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 № *4*

Дисциплина*:* Моделирование

Тема: Программно-алгоритмическая реализация моделей на основе дифференциальных уравнений в частных производных с краевыми условиями II и III рода.

Студент Мирзоян С. А.

Группа ИУ7-65Б

Оценка (баллы) _______________

Преподаватель Градов В.М.

Москва
2020 г.

Цель работы: Получение навыков разработки алгоритмов решения смешанной краевой задачи при реализации моделей, построенных на квазилинейном уравнении параболического типа.

Исходные данные.

1. Уравнение для функции T(x):

$$c(T)\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}(k(T)\frac{\partial T}{\partial x}) - \frac{2}{R}\alpha(x)T + \frac{2T_0}{R}\alpha(x)$$

2. Краевые условия: $$\begin{cases} t = 0, \quad T(x,0) = T_0, \\ x = 0, \quad -k(T(0))\dfrac{\partial T}{\partial x} = F_0, \\ x = l, \quad -k(T(l))\dfrac{\partial T}{\partial x} = \alpha_N\,(T(l) - T_0) \end{cases}$$

3. Разностная схема с разностным краевым условием x = 0

$$\widehat{A_n}\widehat{y_{n-1}} - \widehat{B_n}\widehat{y_n} + \widehat{D_n}\widehat{y_{n+1}} = -\widehat{F_n} \tag{1}$$

Разностный аналог краевого условия при *x*=*l* интегро-интерполяционным методом, (интегрируя на отрезке $[x_{N-1/2}, x_N]$ уравнение (1))

$$\widehat{F_N} = \alpha_N(\widehat{y_N} - T_0)$$

$$\widehat{F_{N-1/2}} = \widehat{\chi_{N-1/2}}\frac{\widehat{y_{N-1}} - \widehat{y_N}}{h}$$

Приведя к общему виду, получаем **(2)**:

$$\left(\frac{h}{4}\widehat{c_N} + \frac{h}{8}\widehat{c_{N-1/2}} + \alpha_N\tau + \frac{\tau}{h}\chi_{N-1/2} + \frac{h}{4}\tau p_N + \frac{h}{8}\tau p_{N-1/2}\right)y_N +$$

$$+\left(\frac{h}{8}\widehat{c_{N-1/2}} - \frac{\tau}{h}\chi_{N-1/2} + \frac{h}{8}\tau p_{N-1/2}\right)\cdot$$

$$\cdot\, y_{N-1} = \frac{h}{4}\widehat{c_N}y_N + \frac{h}{8}\widehat{c_{N-1/2}}(y_N + y_{N-1}) + \tau\alpha_N T_0 + \frac{h}{4}\tau(\widehat{f_N} + \widehat{f_{N-1/2}})$$

Простая аппроксимация:

$$p_{N-1/2} = \frac{p_N + p_{N-1}}{2}, \widehat{f_{N-1/2}} = \frac{\widehat{f_N} + \widehat{f_{N-1}}}{2}, \widehat{c_{N-1/2}} = \frac{\widehat{c_N} + \widehat{c_{N-1}}}{2}$$

Если принять c(u) = 0, сократить τ, формула (2) перейдёт в формулу для разностного краевого условия при $x=l$ из предыдущей лабораторной работы.

Физическое смысл задачи.

1. Сформулированная в данной работе математическая модель описывает нестационарное температурное поле *T(x,t)*, зависящее от координаты *x* и меняющееся во времени.

2. Свойства материала стержня привязаны к температуре, т.е. теплоемкость и коэффициент теплопроводности *c(T), k(T)* зависят от *T* , тогда как в работе №3 *k(x)* зависит от координаты, а *c* = 0.

3. При *x* = 0 цилиндр нагружается тепловым потоком *F(t)* , в общем случае зависящим от времени, а в работе №3 поток был постоянный.

Если в настоящей работе задать поток постоянным, т.е. *F(t) =const*, то будет происходить формирование температурного поля от начальной температуры T_0 до некоторого установившегося (стационарного) распределения *T(x,t)* . Это поле в дальнейшем с течением времени меняться не будет и должно совпасть с температурным распределением *T(x)* , получаемым в лаб. работе №3, если все параметры задач совпадают, в частности, вместо *k(T)* надо использовать *k(x)* из лаб. работы №3. Это полезный факт для тестирования программы.

Если после разогрева стержня положить поток *F(t)* =0, то будет происходить остывание, пока температура не выровняется по всей длине и не станет равной T_0 .

При произвольной зависимости потока *F(t)* от времени температурное поле будет как-то сложным образом отслеживать поток.

Замечание. Варьируя параметры задачи, следует обращать внимание на то, что решения, в которых температура превышает примерно 2000К, физического смысла не имеют и практического интереса не представляют.

Листинг.

```python
from numpy import arange
import matplotlib.pyplot as plt

class Data:
    x0 = 0
    l = 10            # Длина стержня (cm)
    R = 0.5           # Радиус стержня (cm)
    Tenv = 300        # Температура окружающей среды (K)
    F0 = 100         # Плотность теплового потока (W / (cm^2 * K))
    k0 = 0.1         # Коэффициент теплопроводности в начале стержня (W / (cm * K))
    kN = 0.2         # Коэффициент теплопроводности в конце стержня (W / (cm * K))
    alpha0 = 1e-2    # Коэффициент теплоотдачи в начале стержня (W / (cm^2 * K))
    alphaN = 9e-2  # Коэффициент теплоотдачи в конце стержня (W / (cm^2 * K))
    h = 1e-2
    bk = (kN * l) / (kN - k0)
    ak = - k0 * bk
    b_alpha = (alphaN * l) / (alphaN - alpha0)
    a_alpha = - alpha0 * b_alpha


    @staticmethod
    def k(x):
        return Data.ak / (x - Data.bk)

    @staticmethod
    def alpha(x):
        return Data.a_alpha / (x - Data.b_alpha)

    @staticmethod
    def Xn_plus_half(x):
        return (2 * Data.k(x) * Data.k(x + Data.h)) / \
               (Data.k(x) + Data.k(x + Data.h))

```

```
        @staticmethod
        def Xn_minus_half(x):
            return (2 * Data.k(x) * Data.k(x - Data.h)) / \
                   (Data.k(x) + Data.k(x - Data.h))

        @staticmethod
        def p(x):
            return 2 * Data.alpha(x) / Data.R

        @staticmethod
        def f(x):
            return 2 * Data.alpha(x) / Data.R * Data.Tenv


    def thomas_algorithm(A, B, C, D, K0, M0, P0, KN, MN, PN):  # Tridiagonal matrix algorithm
        # Initial values
        xi = [None, - M0 / K0]
        eta = [None, P0 / K0]

        # Straight running
        for i in range(1, len(A)):
            x = C[i] / (B[i] - A[i] * xi[i])
            e = (D[i] + A[i] * eta[i]) / (B[i] - A[i] * xi[i])

            xi.append(x)
            eta.append(e)

        # print(xi)
        # print(eta)

        # Reverse running
        y = [(PN - MN * eta[-1]) / (KN + MN * xi[-1])]

        for i in range(len(A) - 2, -1, -1):
            y_i = xi[i + 1] *  y[0] + eta[i + 1]

            y.insert(0, y_i)

        return y
```

```
    def left_boundary_conditions():
        X_half = Data.Xn_plus_half(Data.x0)
        p1 = Data.p(Data.x0 + Data.h)
        f1 = Data.f(Data.x0 + Data.h)

        p0 = Data.p(Data.x0)
        f0 = Data.f(Data.x0)

        p_half = (p0 + p1) / 2

        K0 = X_half + Data.h * Data.h * p_half / 8 + Data.h * Data.h * p0 / 4
        M0 = Data.h * Data.h * p_half / 8 - X_half
        P0 = Data.h * Data.F0 + Data.h * Data.h * (3 * f0 + f1) / 4

        return K0, M0, P0


    def right__boundary_conditions():
        X_half = Data.Xn_minus_half(Data.l)

        pN = Data.p(Data.l)
        pN1 = Data.p(Data.l - Data.h)
        fN = Data.f(Data.l)
        fN1 = (2 * Data.alpha(Data.l - Data.h)) / Data.R * Data.Tenv

        KN = - (X_half + Data.alphaN * Data.h) / Data.h - Data.h * (5 * pN + pN1) / 16
        MN = X_half / Data.h - Data.h * (pN + pN1) / 16
        PN = - Data.alphaN * Data.Tenv - Data.h * (3 * fN + fN1) / 8

        return KN, MN, PN


    def calc_coefficients():
        A = []
```

```
        B = []
        C = []
        D = []

        for i in arange(Data.x0, Data.l, Data.h):
            An = Data.Xn_minus_half(i) / Data.h
            Cn = Data.Xn_plus_half(i) / Data.h
            Bn = An + Cn + Data.p(i) * Data.h
            Dn = Data.f(i) * Data.h

            A.append(An)
            B.append(Bn)
            C.append(Cn)
            D.append(Dn)

        return A, B, C, D


if __name__ == "__main__":
    a, b, c, d = calc_coefficients()
    # print(a)
    # print(b)
    # print(c)
    # print(d)

    k0, m0, p0 = left_boundary_conditions()
    # print(k0)
    # print(m0)
    # print(p0)

    kN, mN, pN = right__boundary_conditions()
    # print(kN)
    # print(mN)
    # print(pN)

    T = thomas_algorithm(a, b, c, d, k0, m0, p0, kN, mN, pN)
    print(T)
    x = arange(Data.x0, Data.l, Data.h)

    plt.title('Heating the rod')
```

```
        plt.grid(True)
        plt.plot(x, T, 'r', linewidth=0.5)
        plt.xlabel("Length (cm)")
        plt.ylabel("Temperature (K)")

        plt.savefig("plot.png")

        plt.show()
```

Результат работы программы.

$k(T) = a_1(b_1 + c_1 T^{m_1})$,   Вт/см К,

$c(T) = a_2 + b_2 T^{m_2} - \frac{c_2}{T^2}$,   Дж/см^{3}К.

a_1=0.0134, b_1=1, c_1=4.35 10^{-4}, m_1=1,

a_2=2.049, b_2=0.563 10^{-3}, c_2=0.528 10^5, m_2=1.

$\alpha(x) = \frac{c}{x - d}$,

$\alpha_0 = 0.05$ Вт/см^2 К,

$\alpha_N = 0.01$ Вт/см^2 К,

$l =$ 10 см,

$T_0 = 300$К,

$R = 0.5$ см,

$F(t) = 50$ Вт/см^2 (для отладки принять постоянным).

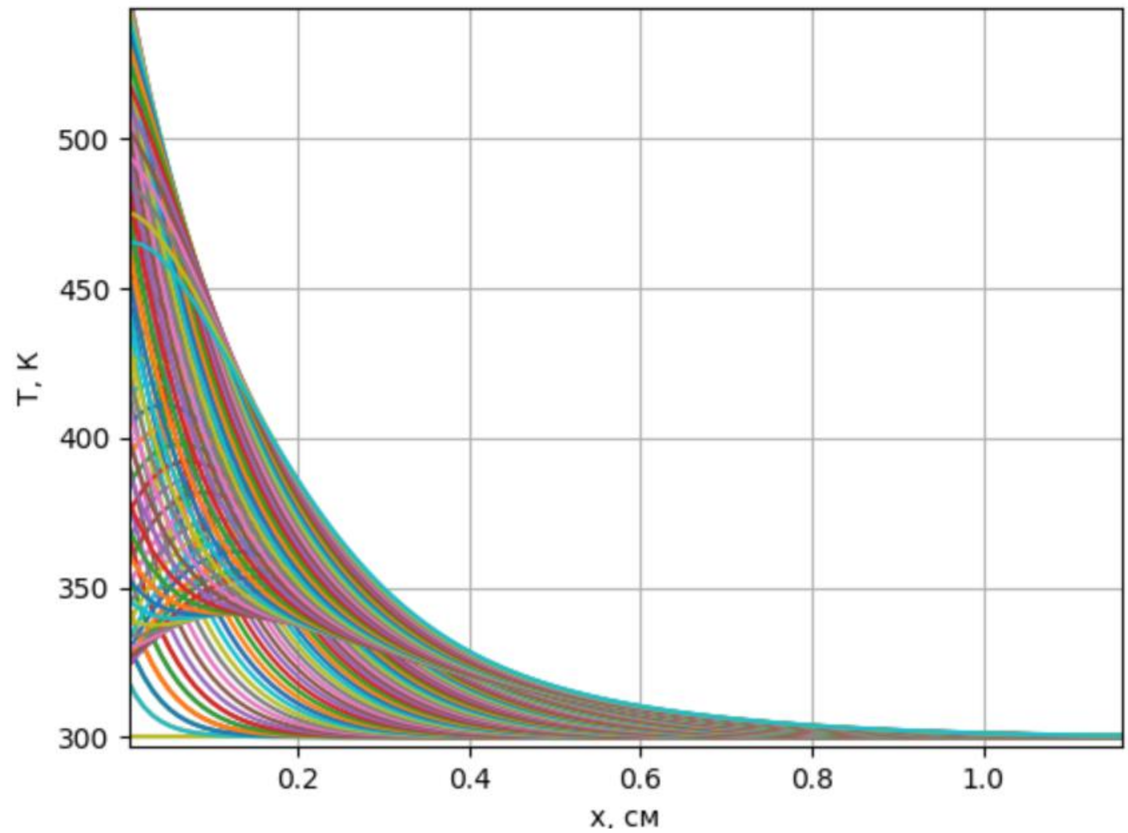

Рисунок 1. График зависимости температуры $T(x)$ от координаты при заданных выше параметрах.

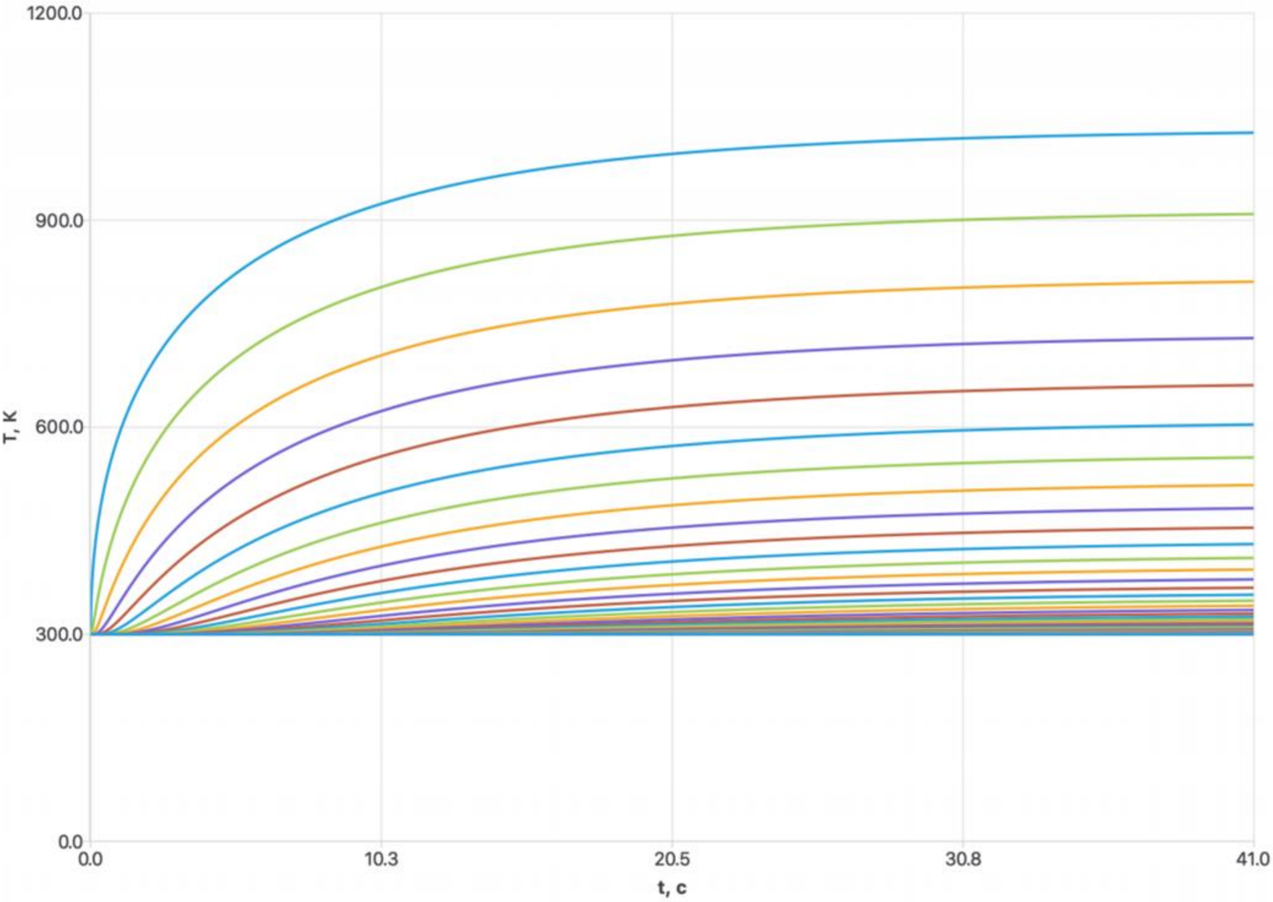

Рисунок 2. График зависимости температуры от времени.

Ответы на вопросы:

1. **Какие способы тестирования программы можно предложить?**
   a. При $F_0 = 0$

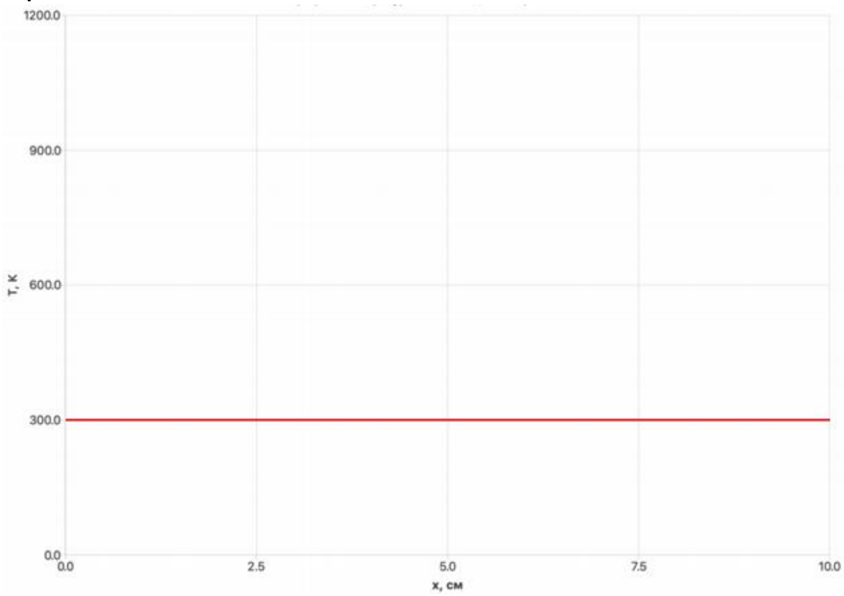

Рисунок 3. График при F = 0 $\frac{\text{Вт}}{\text{см}^2 * K}$

   b. Можно сравнить графики, получившиеся при выходе на стационарном режиме, с графиком из лабораторной работы №3. Для этого заменяем зависимость коэффициента теплопроводности от T на зависимость только от координаты (то есть приводим к тому же условию, что и в предыдущей лабораторной

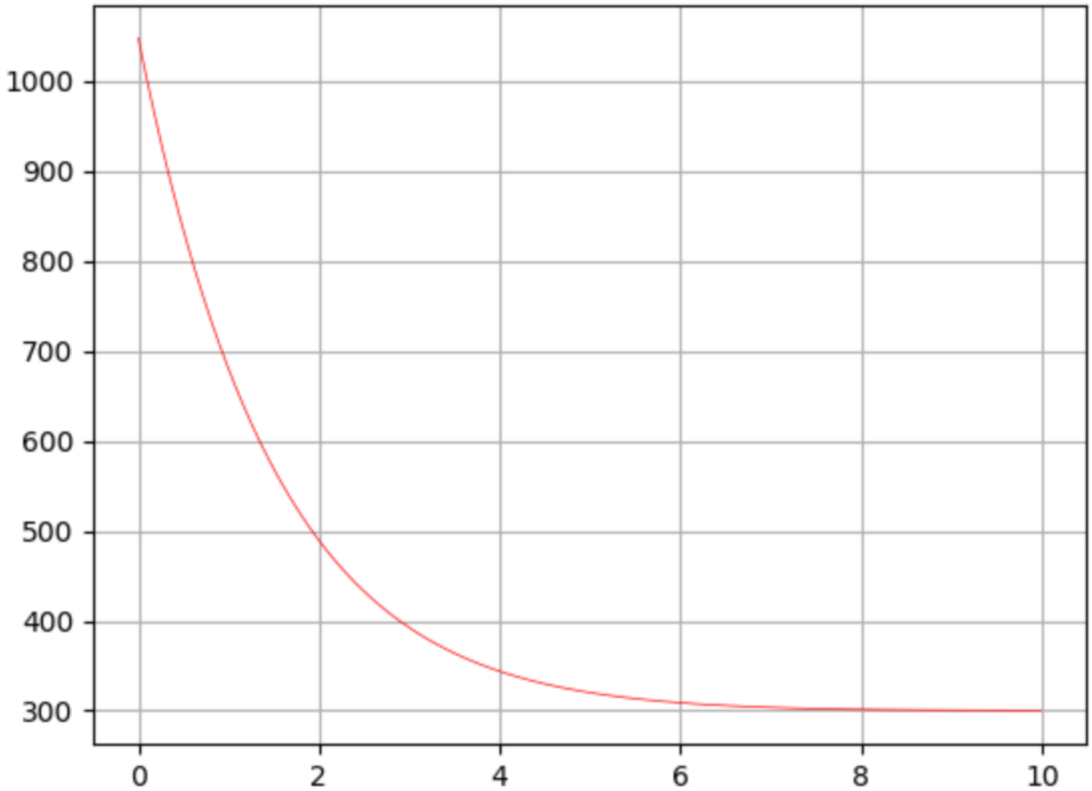

Рисунок 4. График из лабораторной работы 4

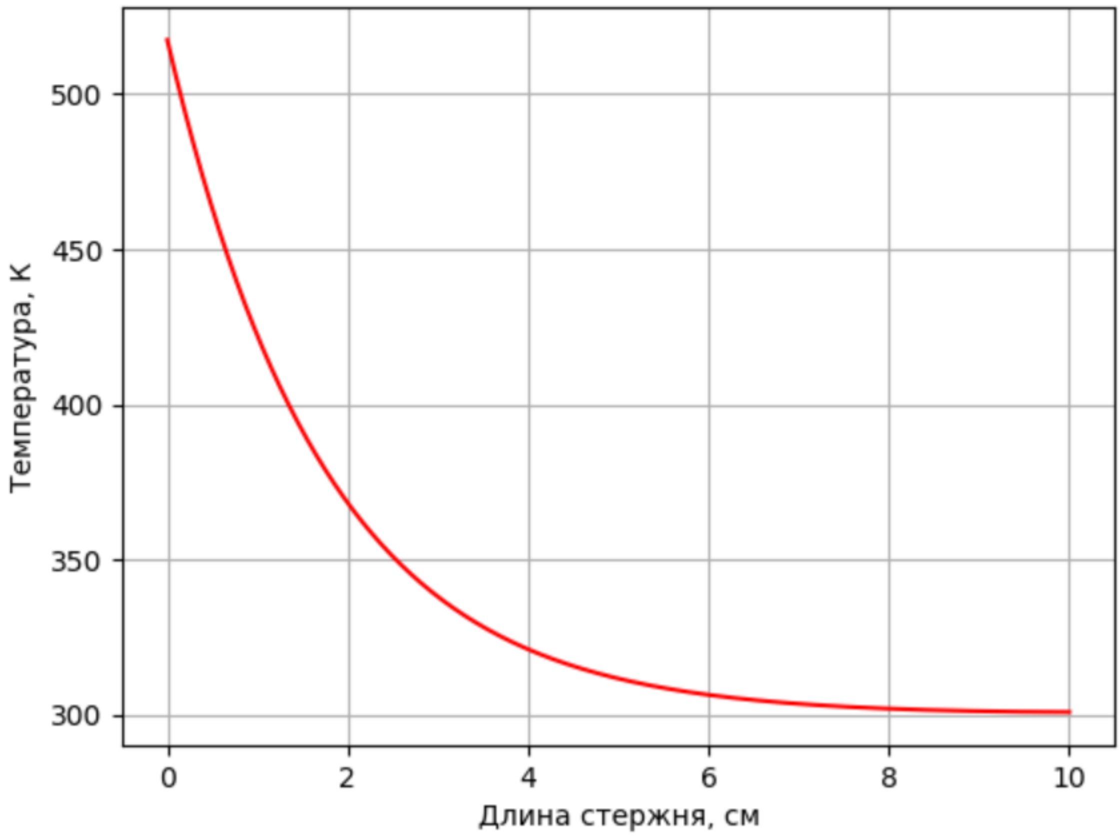

Рисунок 5. График из лабораторной работы 3.

   c. При отрицательном радиусе стержня R<0, должны наблюдаться гармонические колебания.

2. Выполните линеаризацию уравнения (14.8) по Ньютону, полагая для простоты, что все коэффициенты зависят только от одной переменной $\hat{y}_n$. Приведите линеаризованный вариант уравнения и опишите алгоритм его решения. Воспользуйтесь процедурой вывода, описанной в лекции №8.

$$\begin{cases} K_0 y_0 + M_0 y_1 = P_0 \\ A_n y_{n-1} - B_n y_n + D_n y_{n+1} = -F_n, \quad 1 \le n \le N-1 \\ K_N y_N + M_{N-1} y_{N-1} = P_N \end{cases}$$

Линеаризация (по y_{n-1}, y_n, y_{n+1}):

$$\left(A_n y_{n-1} - B_n y_n + D_n y_{n+1} + F_n\right)\Big|_{s} + A^{s-1} \Delta y_{n-1}^{s} +$$

$$+ \left(\frac{\partial A_n}{\partial y_n} y_{n-1} - \frac{\partial B}{\partial y_n} y_n - B_n + \frac{\partial D_n}{\partial y_n} y_{n+1} + \frac{\partial F_n}{\partial y_n}\right)\Big|_{s-1}$$

$$\Delta y_n^s + D_n^{s-1} \Delta y_{n+1}^s = 0$$

Канонический вид:

$$\boxed{A_n \Delta y_{n-1}^s - B_n \Delta y_n^s + D_n \Delta y_{n+1}^s = -F_n \quad 1 \le n \le N-1}$$

где,

$$\begin{cases} A_n = A_n^{s-1} \\ D_n = D_n^{s-1} \\ B_n = \left(-\frac{\partial A_n}{\partial y_n} y_{n-1} + \frac{\partial B_n}{\partial y_n} y_n + B_n - \frac{\partial D_n}{\partial y_n} y_{n+1} + \frac{\partial F_n}{\partial y_n}\right)\Big|_{s-1} \\ F_n = \left(A_n y_{n-1} - B_n y_n + D_n y_{n+1} + F_n\right)\Big|_{s-1} \end{cases}$$

~~Ав~~ Краевые условия: $\Delta y_0^s = 0$, Δy_N^s

Методом прогонки находим все Δy_n^s, затем все значения заданной функции для тек. итерации (S) по формуле:

$$y_n^s = y_n^{s-1} + \Delta y_n^s$$

Зададим начальное приближение y_n^0: в ~~сод~~ соответствие решение y_n с предыдущего шага $t = t_{i-1}$

Условие завершения итераций:

$$\max \left| \frac{\Delta y_n^s}{y_n^s} \right| \le \varepsilon, \quad \text{при } n = \overline{0:N}$$