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

Компьютерный практикум по курсу
«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»
Задание №2 (2)

ОТЧЁТ
о выполненном задании

студента 203 учебной группы факультета ВМК МГУ
Мартынова Олега Павловича

Декабрь 2015

Оглавление

Глава 1

Постановка задачи и её целей

1.1 Цель работы

В данной работе требуется освоить метод прогонки решения краевой задачи для дифференциального уравнения второго порядка.

1.2 Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида:

$$y'' + p(x)y' + q(x)y = -f(x),\ 0 < x < 1, \tag{1.1}$$

с дополнительными условиями в граничных точках:

$$\begin{cases} \sigma_1 y(0) + \gamma_1 y'(0) = \delta_1, \\ \sigma_2 y(1) + \gamma_2 y'(1) = \delta_2. \end{cases} \tag{1.2}$$

1.3 Цели и задачи практической работы

1. Решить краевую задачу (1.1)-(1.2) методом конечных разностей, аппроксимировав её разностной схемой второго порядка точности (на равномерной сетке); полученную систему конечно-разностных уравнений решить методом прогонки.

2. Найти разностное решение задачи и построить её график.

3. Найденное решение сравнить с точным решением дифференциального уравнениям (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.).

Глава 2

Алгоритм решения

2.1 Метод прогонки

Перейдем от исходного уравнения к разностному:

$$\frac{y_{i+1} - 2y_i + y_{i+1}}{h^2} + p(x_i)\frac{y_{i+1} - yi - 1}{2h} + q(x_i)y_i = -f(x_i). \tag{2.1}$$

Приведем уравнение к трехдиагональному виду (выразим):

$$a_i y_{i-1} - b_i y_i + c_i y_{i+1} = d_i,$$

где коэффициенты a_i, b_i, c_i, d_i определяются следующим образом:

$$\begin{aligned} a_i &= \frac{1}{h^2} - \frac{p(x_i)}{2h}, \\ b_i &= \frac{2}{h^2} - q(x_i), \\ c_i &= \frac{1}{h^2} + \frac{p(x_i)}{2h}, \\ d_i &= f_i. \end{aligned}$$

Получившуюся СЛАУ удобно решить методом прогонки (т. к. матрица системы — трехдиагональная). Сперва требуется определить прогоночные коэффициенты:

$$\begin{aligned} \alpha_{i+1} &= \frac{c_i}{b_i - a_i\alpha_i}, \\ \beta_{i+1} &= \frac{a_i\beta_i - d_i}{b_i - a_i\alpha_i}, \end{aligned}$$

где начальные значения определяются, исходя из краевых условий:

$$\alpha_1 = \frac{-\gamma_1}{h\sigma_1 - \gamma_1},$$
$$\alpha_1 = \frac{h\delta_1}{h\sigma_1 - \gamma_1}.$$

После вычисления всех коэффициентов необходимо произвести обратный ход, т. е. по ним вычислить значения $y_j, j = \overline{n-1, 1}$:

$$y_{j-1} = y_j\alpha_j + \beta_j,\ j = \overline{n, 2}, \tag{2.3}$$

где значение y_n определяется исходя из начальных условий:

$$y_n = \frac{\gamma_2\beta_n + h\delta_2}{\gamma_2(1-\alpha_n) + h\sigma_2}. \tag{2.4}$$

Глава 3

Описание программы

3.1 Использование

Программа написана на языке программирования Python и состоит из нескольких модулей, из которых для пользователя представляет интерес только один — модуль *data.py*. Этот модуль содержит в себе единственную переменную — *data*, — представляющую из себя список записей, где каждая запись соответсвует уравнению или системе уравнений, предназначенных для тестирования. После заполнения этой переменной, необходимо запустить модуль *test.py*, который выведет на экран список таблиц, в которых будет отражена информация о сравнении методов решения ОДУ между собой, а также значения погрешностей каждого из методов. Помимо этого, этот модуль генерирует графики для каждой из функций, которые после его запуска могут быть найденны в текущей директории в виде изображений в привычном формате.

3.2 Детали реализации

В модуле *ode_thomas.py* реализована функция *ode_thomas*, соответствующая методу прогонки (также называемого методом Томаса) решения линейного ОДУ второго порядка с граничными условиями. Эта функция принимают в качестве входных данных функции $p(x), q(x), f(x)$, граничные условие, левую и правую границы отрезка, на котором будет происходить вычисление, и величину шага.

Глава 4

Тестирование

4.1 Примеры из одного уравнения

Пример 1. Тестовый пример 14 из оригинального задания.

$$
\begin{aligned}
p(x) &= 2x^2, \\
q(x) &= 1, \\
f(x) &= -x, \\
\sigma_1 = 2,\ &\gamma_1 = -1,\ \delta_1 = 1, \\
\sigma_2 = 1,\ &\gamma_2 = 0,\ \delta_2 = 3, \\
[x_0, x_n] &= [0.5, 0.8], \\
h &= 0.02.
\end{aligned}
$$

Вывод программы:

```
-------------------------------------------------------------
| Number of segments:                                     15 |
| Mean Squared Error (th):                               N/A |
-------------------------------------------------------------
|        x |  thomas |         |         |         |         |
-------------------------------------------------------------
|   0.5000 |  2.1835 |         |         |         |         |
|   0.5200 |  2.2509 |         |         |         |         |
|   0.5400 |  2.3164 |         |         |         |         |
|   0.5600 |  2.3801 |         |         |         |         |
|   0.5800 |  2.4419 |         |         |         |         |
|   0.6000 |  2.5019 |         |         |         |         |
|   0.6200 |  2.5601 |         |         |         |         |
|   0.6400 |  2.6164 |         |         |         |         |
|   0.6600 |  2.6708 |         |         |         |         |
|   0.6800 |  2.7234 |         |         |         |         |
```

0.7000	2.7741				
0.7200	2.8230				
0.7400	2.8700				
0.7600	2.9152				
0.7800	2.9585				
0.8000	3.0000				

Пример 2. Собственный тест №1.

$$p(x) = -2x,$$

$$q(x) = -2,$$

$$f(x) = 4x,$$

$$\sigma_1 = 1,\ \gamma_1 = -1,\ \delta_1 = 0,$$

$$\sigma_2 = 1,\ \gamma_2 = 0,\ \delta_2 = 1 + e,$$

$$[x_0, x_n] = [0, 1],$$

$$h = 0.1.$$

Вывод программы:

Number of segments:	10
Mean Squared Error (th):	0.0013

x	th	exact	err_th		
0.0000	0.9700	1.0000	0.0300		
0.1000	1.0670	1.1101	0.0430		
0.2000	1.1916	1.2408	0.0493		
0.3000	1.3467	1.3942	0.0474		
0.4000	1.5363	1.5735	0.0372		
0.5000	1.7648	1.7840	0.0192		
0.6000	2.0376	2.0333	0.0042		
0.7000	2.3611	2.3323	0.0287		
0.8000	2.7428	2.6965	0.0463		
0.9000	3.1918	3.1479	0.0439		
1.0000	3.7183	3.7183	0.0000		

Глава 5

Вывод

В данной работе был реализован метод прогонки, позволяющий находить решения краевой задачи с точностью до $O(h^2)$. Такая точность позволяет получать достаточно хорошие приближения решений, в том числе и таких, которые не выражаются в элементарных функциях.

Глава 6

Графики

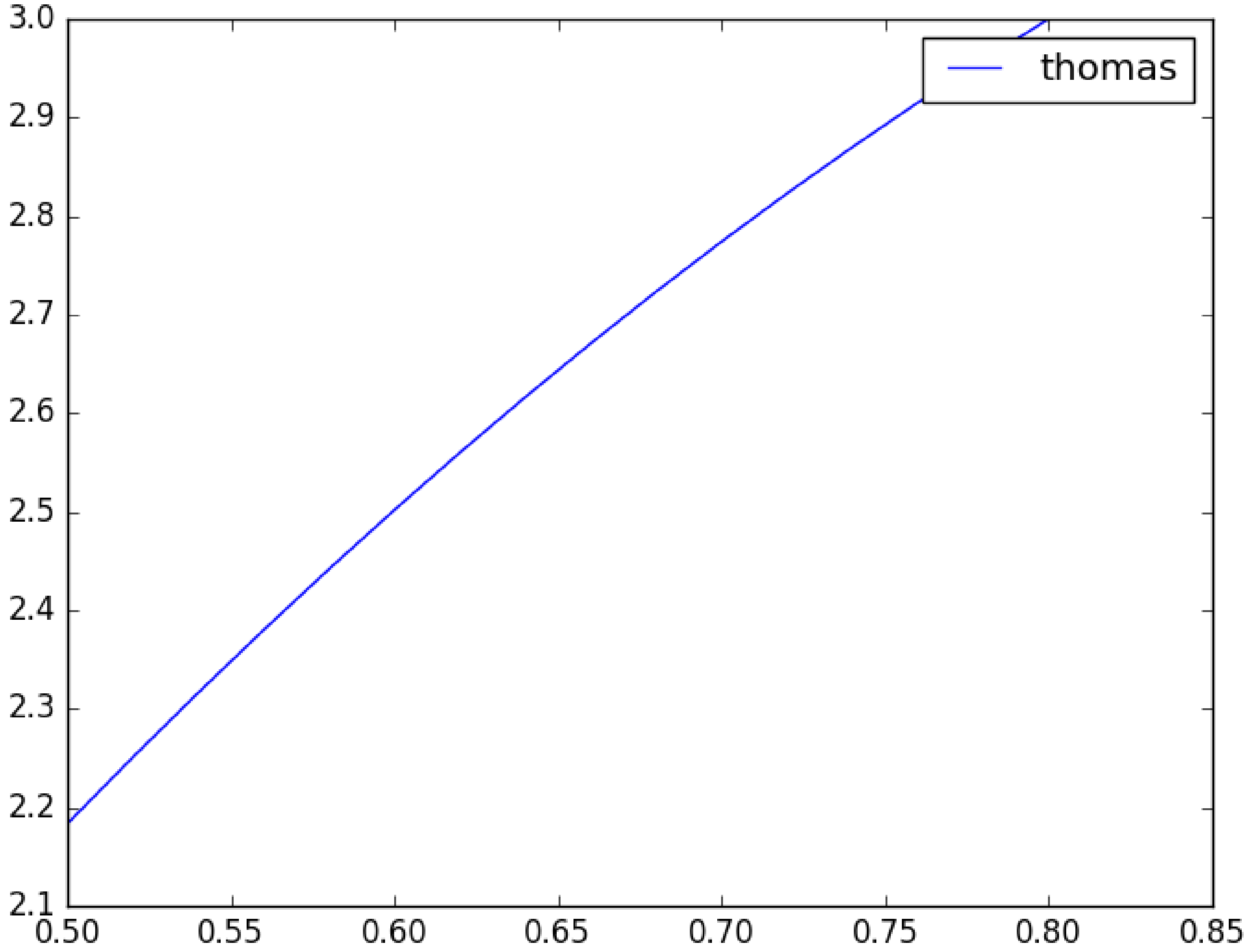

Рис. 6.1: Пример 1.

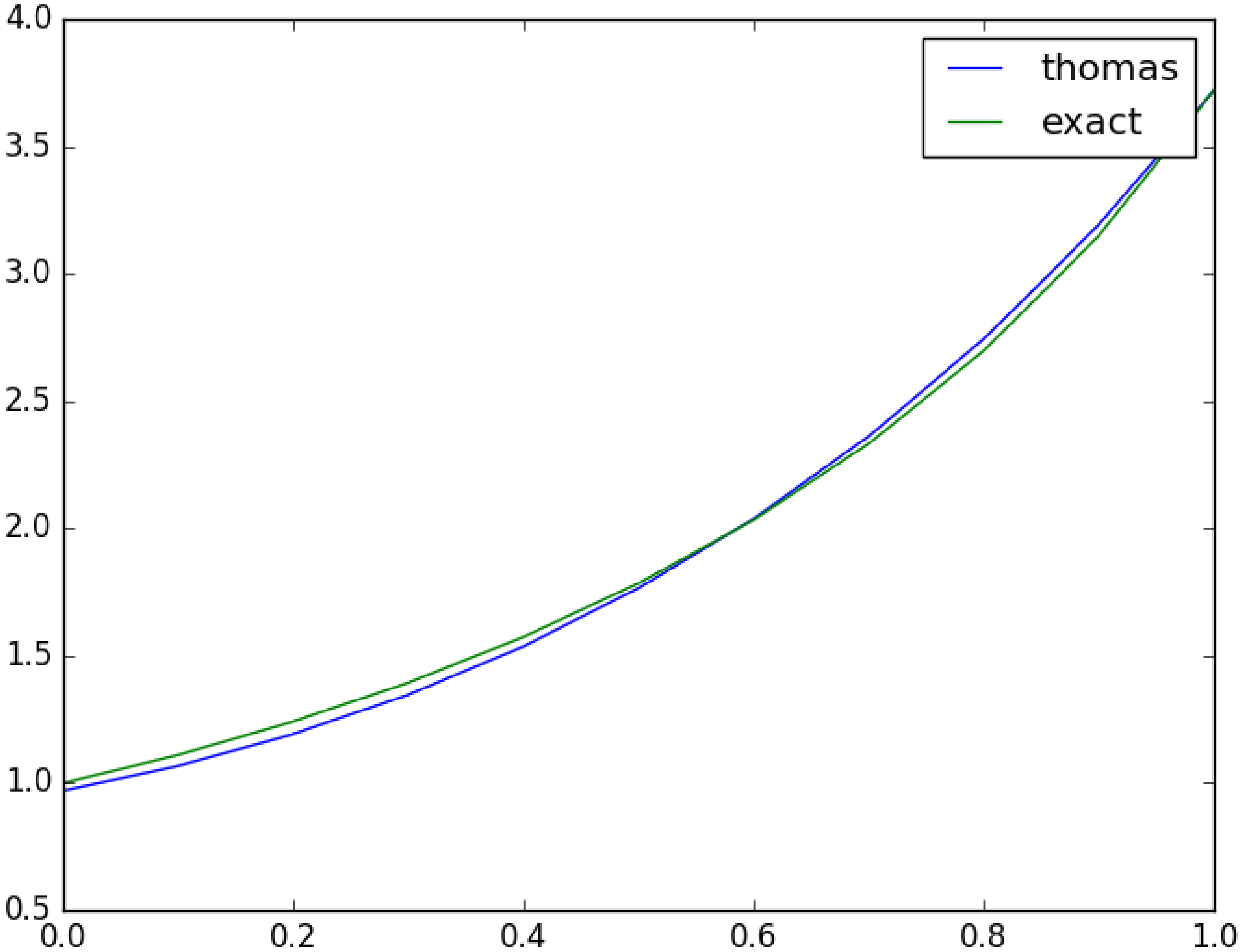

Рис. 6.2: Пример 2.

Глава 7

Исходный код

```
###########################################################
#                          test.py                        #
###########################################################
from data import data
from plot import plot_data
from ode_thomas import apply_data

# Plot table for ODEs with exact solution
for d in data:
    if not d["sol"]:
        continue

    th = apply_data(d)

    x0, xn = d["seg"]

    h_raw = d["h"]
    # normalized step size
    n = round((xn - x0) / h_raw)
    h = (xn - x0) / n

    x = [x0 + h * i for i in range(n + 1)]
    exact = list(map(d["sol"][0], x))

    mse_th = sum(map(lambda a: (a[0] - a[1]) ** 2,
                     zip(th, exact))) / (n + 1)

    print("-" * 61)
    print("| Number of segments:       {:31} |".format(n))
    print("| Mean Squared Error (th):  {:31.4f} |".format(mse_th))
    print("-" * 61)
    print("|{:>8} |{:>8} |{:>8} |{:>8} |{:>8} |{:>8} |"\
          .format("x", "thomas", "exact", "err_th", "", ""))
    print("-" * 61)
    for j in range(n + 1):
        print("|{:8.4f} |{:8.4f} |{:8.4f} |{:8.4f} |{:>8} |{:>8} |".format(x[j], th[j], exact[j]
    print("-" * 61)

# Plot table for ODEs without exact solution
for d in data:
    if d["sol"]:
        continue

    th = apply_data(d)

    x0, xn = d["seg"]

    h_raw = d["h"]
    # normalized step size
    n = round((xn - x0) / h_raw)
    h = (xn - x0) / n
```

```
    x = [x0 + h * i for i in range(n + 1)]
    print("-" * 61)
    print("| Number of segments:       {:31} |".format(n))
    print("| Mean Squared Error (th):  {:>31} |".format("N/A"))
    print("-" * 61)
    print("|{:>8} |{:>8} |{:>8} |{:>8} |{:>8} |{:>8} |"\
          .format("x", "thomas", "", "", "", ""))
    print("-" * 61)
    for j in range(n + 1):
        print("|{:8.4f} |{:8.4f} |{:>8} |{:>8} |{:>8} |{:>8} |".format(x[j], th[j], "", "", "",
    print("-" * 61)

# Plot graphs for ODEs
for d in data:
    plot_data(d)
############################################################
#                          plot.py                         #
############################################################
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
from ode_thomas import apply_data

def plot(f_num, f_ex, x0, xn, h_raw):
    """
    f_num: list of y(x) values, representing the numeric solution;
    f_ex: function y(x), representing the exact solution;
    """
    # normalized step value
    n = round((xn - x0) / h_raw)
    h = (xn - x0) / n

    x = [x0 + h * i  for i in range(n + 1)]
    fig = plt.figure()
    graph = fig.add_subplot(111)
    for fi_num, l in f_num:
        graph.plot(x, fi_num, label=l)
    for fi_ex, l in f_ex:
        graph.plot(x, list(map(fi_ex, x)), label=l)
    graph.legend()
    return fig

def plot_data(data):
    sol_num = [(apply_data(data), "thomas")]
    sol_exact = list(zip(data["sol"], ["exact"] * len(data["sol"])))
    fig = plot(sol_num,
               sol_exact,
               data["seg"][0],
               data["seg"][1],
               data["h"])
    fig.savefig("plot_{:02}.png".format(data["idx"] + 1))
############################################################
#                       ode_thomas.py                      #
############################################################

def ode_thomas(p, q, f, b, x0, xn, h_raw):
    """
    The function is to compute the solution of the following
    differential equation:
        y'' + p(x) y' + q(x) y = -f(x),
    considering following boundary conditions:
        b[0][0] * y(0) + b[0][1] * y'(0) = b[0][2],
        b[1][0] * y(1) + b[1][1] * y'(1) = b[1][2].
    """

    n = round((xn - x0) / h_raw)
    h = (xn - x0) / n
    # forward sweep
    alpha = [0] * (n + 1)
```

```
    beta = [0] * (n + 1)
    alpha[1] = -b[0][1] / (b[0][0] * h - b[0][1])
    beta[1] = b[0][2] * h / (b[0][0] * h - b[0][1])
    for i in range(1, n):
        xi = x0 + h
        ai = 1 / h ** 2 - p(xi) / (2 * h)
        bi = 2 / h ** 2 - q(xi)
        ci = 1 / h ** 2 + p(xi) / (2 * h)
        di = f(xi)
        alpha[i + 1] = ci / (bi - ai * alpha[i])
        beta[i + 1] = (ai * beta[i] - di) / (bi - ai * alpha[i])
    # backward sweep
    y = [0] * (n + 1)
    y[n] = (b[1][1] * beta[n] + b[1][2] * h) / (b[1][1] * (1 - alpha[n]) + b[1][0] * h)
    for i in range(n, 0, -1):
        y[i - 1] = y[i] * alpha[i] + beta[i]
    return y

def apply_data(data):
    return ode_thomas(data["f"][0],
                      data["f"][1],
                      data["f"][2],
                      data["b"],
                      data["seg"][0],
                      data["seg"][1],
                      data["h"])
############################################################
#                          data.py                         #
############################################################
from math import *

data = [
    {
    "idx": 0,
    "desc": "Original task, ex. 14",
    "f": [lambda x: 2 * x ** 2,
          lambda x: 1,
          lambda x: -x],
    "b": [[2., -1., 1.],
          [1., 0., 3.]],
    "seg": (0.5, 0.8),
    "h": 0.02,
    "sol": []
    },
    {
    "idx": 1,
    "desc": "Custom test #1",
    "f": [lambda x: -2 * x,
          lambda x: -2,
          lambda x: 4 * x],
    "b": [[1, -1, 0],
          [1, 0, 1 + e]],
    "seg": (0., 1.),
    "h": 0.1,
    "sol": [lambda x: x + exp(x ** 2)]
    }
]
```