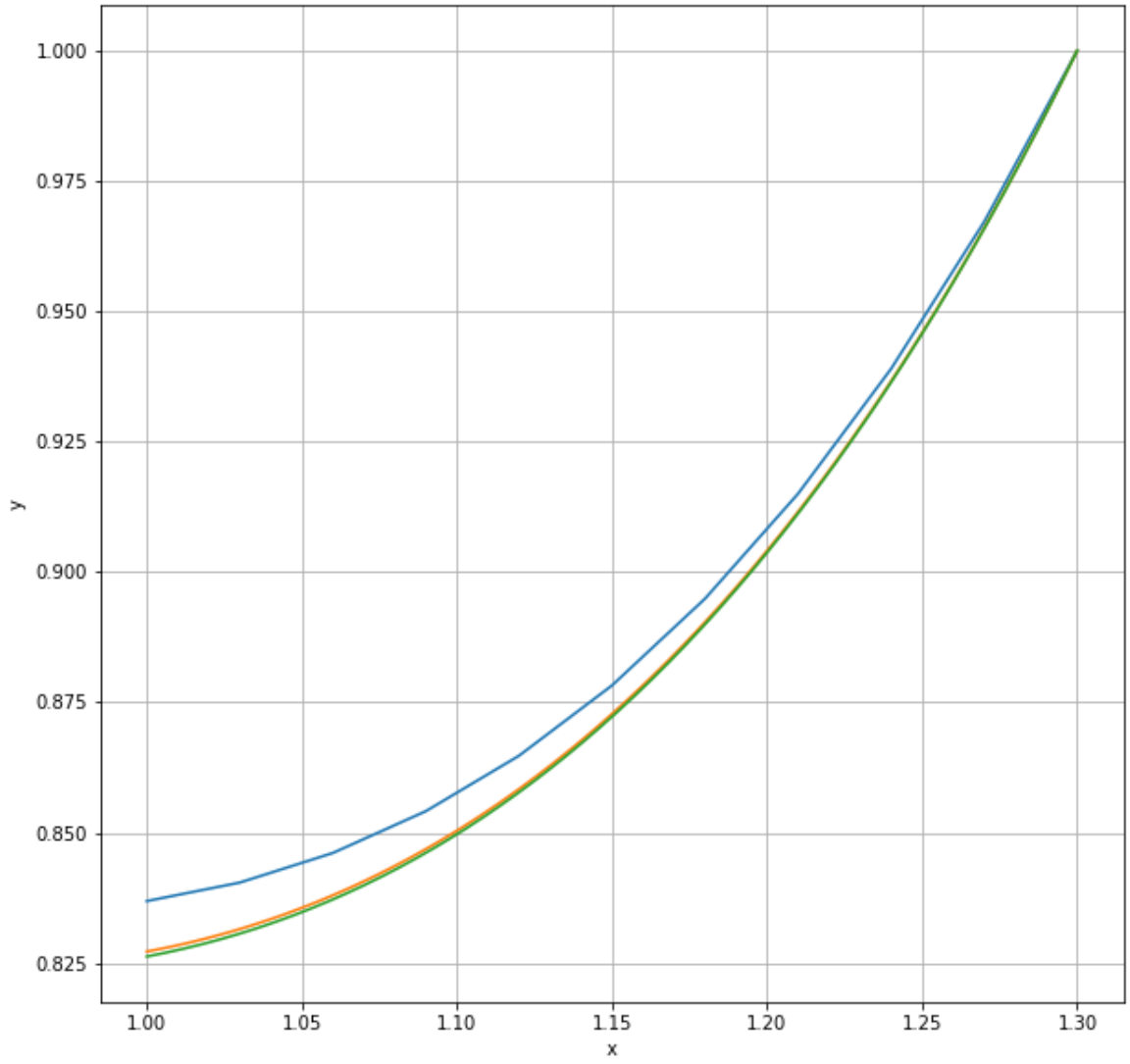

Рис. 1:

Дополнительный тест 1

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

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

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

По результатам работы программы построен график, где отмеченно аналитическое решение и полученное решение с различным количеством итераций. На нем видно, как с увеличением количества итераций найденное решение приближается к правильному.
Количество итераций - 10, 100

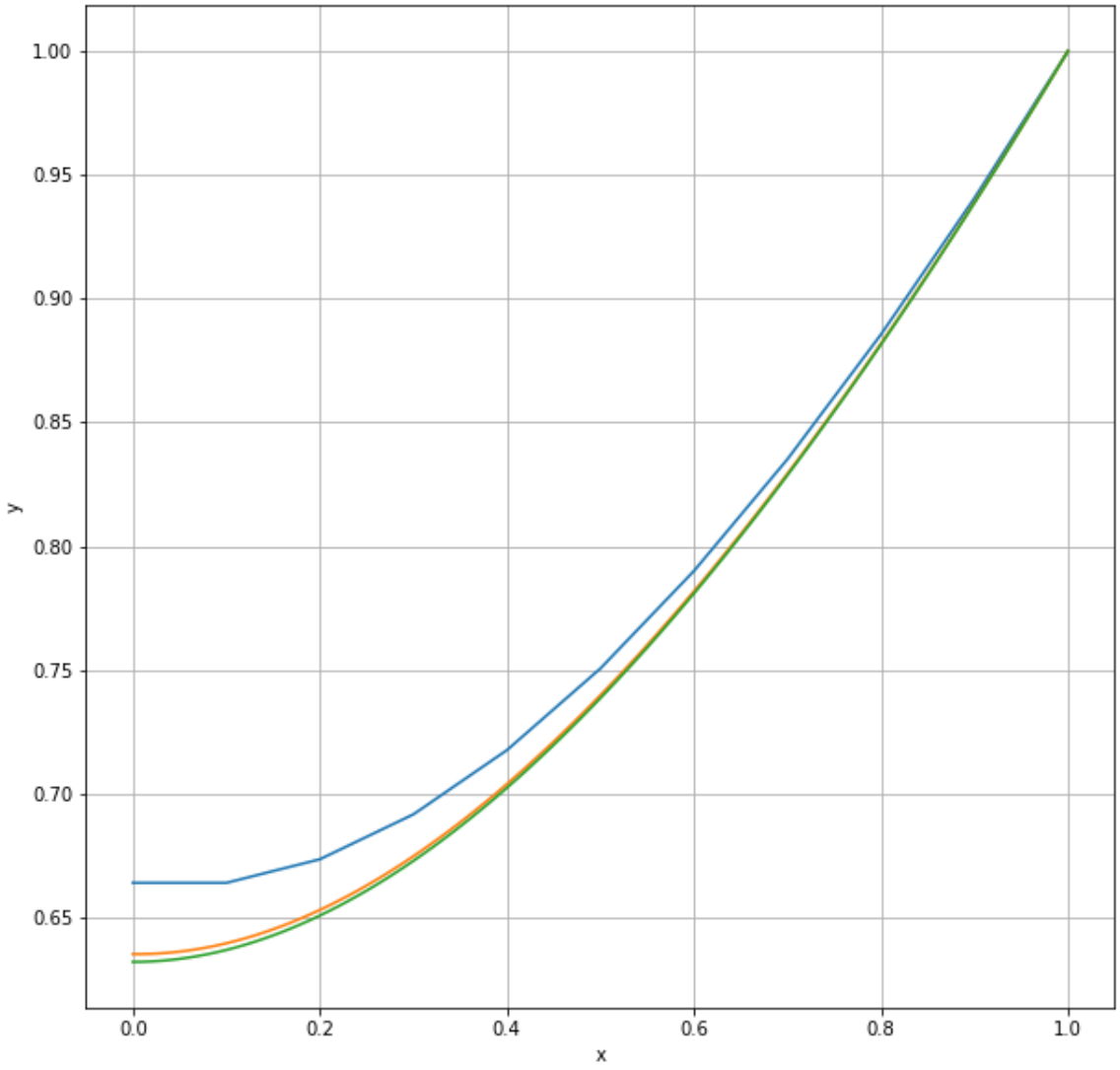

Рис. 2:

Дополнительный тест 2

$$y'' - 5y' + 6y = 2e^x$$

$$y(0) = 2, y(ln3) = -6$$

Аналитическое решение $y(x) = e^x + 2e^{2*x} - e^{3*x}$

По результатам работы программы построен график, где отмеченно аналитическое решение и полученное решение с различным количеством итераций. На нем видно, как с увеличением количества итераций найденное решение приближается к правильному.
Количество итераций - 10, 50

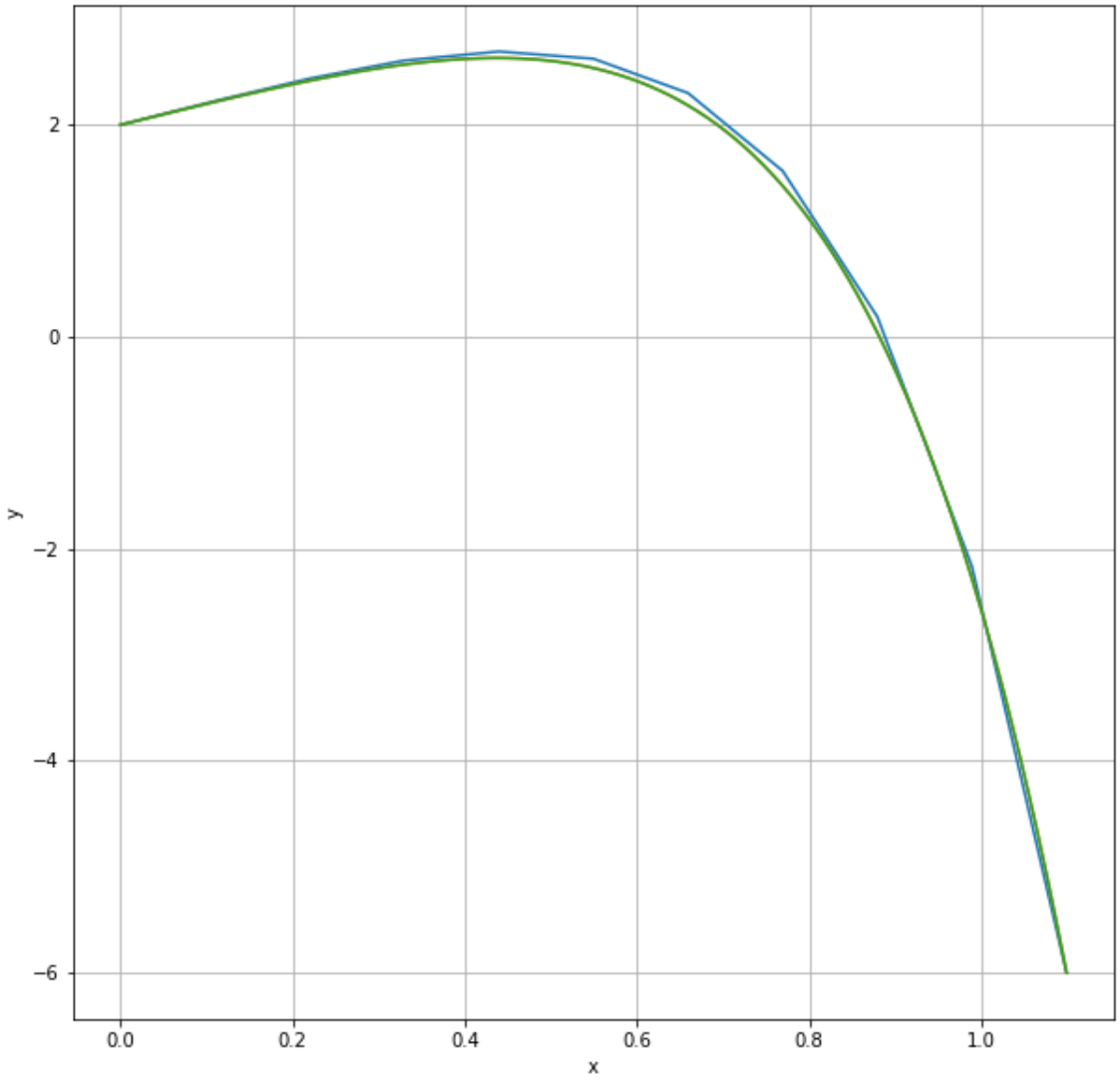

Рис. 3:

Выводы

Был рассмотрен метод прогонки, применяемый для разностного решения краевой задачи для ОДУ второго порядка, разрешенного относительно старшей производной. При тестировании было замечено, что при увеличении числа разбиений точность решения задачи увеличивается.