



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №3 по дисциплине «Моделирование»

Тема ОДУ второго порядка с краевыми условиями 2 и 3 рода

Студент Романов А.В.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Градов В. М.

Тема работы

Программно-алгоритмическая реализация моделей на основе ОДУ второго порядка с краевыми условиями II и III рода.

Цель работы

Получение навыков разработки алгоритмов решения краевой задачи при реализации моделей, построенных на ОДУ второго порядка.

Теоретические сведения

Задана математическая модель:

$$\frac{d}{dx}(\lambda(T) \frac{dT}{dx}) - 4 \cdot k(T) \cdot n_p^2 \cdot \sigma \cdot (T^4 - T_0^4) = 0$$

Краевые условия:

$$\begin{cases} x = 0, -\lambda(T(0)) \frac{dT}{dx} = F_0. \\ x = l, -\lambda(T(l)) \frac{dT}{dx} = \alpha(T(l) - T_0) \end{cases}$$

Функции $\lambda(T)$ и $k(T)$ заданы таблицей.

Заданы начальные параметры:

$n_p = 1.4$ - коэффициент преломления

$l = 0.2$ см - толщина слоя

$T_0 = 300$ К - температура окружающей среды

$\sigma = 5.668 \cdot 10^{-12}$ Вт / ($cm^2 \cdot K^4$) - постоянная Стефана - Больцмана

$F_0 = 100$ Вт / cm^2 - поток тепла

$\alpha = 0.05$ Вт / ($cm^2 \cdot K$) - коэффициент теплоотдачи

Выход из итераций организовать по температуре и по балансу энергии:

$$\max \left| \frac{y_n^s - y_n^{s-1}}{y_n^s} \right| \leq \varepsilon_1$$

для всех $n = 0, 1, \dots, N$. и

$$\max \left| \frac{f_1^s - y_2^s}{f_1^s} \right| \leq \varepsilon_1$$

где

$$f_1 = F_0 - \alpha(T(l) - T_0)$$

$$f_2 = 4n_p^2 \sigma_0^1 k(T(x))(T^4(x) - T_0^4) dx$$

Исходный код алгоритма

```

1 from collections import namedtuple
2 from math import pow
3 from scipy.interpolate import InterpolatedUnivariateSpline
4 from numpy import arange
5
6 import plot
7
8 Params = namedtuple('Params', 'Np l T0 Tconst, sigma F0 alpha h')
9
10 params = Params(
11     1.4, 0.2, 300, 400, 5.668 * pow(10, -12), 100, 0.05, pow(10, -4)
12 )
13
14 fst_table = (
15     (
16         300, 500, 800, 1100, 2000, 2400
17     ),
18     (
19         1.36 * pow(10, -2), 1.63 * pow(10, -2), 1.81 * pow(10, -2),
20         1.98 * pow(10, -2), 2.50 * pow(10, -2), 2.74 * pow(10, -2)
21     )
22 )
23
24 snd_table = (
25     (
26         293, 1278, 1528, 1677, 2000, 2400
27     ),
28     (
29         2.0 * pow(10, -2), 5.0 * pow(10, -2), 7.8 * pow(10, -2),
30         1.0 * pow(10, -1), 1.3 * pow(10, -1), 2.0 * pow(10, -1)
31     ),
32 )
33
34 def interpolate(x_pts, y_pts, order=1):
35     return InterpolatedUnivariateSpline(x_pts, y_pts, k=order)
36
37
38 def p(k_t, t, n):
39     return 4 * params.Np * params.Np * params.sigma * k_t(t[n]) * pow(t[n], 3)
40

```

```

41
42 def f(k_t, t, n):
43     return 4 * params.Np * params.Np + params.sigma * k_t(t[n]) * pow(params.T0
44         , 4)
45
46 def x_right(l_t, t, n):
47     return (l_t(t[n]) + l_t(t[n + 1])) / 2
48
49
50 def x_left(l_t, t, n):
51     return (l_t(t[n]) + l_t(t[n - 1])) / 2
52
53
54 def p_right(k_t, t, n):
55     return (p(k_t, t, n) + p(k_t, t, n + 1)) / 2
56
57
58 def p_left(k_t, t, n):
59     return (p(k_t, t, n) + p(k_t, t, n - 1)) / 2
60
61
62 def f_right(k_t, t, n):
63     return (f(k_t, t, n) + f(k_t, t, n + 1)) / 2
64
65
66 def f_left(k_t, t, n):
67     return (f(k_t, t, n) + f(k_t, t, n - 1)) / 2
68
69
70 def A(l_t, t, n):
71     return (l_t(t[n]) + l_t(t[n - 1])) / 2 / params.h
72
73
74 def B(l_t, k_t, t, n):
75     return A(l_t, t, n) + C(l_t, t, n) + 4 * params.Np * params.Np * params.
76         sigma * k_t(t[n]) * pow(t[n], 3) * params.h
77
78
79 def C(l_t, t, n):
80     return (l_t(t[n]) + l_t(t[n + 1])) / 2 / params.h
81
82
83 def D(k_t, t, n):
84     return 4 * params.Np * params.Np + params.sigma * k_t(t[n]) * pow(params.
85         T0, 4) * params.h
86
87
88 def get_right_conditions(l_t, k_t, t):
89     K0 = x_right(l_t, t, 0) + pow(params.h, 2) / 8 * p_right(k_t, t, 0) + pow(

```

```

88     params.h, 2) / 4 * p(k_t, t, 0)
89 M0 = pow(params.h, 2) / 8 * p_right(k_t, t, 0) - x_right(l_t, t, 0)
90 P0 = params.h * params.F0 + pow(params.h, 2) / 4 * (f_right(k_t, t, 0) +
91     f_left(k_t, t, 0))
92
93
94 return K0, M0, P0
95
96
97 def get_left_conditions(k_t, l_t, t, n):
98     Kn = x_left(l_t, t, n) / params.h - params.alpha - params.h * p(k_t, t, n)
99         / 4 - params.h * p_left(k_t, t, n) / 8
100     Mn = x_left(l_t, t, n) / params.h - params.h * p_left(k_t, t, n) / 8
101     Pn = -(params.alpha * params.T0 + (f_right(k_t, t, n) + f_left(k_t, t, n))
102         / 4 * params.h)
103
104     return Kn, Mn, Pn
105
106
107 def start():
108     l_t = interpolate(fst_table[0], fst_table[1])
109     k_t = interpolate(snd_table[0], snd_table[1])
110     t = [0 for _ in range(int(1 / params.h))]
111
112     K0, M0, P0 = get_right_conditions(l_t, k_t, t)
113
114     xi_list = [0]
115     eta_list = [0]
116     x_list = list()
117
118     x = 0
119     n = 0
120     while x + params.h < 1:
121         x_list.append(x)
122
123         xi_list.append((C(l_t, t, n) / (B(l_t, k_t, t, n) - A(l_t, t, n) *
124             xi_list[n])))
125         eta_list.append((D(k_t, t, n) + A(l_t, t, n) * xi_list[n]) / (B(l_t, k_t
126             , t, n) - A(l_t, t, n) * xi_list[n])))
127
128         n += 1
129         x += params.h
130
131     Kn, Mn, Pn = get_left_conditions(k_t, l_t, t, n)
132
133     t[n] = (Pn - Mn * xi_list[n]) / (Kn + Mn * xi_list[n])
134     for i in range(n - 1, -1, -1):
135         t[i] = xi_list[i + 1] * t[i + 1] + eta_list[i + 1]

```

Результаты работы программы

1. Представить разностный аналог краевого условия при $x = l$ и его краткий вывод интегро-интерполяционным методом.

Проинтегрируем уравнение на отрезке $[X_{n-\frac{1}{2}}; x_n]$

$$-\int_{x_{n-\frac{1}{2}}}^{x_n} \frac{dF}{dx} dT - \int_{x_{n-\frac{1}{2}}}^{x_n} P(T) \cdot T^4 dT + \int_{x_{n-\frac{1}{2}}}^{x_n} f(t) dT = 0$$

Второй и третий интеграл вычислим с помощью метода трапеций:

$$F_{n-\frac{1}{2}} - F_n - \frac{h}{4}(p_n y_n + p_{n-\frac{1}{2}} y_{n-\frac{1}{2}}) + \frac{h}{4}(f_n + f_{n-\frac{1}{2}}) = 0$$

Зная, что:

$$F_{n-\frac{1}{2}} = x_{n-\frac{1}{2}} \frac{y_{n-1}}{y_n} h$$

$$F_n = \alpha_n (y_n - T_0)$$

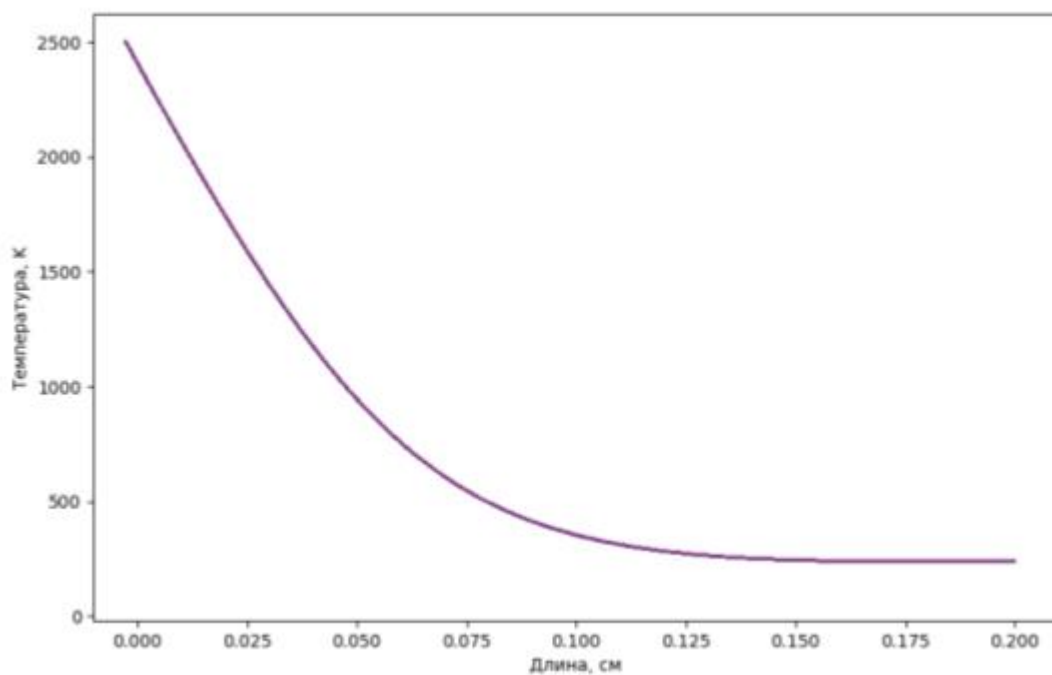
$$y_{n-\frac{1}{2}} = \frac{y_n + y_{n-1}}{2h}$$

Имеем:

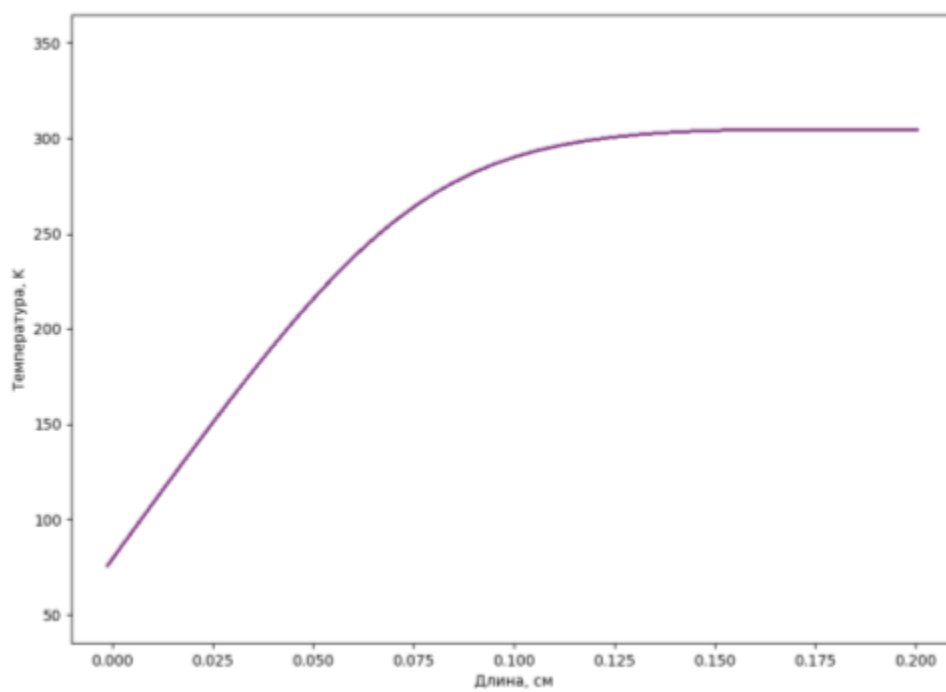
$$\frac{x_{n-\frac{1}{2}} y_{n-1}}{h} - \frac{x_{n-\frac{1}{2}} y_n}{h} - \alpha_n y_n + \alpha_n T_0 - \frac{h p_n y_n}{48} - \frac{h p_{n-\frac{1}{2}} y_n}{8} - \frac{h p_{n-\frac{1}{2}} y_{n-1}}{8} + \frac{f_{n-\frac{1}{2}} + f_n}{4} h = 0$$

$$y_n \left(-\frac{x_{n-\frac{1}{2}}}{h} - \alpha_n - \frac{h p_n}{4} - \frac{h p_{n-\frac{1}{2}}}{8} \right) + y_{n-1} \left(\frac{x_{n-\frac{1}{2}}}{h} - \frac{h p_{n-\frac{1}{2}}}{8} \right) = -(\alpha_n T_0 + \frac{f_n - \frac{1}{2}}{4} h)$$

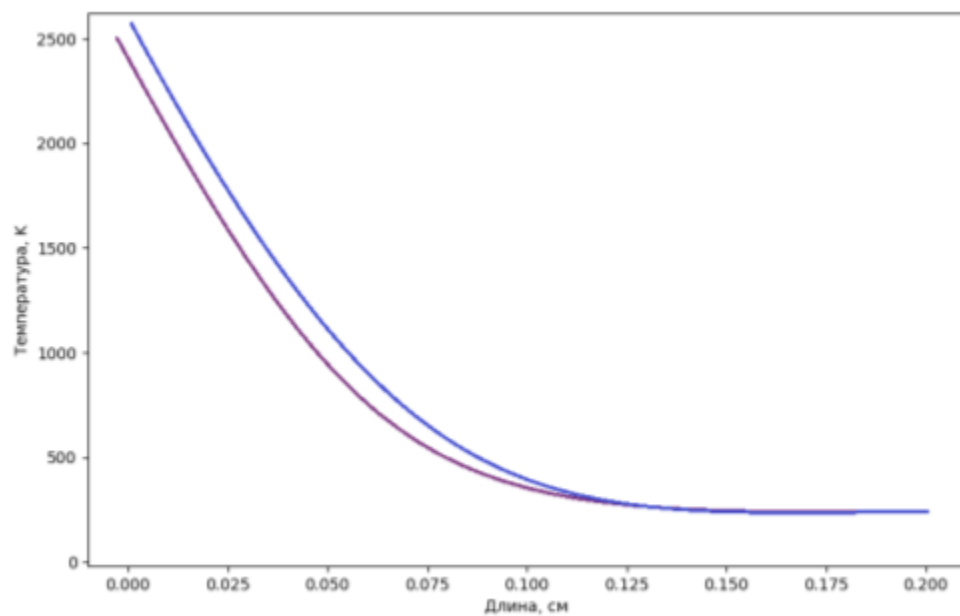
2. График зависимости температуры $T(x)$ координаты x при заданных выше параметрах.



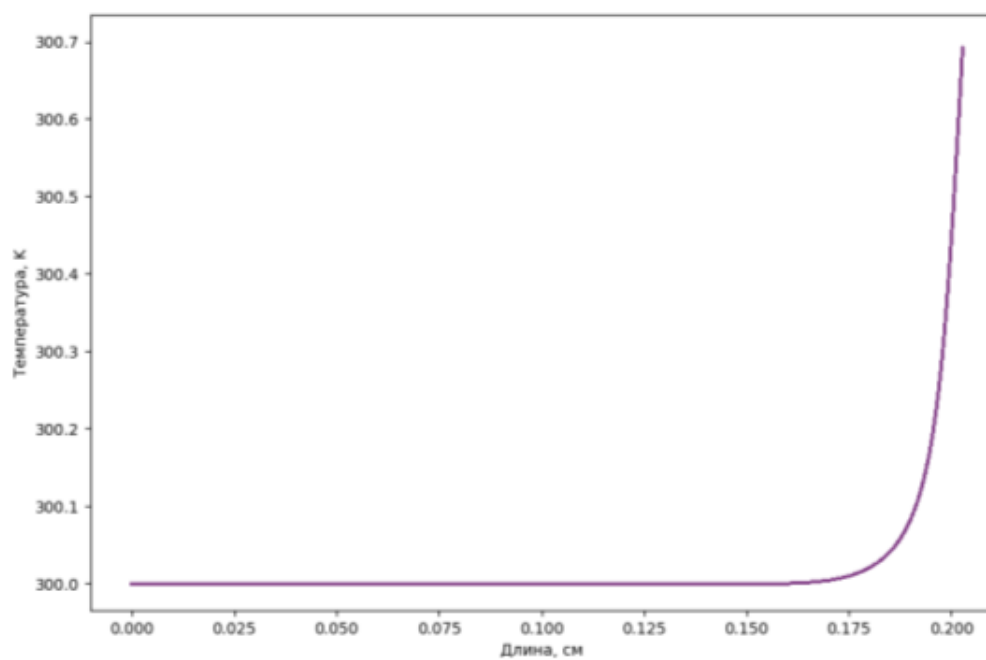
3. График зависимости $T(x)$ при $F_0 = -10 \text{ Вт} / \text{см}^2$.



4. График зависимости $T(x)$ при увеличенных значениях α (например, в 3 раза). Сравнить с п. 2.



5. График зависимости $T(x)$ при $F_0 = 0$.



6. Для указанного в задании исходного набора параметров привести данные по балансу энергии.

- Точность выхода $\varepsilon_1 = 0.069$ (по температуре)
- Точность выхода $\varepsilon_2 = 1.12$ (по балансу)

Ответы на вопросы

1. Какие способы тестирования программы можно предложить?

При $F_0 > 0$ происходит охлаждение пластины, при $F_0 < 0$ нагревание. Кроме того, при увеличении показателя теплосъема, уровень должен снижаться, а градиент увеличиваться.

2. Получите простейший разностный аналог нелинейного краевого условия при $x = l$.

Аппроксимируем производную:

$$\frac{dT}{dx} = \frac{y_N - y_{N-1}}{h}$$

Подставим в исходное уравнение:

$$-k_N \frac{y_N - y_{N-1}}{h} = \alpha_N (y_N - T_0) + \varphi(y_N)$$

Учтём, что $y_{N-1} = \xi_N y_N + \eta_N$:

$$-k_N (y_N - \xi_N y_N + \eta_N) = \alpha_N (y_N - T_0) h + \varphi(y_N) h$$

Приводя подобные, получим:

$$\varphi(y_N) h + (k_N + \alpha_N h - k_N \xi_N - k_N \eta_N) y_N - h \alpha_N T_0 = 0$$

3. Опишите алгоритм применения метода прогонки, если при $x = 0$ краевое условие квазилинейное (как в настоящей работе), а при $x = l$, как в п. 2.

Найдем начальные прогоночные коэффициенты по формулам:

$$\xi = \frac{-M_0}{P_0}$$

$$\eta = \frac{-K_0}{P_0}$$

коэффициенты M_0, P_0, K_0 были получены в лекции №7. Далее, находим последующие прогоночные коэффициенты:

$$\xi_{n+1} = \frac{C_n}{B_n - A_n \xi_n}$$

$$\eta_{n+1} = \frac{F_n + A_n \eta_n}{B_n - A_n \xi_n}$$

Из уравнения, полученного в п. 2., можем получить y_N (решив это уравнение). По прогонной формуле можем найти все значения неизвестных y_N

$$y_n = \xi_{n+1}y_{n+1} + \eta_{n+1}$$

4. Опишите алгоритм определения единственного значения сеточной функции y_p в одной заданной точке p . Использовать встречную прогонку, т.е. комбинацию правой и левой прогонок.

1. Вычислим начальные прогоночные коэффициенты:

Для правой прогонки:

$$\xi = \frac{-M_0}{P_0}$$

$$\xi = \frac{-K_0}{P_0}$$

Для левой прогонки:

$$\alpha_{N-1} = \frac{-M_N}{K_N}$$

$$\beta_{N-1} = \frac{-P_N}{K_N}$$

2. Найдем прогоночные коэффициенты:

Для левой прогонки:

$$\xi_{n+1} = \frac{C_n}{B_n - A_n\xi_n}$$

$$\eta_{n+1} = \frac{F_n + A_n\eta_n}{B_n - A_n\xi_n}$$

Для правой прогонки:

$$\alpha_{n-1} = \frac{A_n}{B_n - C_n\alpha_n}$$

$$\beta_{n-1} = \frac{F_n + C_n\beta_n}{B_n - C_n\alpha_n}$$

3. Левые и правые прогонки:

$$y_n = \xi_{n+1}y_{n+1} + \eta_{n+1}$$

$$y_n = \alpha_{n-1}y_{n+1} + \beta_{n-1}$$

4. Выразим y_p :

$$y_{p-1} = \xi_p y_p + \eta_p$$

$$y_p = \alpha_{p-1} y_{p-1} + \beta_{p-1}$$

$$y_p = \frac{\xi_{n+1} \beta_n + \eta_{n+1}}{1 - \xi_{n+1} \alpha_n}$$