Введение

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

Задача

Задана математическая модель. Уравнение для функции *T(x)*

$$\frac{d}{dx}\left(gamma(x)\frac{dT}{dx}\right)-4k(T)n_p^2*sigma*(T^4-T_0^4)=0 \quad (1)$$

Краевые условия

$$\begin{cases} x=0,-gamma(T(0))\dfrac{dT}{dx}=F_0, \\ x=l,-gamma(T(l))\dfrac{dT}{dx}=\alpha(T(l)-T_0) \end{cases} \quad (2)$$

2. Функции gamma(T), k(T) заданы таблицей (3,4)

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

Разностная схема с разностным краевым условием при x = 0 была получена в лекции №7:

$$A_n y_{n+1}-B_n y_n+C_n y_{n+1}=-D_n, 1\le n\le N-1, \quad (5)$$

где

$$A_n=\frac{X_{n+1/2}}{h}$$

$$C_n=\frac{X_{n-1/2}}{h}$$

$$B_n=A_n+C_n+p_n h$$

$$D_n=f_n h$$

Система (5) совместно с краевыми условиями решается методом прогонки.

Для величин $X_{n+\frac{1}{2}}$ можно получить различные приближенные выражения, численно вычисляя интеграл методом трапеций или методом средних.

Для вычислений будет использоваться метод средних:

$$X_{n\pm\frac{1}{2}}=\frac{k_n+k_{n\pm1}}{2} \quad (6)$$

Разностный аналог краевого условия при $x=0$:

$$\left(X_{\frac{1}{2}}+\frac{h^2}{8}p_{\frac{1}{2}}+\frac{h^2}{4}p_0\right)y_0-\left(X_{\frac{1}{2}}-\frac{h^2}{8}p_{\frac{1}{2}}\right)y_1=hF_0+\frac{h^2}{4}\left(f_{\frac{1}{2}}+f_0\right)$$

(7)

Простая аппроксимация:

$$p_{\frac{1}{2}}=\frac{p_0+p_1}{2}$$

$$f_{\frac{1}{2}}=\frac{f_0+f_1}{2}$$

(8,9)

При выполнении лабораторной необходимо учесть, что

$$F_N=a_N\left(y_N-T_0\right)$$

$$F_{N-\frac{1}{2}}=X_{N-\frac{1}{2}}\frac{y_{N-1}-y_N}{h}$$

(10,11)

4. Значения параметров для отладки (все размерности согласованы)

$n_p = 1.4$ – коэффициент преломления,

$l =$ 0.2 см – толщина слоя,

$T_0 = 300$К – температура окружающей среды,

σ=5.668 10^{-12} Вт/(см^{2}К^4)- постоянная Стефана- Больцмана,

F_0=100 Вт/см^2 - поток тепла,

$\alpha = 0.05$ Вт/(см^2 К) – коэффициент теплоотдачи.

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

$$\max\left|\frac{y_n^s - y_n^{s-1}}{y_n^s}\right| \le \varepsilon_1, \text{ для всех } n = 0,1,\ldots N.$$

и

$$\max\left|\frac{f_1^s - f_2^s}{f_1^s}\right| \le \varepsilon_2,$$

где

$$f_1 = F_0 - \alpha(T(l) - T_0) \text{ и } f_2 = 4n_p^2\sigma\int_0^l k(T(x))(T^4(x) - T_0^4)\,dx.$$

Физическое содержание задачи (для понимания получаемых результатов при отладке программы). Сформулированная математическая модель описывает температурное поле T(x) в плоском слое с внутренними стоками тепловой энергии. Можно представить, что это стенка из полупрозрачного материала, например, кварца или сапфира, нагружаемая тепловым потоком на одной из поверхностей (у нас - слева). Другая поверхность (справа) охлаждается потоком воздуха, температура которого равна T0 . Например, данной схеме удовлетворяет цилиндрическая оболочка, ограничивающая разряд в газе, т.к. при больших диаметрах цилиндра стенку можно считать плоской. При высоких температурах раскаленный слой начинает объемно излучать, что описывает второе слагаемое в (1) (закон Кирхгофа). Зависимость от температуры излучательной способности материала очень резкая. При низких температурах стенка излучает очень слабо, второе слагаемое в уравнении(1) практически отсутствует. Функции gamma(T),k(T) являются, соответственно, коэффициентами теплопроводности и оптического поглощения материала стенки.

Листинг

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import integrate
from scipy.interpolate import InterpolatedUnivariateSpline
from decimal import *





def my_alpha(x):
    return interpolate(x, T_lambda_table[0], T_lambda_table[1])


def my_k(x):
    return interpolate(x, T_k_table[1], T_k_table[0])


def p(x):
    return (4 * (my_k(x)**4) * n_p * n_p * sigma)*x


def f(x):
    return 4 * x * n_p * n_p * sigma * T0


def A(n):
    return approc_plus_half(my_alpha, n) / h


def C(n):
    return approc_minus_half(my_alpha, n) / h
```

```
def B(n):
    return A(n) + C(n) + p(n) * h



def D(n):
    return f(n) * h



def approc_plus_half(func, n):
    return (func(n) + func(n + h)) / 2



def approc_minus_half(func, n):
    return (func(n - h) + func(n)) / 2



def left():
    k0 = approc_plus_half(my_k, 0) + (h * h * approc_plus_half(p, 0) / 8) + ((h * h *
p(0)) / 4)
    M0 = -approc_plus_half(my_k, 0) + (h * h * approc_plus_half(p, 0) / 8)
    P0 = h * F0 + (h * h / 4) * (approc_plus_half(f, 0) + f(0))
    return k0, M0, P0



def right():
    kN = (approc_minus_half(my_k, N) / h) - (h * approc_minus_half(p, N) / 8)
    MN = -my_alpha(N) - (approc_minus_half(my_k, N) / h) - (h * p(N) / 4) - (h *
approc_minus_half(p, N) / 8)
    PN = -(h / 4) * (f(N) + approc_minus_half(f, N)) - T0 * my_alpha(N)
    return kN, MN, PN


def interpolate(x, first_ar, second_ar):
    order = 1
    s = InterpolatedUnivariateSpline(first_ar, second_ar, k=order)
    return float(s(float(x)))
```

```python
def find_abs_max_y(y_n, len):
    cur_max = -1e10
    for i in range(len-1):
        if (abs((y_n[i+1]-y_n[i])/y_n[i+1]) > cur_max):
            cur_max = abs((y_n[i+1]-y_n[i])/y_n[i+1])
    if ((cur_max <= eps_1) and (cur_max != -1e10)):
        return False
    print(cur_max)
    return True


def simpson(func, left, right, n):
    h = (right - left) / n
    ans = h / 3
    even = 0.0
    odd = 0.0
    for i in range(1, n):
        if i % 2 == 0:
            even += func(left + h * i)
        else:
            odd += func(left + h * i)
    ans *= (2 * even + 4 * odd + func(left) + func(right))
    return ans


def f_2():
    func = lambda x: x*(my_k(x)**4 - T0**4)
    integral = simpson(func, 0, l, 41)
    return 4 * n_p * n_p * sigma * integral


def f_1():
    return F0 - alpha * (2400 - T0)


def find_abs_max_f(s):
    cur_max = -1e10
    for i in range(1, s):
        if (abs((f_1() ** i - f_2() ** i) / f_1() ** i) > cur_max):
            cur_max = abs((f_1() ** i - f_2() ** i) / f_1() ** i)
    if ((cur_max <= eps_2) and (cur_max != -1e10)):
        return False
    return True
```

```
if __name__ == "__main__":

    h = 1e-4
    N = 0.2
    eps_1 = 1e-2
    eps_2 = 2e-2

    n_p = 1.4
    l = 0.2
    T0 = 300
    sigma = 5.668 * 1e-12
    F0 = 100
    alpha = 0.05
    T_lambda_table = [[300, 500, 800, 1100, 2000, 2400],
                      [1.36 * 1e-2, 1.63 * 1e-2, 1.81 * 1e-2, 1.98 * 1e-2, 2.5 * 1e-2,
2.74 * 1e-2]]
    T_k_table = [[293, 1278, 1528, 1677, 2000, 2400], [2 * 1e-2, 5 * 1e-2, 7.8 * 1e-2,
1 * 1e-1, 1.3 * 1e-1, 2 * 1e-1]]

    cnt  = 0
    y_n = [0]*200
    x = h
    n = 1
    t = [0] * (n + 1)
    while ((cnt < 200) and (find_abs_max_y(y_n, cnt)) and (find_abs_max_f(cnt))):
        k0, M0, P0 = left()
        kN, MN, PN = right()

        eps = [0, -M0 / k0]
        eta = [0, P0 / k0]

        x = h
        n = 1
        while x + h < N:
```

```
        eps.append(C(x) / (B(x) - A(x) * eps[n]))
        eta.append((A(x) * eta[n] + D(x)) / (B(x) - A(x) * eps[n]))
        n += 1
        x += h

    t = [0] * (n + 1)

    t[n] = (PN - MN * eta[n]) / (kN + MN * eps[n])

    for i in range(n - 1, -1, -1):
        t[i] = (eps[i + 1] * t[i + 1] + eta[i + 1])

    y_n[cnt] = t[n]
    cnt +=1
    h = h / 2




x = [i for i in np.arange(0, N, h)]

plt.plot(x, t[:-1])
plt.xlabel("x, cm")
plt.ylabel("temperature, K")
plt.grid()
plt.show()
```

Результаты работы

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

$$F=-gamma(T)\frac{dT}{dx}$$

$$p(x)=4k(T)n_p^2*sigma$$

$$f(x)=4k(T)n_p^2*sigma*T_0^4$$

Тогда (11) можно записать как

$$\frac{-d}{dx}(F)-p(x)+f(x)=0 \quad (12)$$

Проинтегрируем (12) на отрезке $[x_{N-\frac{1}{2}};x_N]$:

$$-\int\limits_{X_{N-\frac{1}{2}}}^{X_N}\frac{dF}{dx}dx-\int\limits_{X_{N-\frac{1}{2}}}^{X_N}p(x)u^4dx+\int\limits_{X_{N-\frac{1}{2}}}^{X_N}f(x)dx=0 \quad (13)$$

Второй и третий интегралы вычислим методом трапеций:

$$-\left(F_N-F_{N-\frac{1}{2}}\right)-\frac{h}{4}\left(p_N y^4{}_N+p_{N-\frac{1}{2}}y^4{}_{N-\frac{1}{2}}\right)+\frac{h}{4}\left(f_N+f_{N-\frac{1}{2}}\right)=0 \quad (14)$$

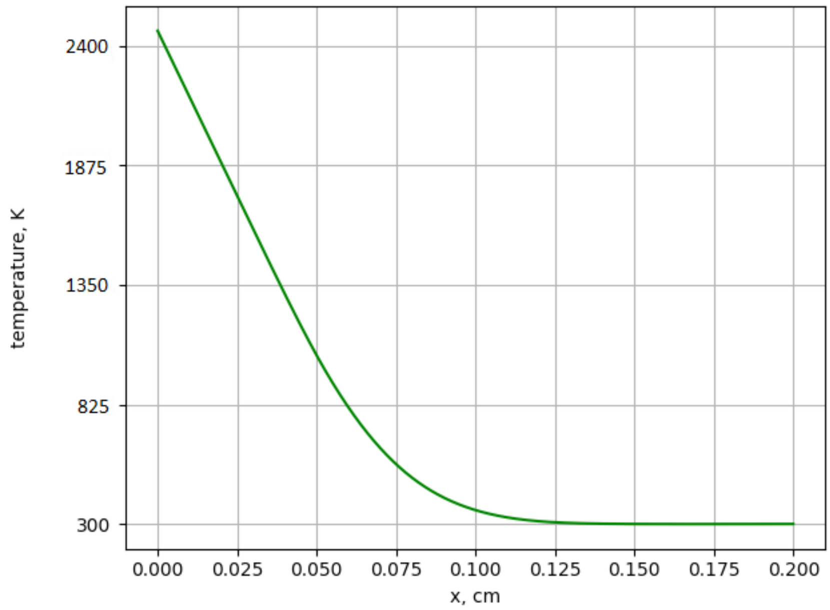
Применив (10, 11) к (14) получим:

$$-\left(a_N(y_N-T_0)-X_{N-\frac{1}{2}}\frac{y_{N-1}-y_N}{h}\right)-\frac{h}{4}\left(p_N y^4{}_N+p_{N-\frac{1}{2}}\frac{y^4{}_N+y^4{}_{N-1}}{2}\right)+\frac{h}{4}\left(f_N+f_{N-\frac{1}{2}}\right)=0$$

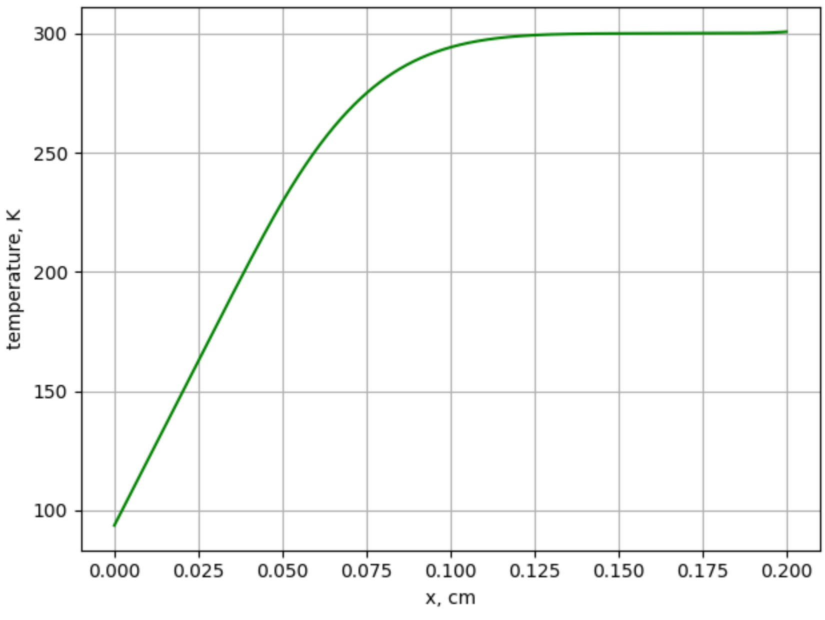
$$X_{N-\frac{1}{2}}\frac{y_{N-1}-y_N}{h}-\frac{h}{4}p_N y_N-\frac{h}{4}p_{N-\frac{1}{2}}\frac{y^4{}_N+y^4{}_{N-1}}{2}=\frac{-h}{4}\left(f_N+f_{N-\frac{1}{2}}\right)+a_N(y_N-T_0)$$

$$y_{N-1}\left(\frac{X_{N-\frac{1}{2}}}{h}-\frac{h}{8}p_{N-\frac{1}{2}}y_{N-1}^3\right)+y_N\left(-a_N-\frac{X_{N-\frac{1}{2}}}{h}-\frac{h}{4}p_N-\frac{h}{8}p_{N-\frac{1}{2}}y_N^3\right)=\frac{-h}{4}\left(f_N+f_{N-\frac{1}{2}}\right)-T_0 a_N$$

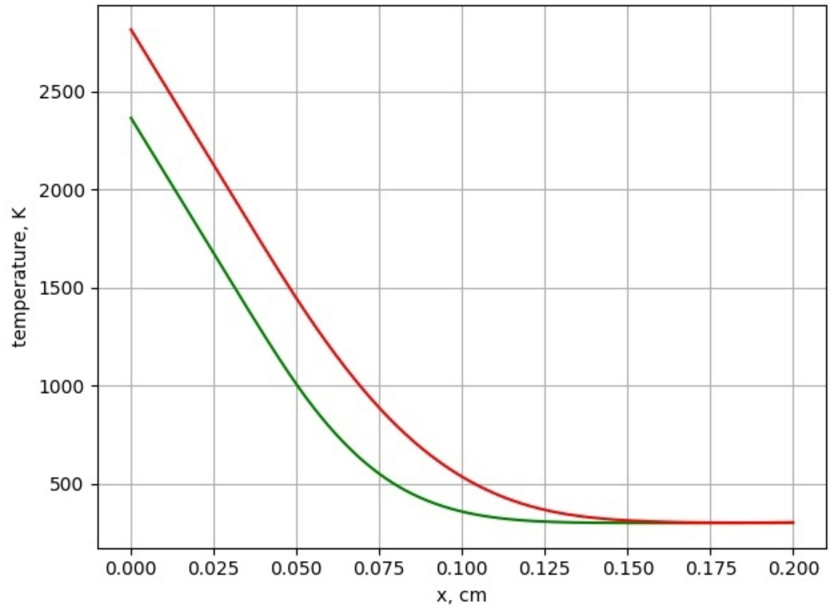
2. График зависимости температуры $T(x)$ от координаты x при заданных выше параметрах.



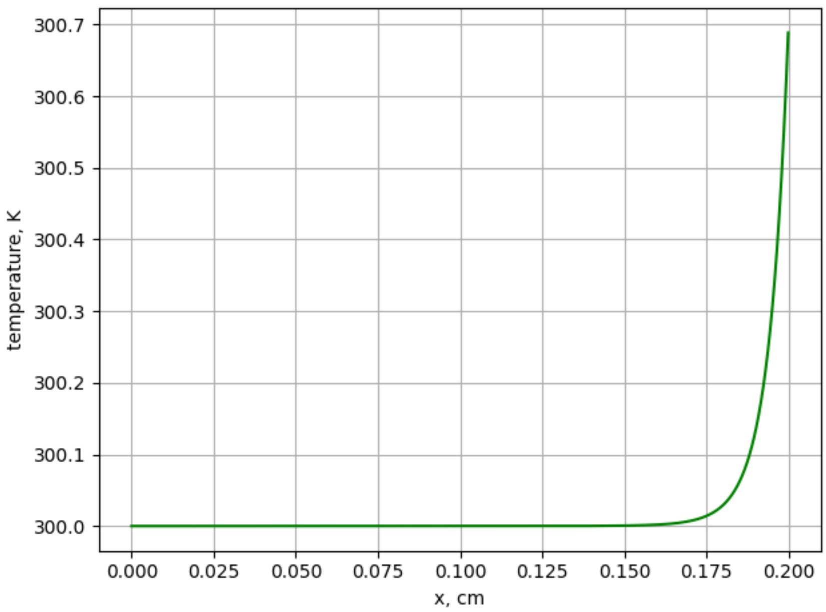
3. График зависимости T(x) при F0 = -10 Вт/см2



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



5. График зависимости T(x) при F0 = 0



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

Точность выхода eps_1 (по температуре) = 0.07

Точность выхода eps_2 (по балансу энергии) = 1.14