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»

Лабораторная работа № 3

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

Студент Брянская Е.В.

Группа ИУ7-62Б

Оценка (баллы) _______________

Преподаватель Градов В.М.

Москва.
2021 г.

Задание

Тема. 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.

Исходные данные.

1. Задана математическая модель.
   Квазилинейное уравнение для функции T(x)

$$\frac{d}{dx}\left(\lambda(T)\frac{dT}{dx}\right) - 4 \cdot k(T) \cdot n_p^2 \cdot \sigma \cdot (T^4 - T_0^4) = 0 \tag{1}$$

   Краевые условия:

$$\begin{cases} x = 0, -\lambda(T(0))\frac{dT}{dx} = F_0, \\ x = l, -\lambda(T(l))\frac{dT}{dx} = \lambda(T(l) - T_0) \end{cases} \tag{2}$$

2. Функции $\lambda(T), k(T)$ заданы таблицей

Таблица 1

T, K	λ, Вт/(см К)		T, K	k, 1/см
300	$1.36 \cdot 10^{-2}$		293	$2.0 \cdot 10^{-2}$
500	$1.63 \cdot 10^{-2}$		1278	$5.0 \cdot 10^{-2}$
800	$1.81 \cdot 10^{-2}$		1528	$7.8 \cdot 10^{-2}$
1100	$1.98 \cdot 10^{-2}$		1677	$1.0 \cdot 10^{-1}$
2000	$2.50 \cdot 10^{-2}$		2000	$1.3 \cdot 10^{-1}$
2400	$2.74 \cdot 10^{-2}$		2400	$2.0 \cdot 10^{-1}$

3. Разностная схема с разностным краевым условием при $x = 0$. Получена в Лекции №7, и может быть использована в данной работе. Самостоятельно надо получить интегро-интерполяционным методом разностный аналог краевого условия при $x = l$, точно так же, как это было сделано применительно к краевому условию при $x = 0$ в указанной лекции. Для этого надо проинтегрировать на отрезке $[x_{N-\frac{1}{2}}, x_N]$ записанное выше уравнение 1 и учесть, что поток $F_N = \alpha_N(y_N - T_0)$, а $F_{N-\frac{1}{2}} = \chi_{N-\frac{1}{2}}(\frac{y_{N-1}-y_N}{h})$

4. Значения параметров для отладки (все размерности согласованы)
   $n_p = 1.4$ – коэффициент преломления,

l = 0.2 см – толщина слоя,

T_0 = 300K – температура окружающей среды,

$\sigma = 5.668 \cdot 10_{-12}$ Вт/(см2К4) - постоянная Стефана- Больцмана,

F_0 = 100 Вт/см2 - поток тепла,

α = 0.05 Вт/(см2 К) – коэффициент теплоотдачи.

5. Выход из итераций организовать по температуре и по балансу энергии, т.е.

$$max\left|\frac{y_n^s - y_n^{s-1}}{y_n^s}\right| \leq \varepsilon_1, n = 0, 1, ..., N \tag{3}$$

$$max\left|\frac{f_1^s - f_2^s}{f_1^s}\right| \leq \varepsilon_2 \tag{4}$$

где

$$f_1 = F_0 - \alpha(T(l) - T_0) \tag{5}$$

и

$$f_2 = 4n_p^2\sigma \int_0^l k(T(x))(T^4(x) - T_0^4)dx \tag{6}$$

Выполнение

Задача решается методом разностной аппроксимации.

Обозначим выражение потока как

$$F = \lambda(T)\frac{dT}{dx} \tag{7}$$

Тогда $F_{n+1/2}$ может быть получен использованием интегрирования (7) на интервале $[x_n, x_{n+1}]$ и применением метода средних справа:

$$F_{n+1/2} = \chi_{n+1/2}\frac{t_n - t_{n+1}}{h} \tag{8}$$

где

$$\chi_{n+1/2} = \frac{\lambda_n + \lambda_{n+1}}{2} \tag{9}$$

Аналогичным образом выражается и $F_{n-1/2}$.

Приближённое интегрирование (1) на интервале $[x_{n-1/2}, x_{n+1/2}]$ даёт:

$$-(F_{n+1/2} - F_{n-1/2}) - p_n t_n h + f_n h = 0 \tag{10}$$

где

$$\begin{cases} p_n = p(x_n) = 0 \\ f_n = f(x_n) = -4 \cdot k(t_n) \cdot n_p^2 \cdot \sigma \cdot (t_n^4 - T_0^4) \end{cases} \tag{11}$$

Полученное уравнение использовано для составления разностной схемы для $1 \leq n \leq N-1$:

$$\begin{cases} A_n t_{n-1} - B_n t_n + C_n t_{n+1} = -D_n \\ A_n = \frac{\chi_{n-1/2}}{2} \\ C_n = \frac{\chi_{n+1/2}}{2} \\ B_n = A_n + C_n \\ D_n = f_n h \end{cases} \tag{12}$$

Также получаются разностные схемы в краевых точках $n = 0$ и $n = N$. Решение выводится при помощи простой правой прогонки в несколько итераций. В качестве значений температуры в выражениях используется результат предыдущей итерации (с применением релаксации). Начальное распределение t_n^0 задаётся как T_0. Выход из итераций организован по температуре и балансу энергии:

$$max|\frac{t_n^s - t_n^{s-1}}{t_n^s}| \leq \varepsilon_1, \forall n = 0, 1, ...N. \tag{13}$$

и

$$max|\frac{f_1^s - f_2^s}{f_1^s}| \le \varepsilon_2, \tag{14}$$

где f_1 и f_2 заданы формулами (5) и (6) соответственно.

Результаты работы.

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

   Проинтегрируем исходное выражение на отрезке $[x_{N-1/2}; x_N]$.

$$\int_{x_{N-1/2}}^{x_N} \frac{dF}{dx}dx + \int_{x_{N-1/2}}^{x_N} f(T(x))dx = 0$$

   Вычислим интегралы:

$$-(F_N - F_{N-1/2}) + \frac{h}{4}(f(t_N) + f(t_{N-1/2})) = 0$$

   Подставляя F_N и $F_{N-1/2}$ заданные правым краевым условием:

$$-\alpha(t_N - T0) + \chi_{N-1/2}\frac{t_{N-1} - t_N}{h} + \frac{h}{4}(f(t_N) + f(t_{N-1/2})) = 0$$

   Получаем $K_N t_{N-1} + M_N t_N = P_N$, где

$$\begin{cases} K_N = \frac{\chi_{N-1/2}}{h} \\ M_N = -\alpha - \frac{\chi_{N-1/2}}{h} \\ P_N = -\alpha \cdot T0 - \frac{h}{4}(f(t_N) + f(t_{N-1/2})) \end{cases}$$

   Для удобства умножим все коэффициенты на $-h$. В полученном выражении, учитывая, что $h \to 0$ пренебрегаем членами с h^2.

$$\begin{cases} K_N = -\chi_{N-1/2} \\ M_N = \alpha \cdot h + \chi_{N-1/2} \\ P_N = \alpha \cdot T0 \cdot h \end{cases}$$

2. *График зависимости температуры T(x) от координаты x при заданных выше параметрах.*
   Выяснить, как сильно зависят результаты расчета T(x) и необходимое для этого количество итераций от начального распределения температуры и шага сетки.

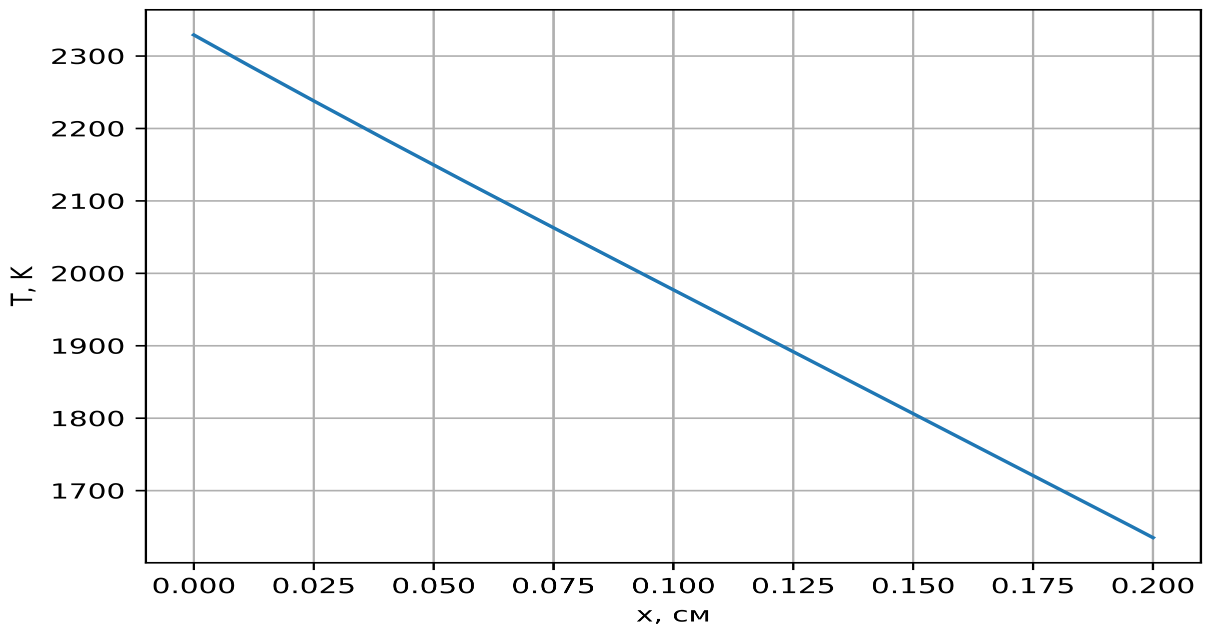

Рис. 1 — Задание 2

При увеличении на порядок шага, использованного в работе, ухудшаются результаты работы программы: меняются значения температур левого/правого края. Уменьшение шага незначительно влияет на результат. Количество итераций практически не меняется. Начальное распределение температуры не влияет на результат. Разница в количестве итераций между достаточно приближенным распределением (от 2400 до 1600) и удаленным распределением (от 300 до 300) равна примерно в 3-4 итерации.

3. *График зависимости* $T(x)$ *при* $F_0 = -10$ *Вт/см2.*

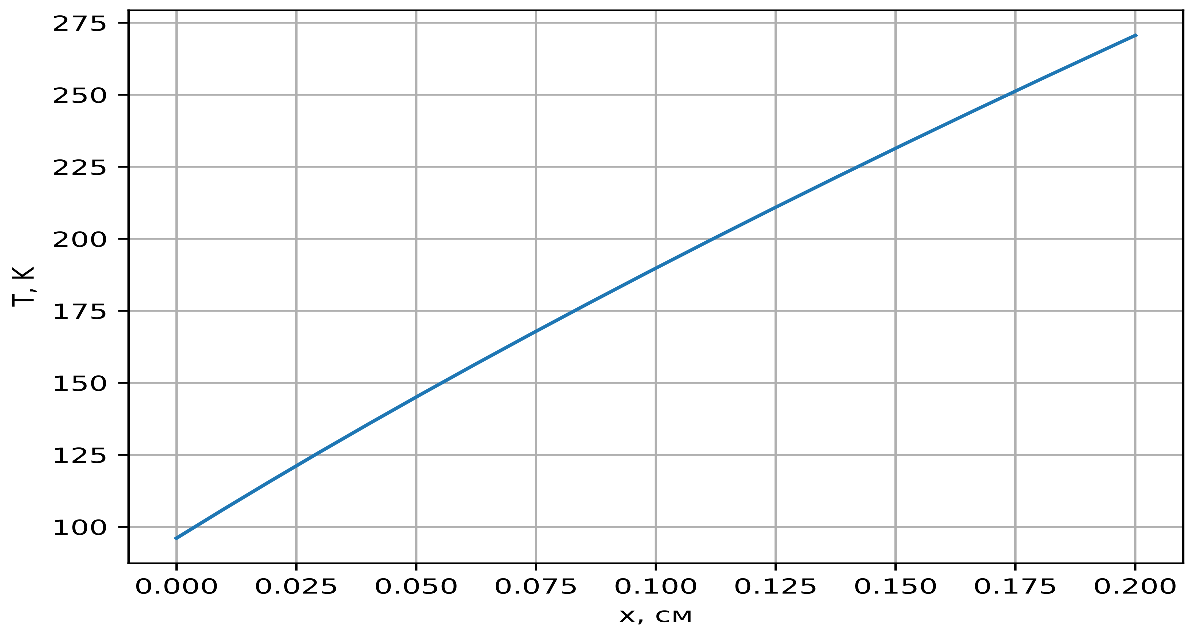

Рис. 2 — Задание 3

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

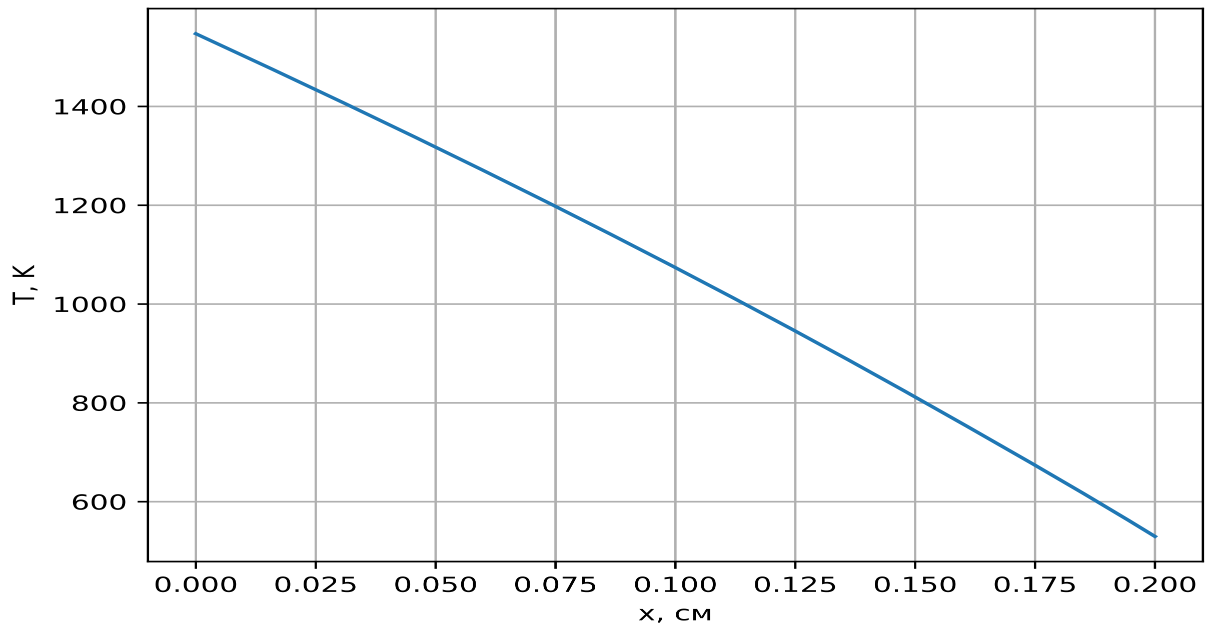

Рис. 3 — Задание 4

При α, увеличенном в 3 раза, можно заметить, что снизился общий уровень температуры графика, температура правой границы приблизилась к T0. Это обусловлено физической природой модели: увеличился теплосъём на правой границе.

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

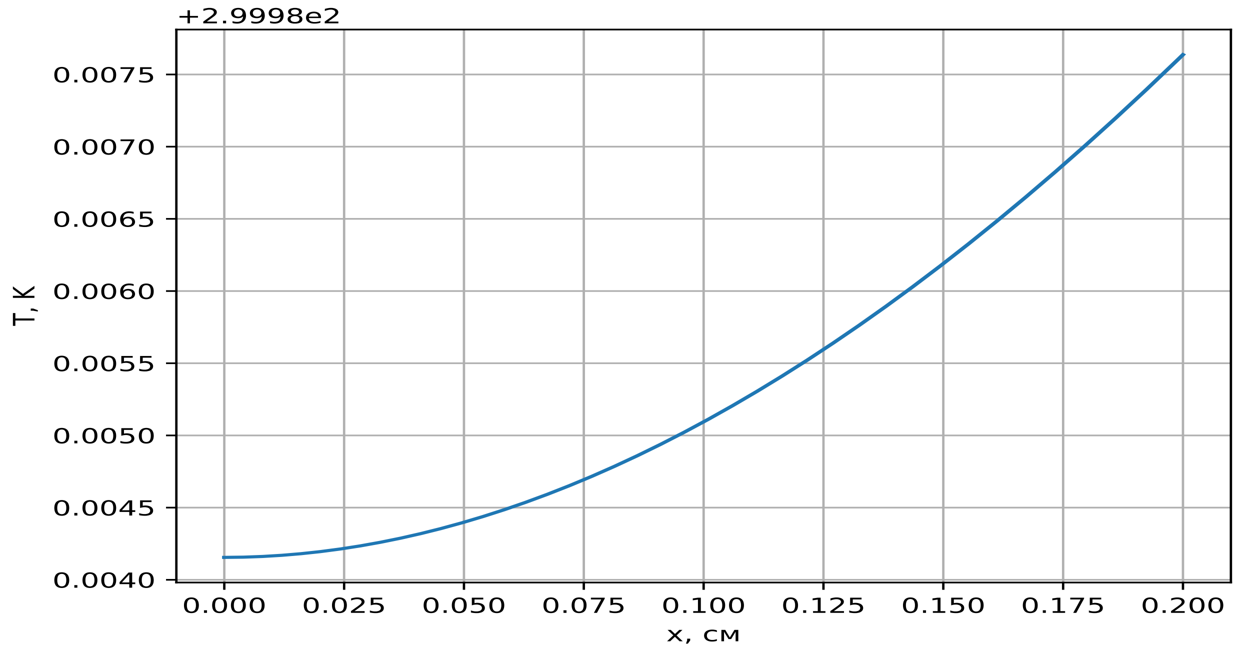

Рис. 4 — Задание 5

6. *Для указанного в задании исходного набора параметров привести данные по балансу энергии, т.е. значения величин*

$$f_1 = F_0 - \alpha(T(l) - T_0) \tag{15}$$

и

$$f_2 = 4n_p^2\sigma \int_0^l k(T(x))(T^4(x) - T_0^4)dx \tag{16}$$

Каковы использованные в работе значения точности выхода из итераций ε_1 *(по температуре) и* ε_2 *(по балансу энергии)?*

ε_1 = 2.5e-4

ε_2 = 2.5e-4

Таблица 2 — Задание 6

Номер итерации	f_1	f_2
0	22.0000	23.1089
1	21.9726	22.5581
2	21.9020	22.1540
3	21.8067	21.8528
4	21.6985	21.6249
5	21.5856	21.4501
6	21.4729	21.3143
7	21.3639	21.2074
8	21.2604	21.1222
9	21.1635	21.0534
10	21.0738	20.9974
11	20.9914	20.9511
12	20.9162	20.9125

Вопросы при защите лабораторной работы

1. *Какие способы тестирования программы можно предложить?*

   Установив $F_0 = 0$, можно протестировать уровень температуры, который должен с небольшой погрешностью равняться T_0, также при данном условии можно изменить сам T_0, чтобы удостовериться, что это происходит при всех температурах.

   Проверка поведения графика при отрицательном/положительном F_0 (значение температуры должны строго возрастать/убывать).

   Изменение α. Увеличение должно приводить к тому, что $T(l) \to T_0$.

   Проверка сходимости решения при различных начальных распределениях и ε (коэффициент релаксации).

2. *Получите простейший разностный аналог нелинейного краевого условия при $x = l, -k(l)\frac{dT}{dx} = \alpha_N(T(l) - T_0) + \phi(T)$. где $\phi(T)$ - заданная функция. Производную аппроксимируйте односторонней разностью.*

   Применяя аппроксимацию односторонней разностью получаем:

$$-k(l)\frac{t_N - t_{N-1}}{h} = \alpha_N(t_N - T_0) + \phi(t_N) \tag{17}$$

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*

Применяется правая прогонка, то есть, коэффициенты определяются слева направо, значение функции справа налево. Так как правое краевое условие зависит от T, прогонка используется в несколько итераций, в каждой итерации s в качестве значения T используется $t_{\varepsilon n}^{s-1}$. При вычислении значения t_N используется полученные прогонкой коэффициенты и $t_{\varepsilon N}^{s-1}$.

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 (лекция №8). Оба краевых условия линейные.*

Левая прогонка:

$$y_n = \varepsilon_{n+1} y_{n+1} + \eta_{n+1} \tag{18}$$

Правая прогонка:

$$y_n = \alpha_{n-1} y_{n-1} + \beta_{n-1} \tag{19}$$

В правой прогонке при $n = p$ получаем выражение:

$$-\alpha_{p-1} y_{p-1} + y_p = \beta_{p-1} \tag{20}$$

Данные выражения можно применить как правое краевое условие для левой прогонки:

$$\begin{cases} K_p = -\alpha_{p-1} \\ M_p = 1 \\ P_p = \beta_{p-1} \end{cases}$$

Применяя составленное краевое условие с коэффициентами, полученными левой прогонкой получаем:

$$y_p = \frac{P_p - K_p \cdot \eta_p}{M_p + K_p \cdot \varepsilon_p} = \frac{\beta_{p-1} + \alpha_{p-1} \cdot \eta_p}{1 - \alpha_{p-1} \cdot \varepsilon_p} \tag{21}$$

По условию оба краевых условия - линейные, поэтому прогонку можно начинать с обеих сторон. В левой прогонке коэффициенты нужно вычислить от 0 до p, в правой от N до $p - 1$.

Код программы

Листинг 1 — Лабораторная работа №3

```
import lab_02 as interpolate
import matplotlib.pyplot as plt

np = 1.4
l = 0.2
T0 = 300
sigma = 5.668e-12
F0 = 100
alpha = 0.05

step = 2e-4
N = round(l / step) + 1
max_iter = 20
e1, e2 = 2.5e-4, 2.5e-4

t = [0] * N
t_balanced = [0] * N

k_relax = 0.1

table_k = [2.0e-2, 5.0e-2, 7.8e-2, 1.0e-1, 1.3e-1, 2.0e-1]
table_kt = [293, 1278, 1528, 1677, 2000, 2400]

table_lambda = [1.36e-2, 1.63e-2, 1.81e-2, 1.98e-2, 2.50e-2, 2.74e-2]
table_lambdat = [300, 500, 800, 1100, 2000, 2400]


def draw_graphs():
  x = [i * step for i in range(0, N)]

  plt.plot(x, t)
  plt.xlabel("x, см")
  plt.ylabel("T, K")
  plt.grid()
  plt.show()

def k(t):
```

```
    return interpolate.interpolation(t, 2, table_kt, table_k)

def lmbd(t):
    return interpolate.interpolation(t, 2, table_lambdat, table_lambda)

def hee_minus(n):
    return (lmbd(t_balanced[n]) + lmbd(t_balanced[n - 1])) / 2

def hee_plus(n):
    return (lmbd(t_balanced[n]) + lmbd(t_balanced[n + 1])) / 2

def f(n):
    return -4 * k(t_balanced[n]) * np**2 * sigma * (t_balanced[n]**4 - T0
     **4)

def A(n):
    return hee_minus(n) / step
def B(n):
    return A(n) + C(n)
def C(n):
    return hee_plus(n) / step
def D(n):
    return f(n) * step

def K0():
    return hee_plus(0)
def M0():
    return -hee_plus(0)
def P0():
    f12 = (f(0) + f(1)) / 2
    return step * F0 + step**2 / 4 * (f12 + f(0))
def KN():
    return alpha * step + lmbd(N - 1)
def MN():
    return -lmbd(N - 1)
def PN():
    return alpha * T0 * step

def forward_move():
    eps = [0] * N
```

```
  eta = [0] * N
  eps[0] = - M0() / K0()
  eta[0] = P0() / K0()

  for i in range(0, N - 1):
    eps[i + 1] = C(i) / (B(i) - A(i) * eps[i])
    eta[i + 1] = (A(i) * eta[i] + D(i)) / (B(i) - A(i) * eps[i])
  return eps, eta

def back_move(eps, eta):
  t2 = [0] * N
  t2[N - 1] = (PN() - MN() * eta[N - 1]) / (KN() + MN() * eps[N - 1])
  for i in range(N - 1, 0, -1):
    t2[i - 1] = eps[i] * t2[i] + eta[i]
  return t2

def init_t_values(begin, end):
  global t, t_balanced
  temp_step = (end - begin) / (N - 1)

  for i in range(N):
    t[i] = begin + temp_step * i
    t_balanced[i] = t[i]

def temperature_check(t_old, t_new):
  for i in range(N):
    if abs((t_new[i] - t_old[i]) / t_new[i]) > e1:
      return False
  return True

def energy_check(j):
  f1 = F0 - alpha * (t_balanced[N - 1] - T0)
  f2 = 0
  for i in range(N):
    f2 += k(t_balanced[i]) * (t_balanced[i] ** 4 - T0 ** 4)
  f2 *= 4 * np**2 * sigma
  f2 *= l / N
  return abs((f1 - f2) / f1) <= e2
```

```
def do_balance():
    global t_balanced
    for i in range(N):
        t_balanced[i] += k_relax * (t[i] - t_balanced[i])

def main():
    global t
    init_t_values(2400, 1600)

    for j in range(max_iter):
        eps, eta = forward_move()
        t2 = back_move(eps, eta)

        t_old, t = t, t2
        if energy_check(j) and temperature_check(t_old, t):
            break
        do_balance()
    draw_graphs()


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