|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

**Дисциплина: Моделирование**

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

**Студент: Турсунов Ж.Р.**

**Группа ИУ7-66Б**

**Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Преподаватель : Градов В.М.**

Москва. 2021г

1. **АНАЛИТИЧЕСКАЯ ЧАСТЬ**

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

**Исходные данные.**

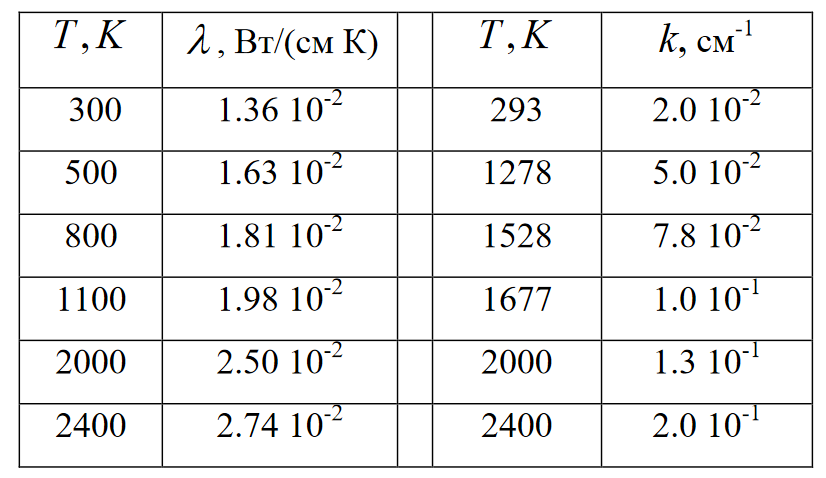
1. Задана математическая модель.

Квазилинейное уравнение для функции

(1)

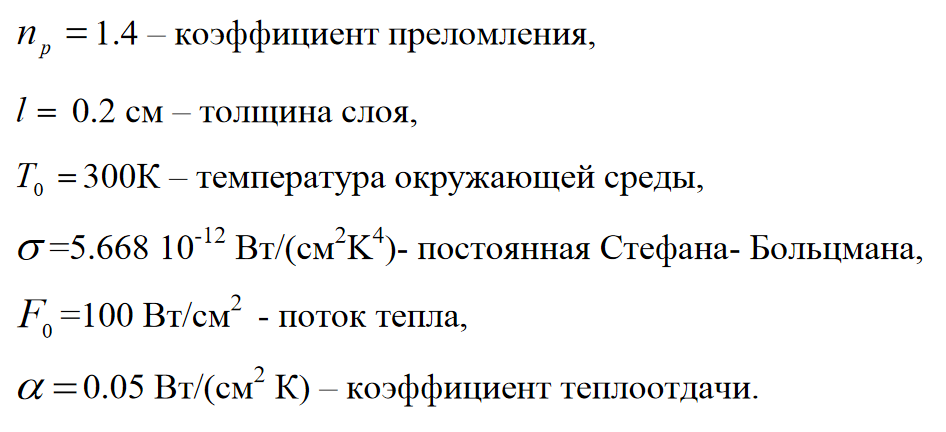
Краевые условия

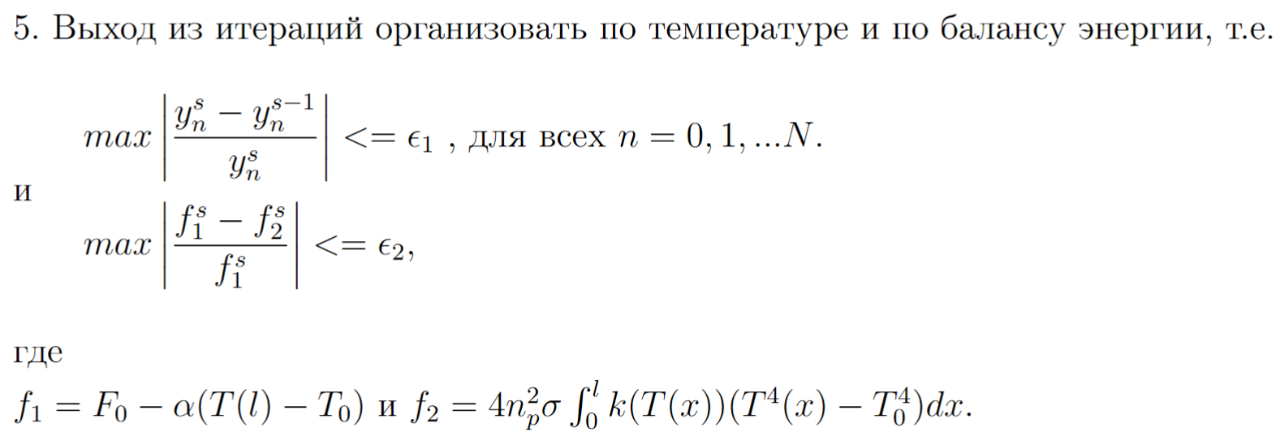
2. Функция заданы таблицей



3. Разностная схема с разностным краевым условием при получена в Лекции и может быть использована в данной работе. Самостоятельно надо получить интегро -интерполяционным методом разностный аналог краевого условия при , точно так же, как это сделано применительно к краевому условию при в указанной лекции. Для этого надо проинтегрировать на отрезке [xN-1/2, xN] выписанное выше уравнение (1) и учесть, что поток , а

4. Значения параметров для отладки (все размерности согласованы)





**Физическое содержание задачи**

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 равнаT0. 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 λ(T), k(T)являются, соответственно, коэффициентами теплопроводности и оптического поглощения материала стенки.

# II. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

ЯП был выбран Python3 из-за простоты работы с графиками и библиотеки matplotlib. Ниже на листингах будет представлена реализация программы:

## Листинг

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import math

from scipy import integrate

from scipy.interpolate import interp1d

plt.rcParams['figure.figsize']=12,8

SAVE\_DIR = './'

def plot\_helper(x, y, xlabel, ylabel, title=False, pic='result.png', save=True):

    plt.scatter(x, y, s=8, c='g')

    plt.plot(x,y,'-', c='r')

    plt.grid()

    plt.ylabel(ylabel)

    plt.xlabel(xlabel)

    if title: plt.title(title)

    if save: plt.savefig(SAVE\_DIR + pic, bbox\_inches = 'tight')

    fig = plt.figure()

n\_p   = 1.4

l     = 0.2

T\_0   = 300

sigma = 5.668e-12

F\_0   = 100

alpha = 0.05

data1 = pd.read\_csv('k\_T.csv')

data2 = pd.read\_csv('lambda\_T.csv')

k\_T = interp1d(data1['T'], data1['k'], fill\_value="extrapolate")

lambda\_T = interp1d(data2['T'], data2['lambda'], fill\_value="extrapolate")

f\_helper = 4 \* n\_p\*\*2 \* sigma

T\_04 = T\_0\*\*4

def T(n):

    if (n \* 2) % 2 == 1:

        n = int(n - 1/2)

        return (T\_list[n] + T\_list[n+1]) / 2

    return T\_list[int(n)]

def k\_plus\_avg(n):

    return (k(n) + k(n+1))/2

def k\_minus\_avg(n):

    return (k(n) + k(n-1))/2

def k(n):

    return lambda\_T(T(n))

def f\_inner(n):

    return k(n) \* (T(n)\*\*4 - T\_04)

def f(n):

    return -f\_helper \* f\_inner(n)

def A(n): return k\_minus\_avg(n) / h

def C(n): return k\_plus\_avg(n) / h

def B(n): return A(n) + C(n)

def D(n): return f(n) \* h

def intergrate(func, a, b, step):

    S = (func(a) + func(b)) / 2

    for x in range(a+step, b, step):

        S += func(x)

    return S \* step

def edge\_condition():

    K0 = k\_plus\_avg(0)

    M0 = -K0

    P0 = h \* F\_0 + h\*\*2 / 4 \* (f(1/2) + f(0))

    KN = - k\_minus\_avg(N)

    MN = alpha \* h - KN

    PN = alpha \* h \* T\_0 + h\*\*2 / 4 \* (f(N-1/2) + f(N))

    return K0, M0, P0, KN, MN, PN

def err\_temperature(t\_old, t\_new):

    return max([abs(1 - t\_old[i] / t\_new[i]) for i in range(len(t\_old))])

def err\_energy\_balance(t\_new):

    f1 = F\_0 - alpha \* (t\_new[N] - T\_0)

    f2 = f\_helper \* intergrate(f\_inner, 0, N, 1) \* h

    return abs(1 - f2/f1)

def thomas\_algorithm():

    K0, M0, P0, KN, MN, PN = edge\_condition()

    # forward

    xi = [None, - M0 / K0]

    eta = [None, P0 / K0]

    for i in range(1, N):

        denominator = (B(i) - A(i) \* xi[i])

        x = C(i) / denominator

        e = (D(i) + A(i) \* eta[i]) / denominator

        xi.append(x)

        eta.append(e)

    # backward

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

    for i in range(N - 1, -1, -1):

        yi = xi[i + 1] \*  y[0] + eta[i + 1]

        y.insert(0, yi)

    return y

def fixed\_iteration(esp1, esp2, max\_loop=20):

    global T\_list

    err1 = err2 = 1

    for i in range(max\_loop):

        if err1 <= esp1 and err2 <= esp2:

            break

        t = thomas\_algorithm()

        err1 = err\_temperature(T\_list, t)

        T\_list = t

        err2 = err\_energy\_balance(T\_list)

        print(i, err1, err2)

h = 0.01

N = int(l // h)

esp1 = 0.01

esp2 = 0.01

x = [i for i in np.arange(0, l+h, h)]

# F = 100, alpha = 0.05

T\_list = [T\_0] \* (N + 1)

F\_0   = 100

alpha = 0.05

fixed\_iteration(esp1, esp2)

plot\_helper(x, T\_list, 'x, cm', 'T, K', pic=1.png')

# F = -10

T\_list = [T\_0] \* (N + 1)

F\_0 = -10

fixed\_iteration(esp1, esp2)

plot\_helper(x, T\_list, 'x, cm', 'T, K', 'F0 = -10', pic='2.png')

# alpha x3

T\_list = [T\_0] \* (N + 1)

F\_0 = 100

alpha = 0.15

fixed\_iteration(esp1, esp2)

plot\_helper(x, T\_list, 'x, cm', 'T, K', 'alpha = 0.15 (x3)', pic='3.png')

# F = 0

T\_list = [T\_0] \* (N + 1)

F\_0 = 0

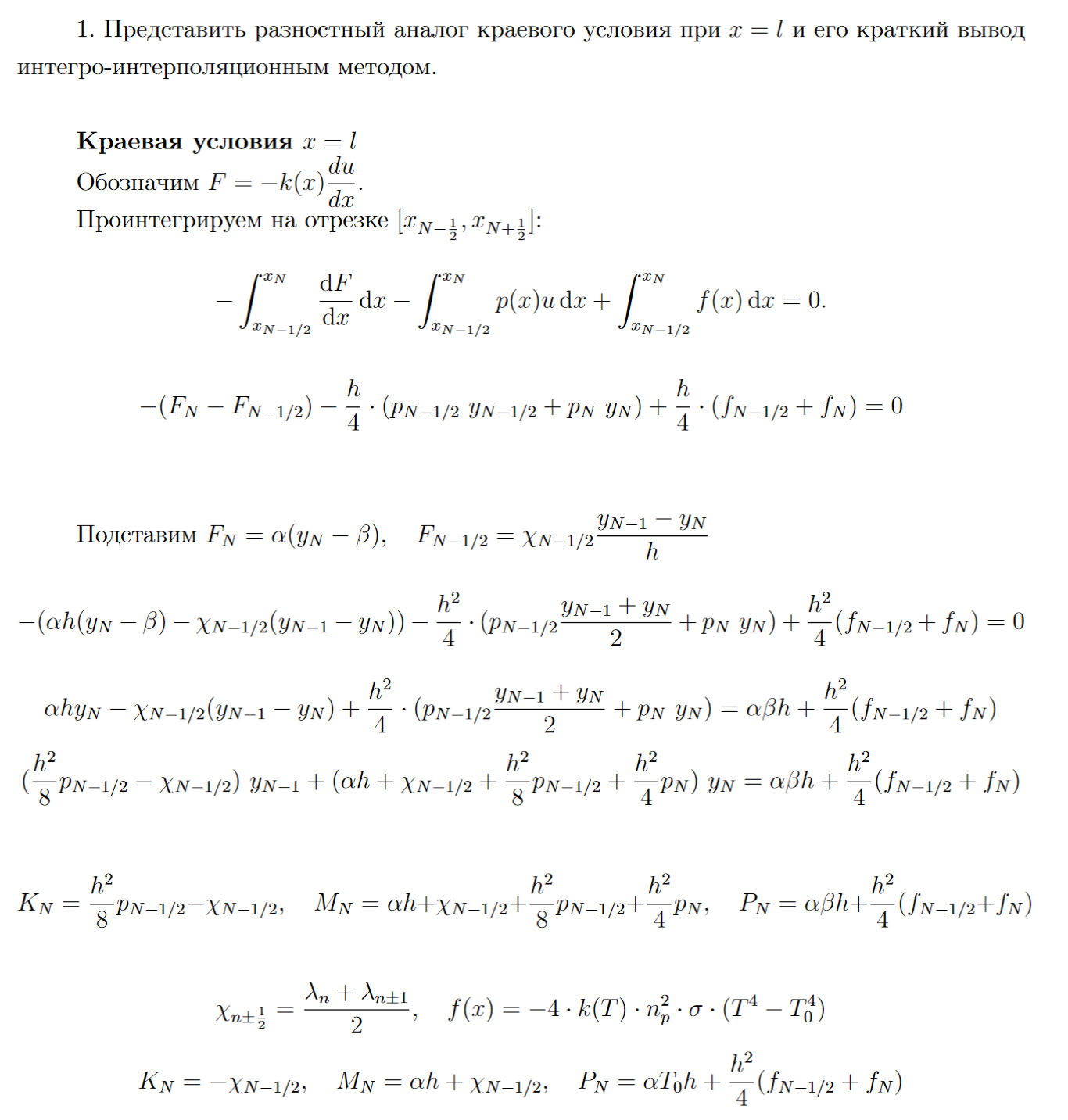
alpha = 0.05

fixed\_iteration(esp1, esp2)

plot\_helper(x, T\_list, 'x, cm', 'T, K', 'F0 = 0', pic='4.png')

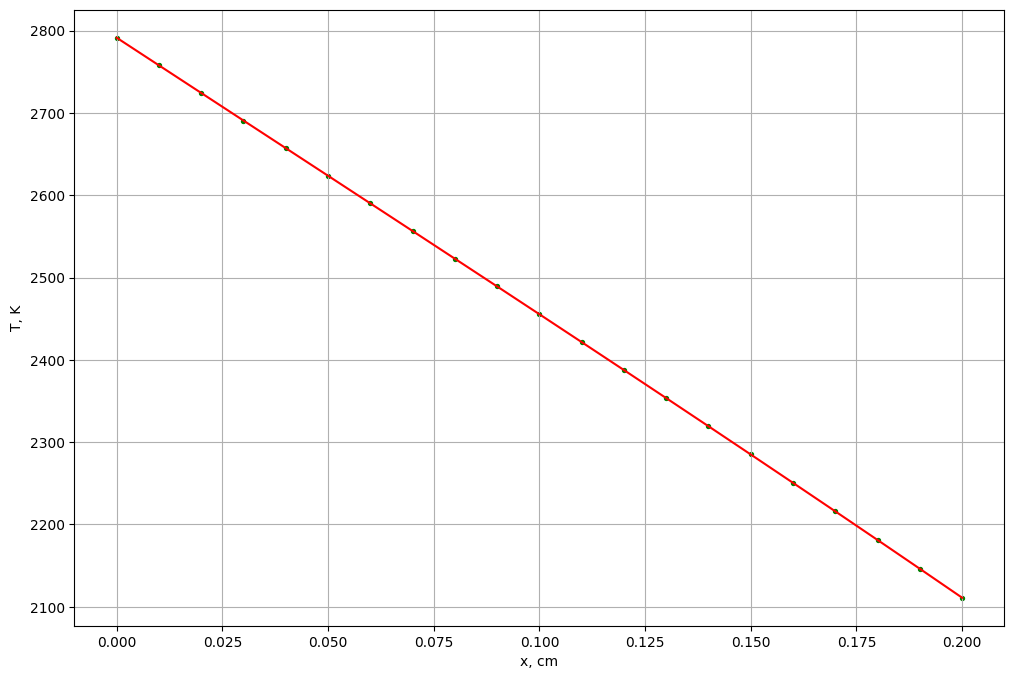
# III. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

В данном разделе будет рассмотрен вывод программы и представлены графики зависимостей.

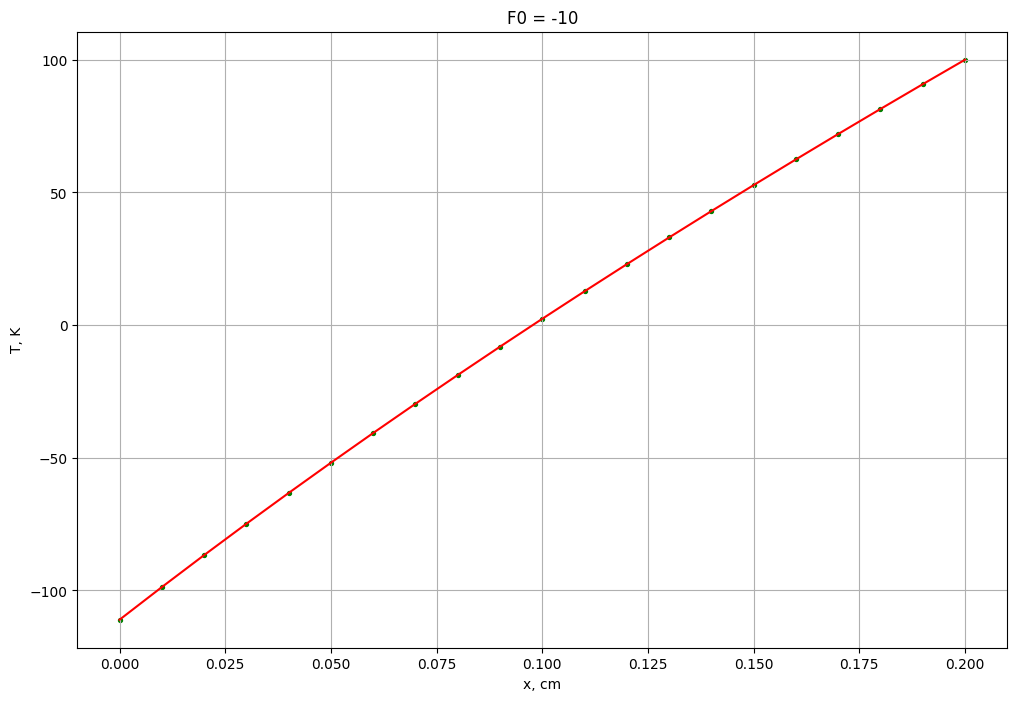


2. График зависимости температуры T(x) от координаты x при заданных выше параметрах.

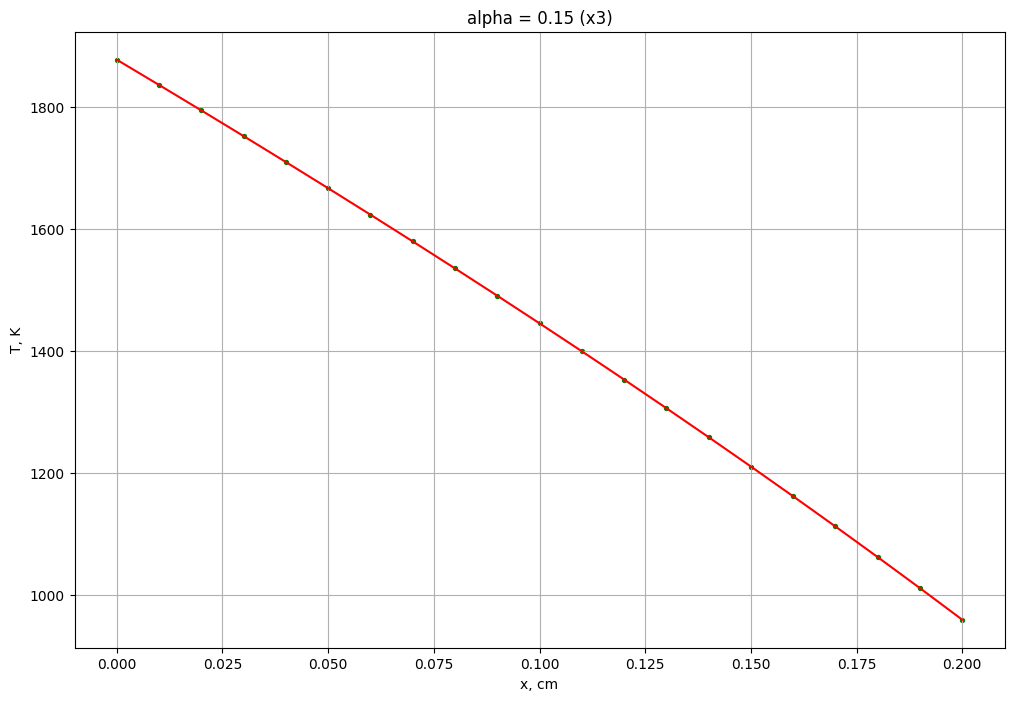
Выяснить, как сильно зависят результаты расчета T(x) и необходимое для этого количество итераций от начального распределения температуры и шага сетки. Количество необходимых повторений не сильно зависит от начального распределения температуры и шага сетки. В этом случае при h= 0.01см и = 0.01 требуется 10 итераций.



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

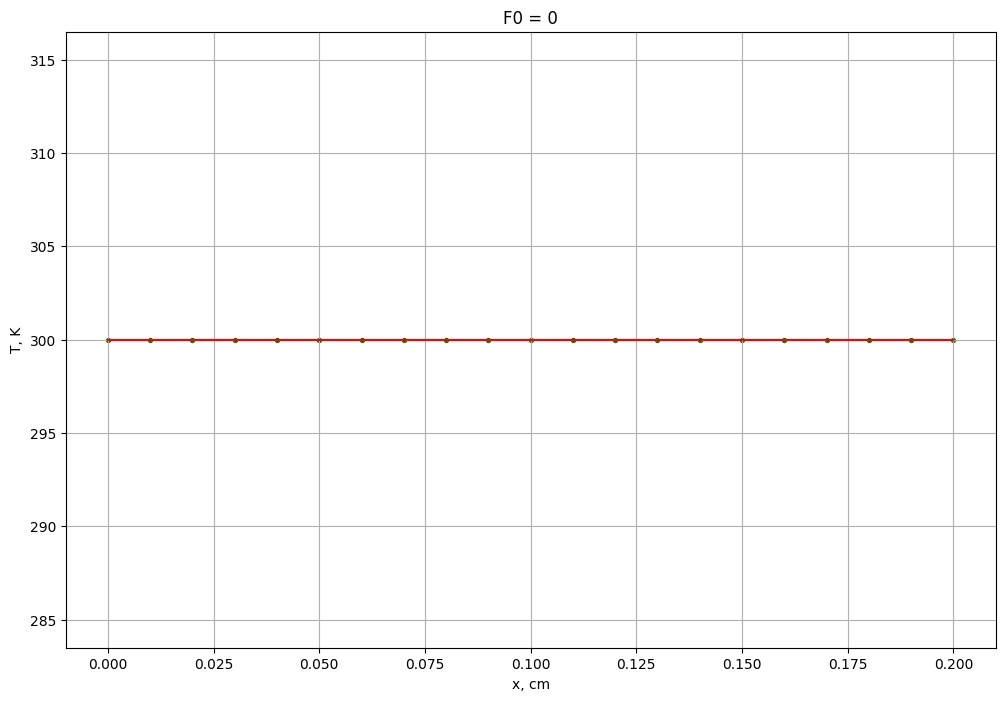


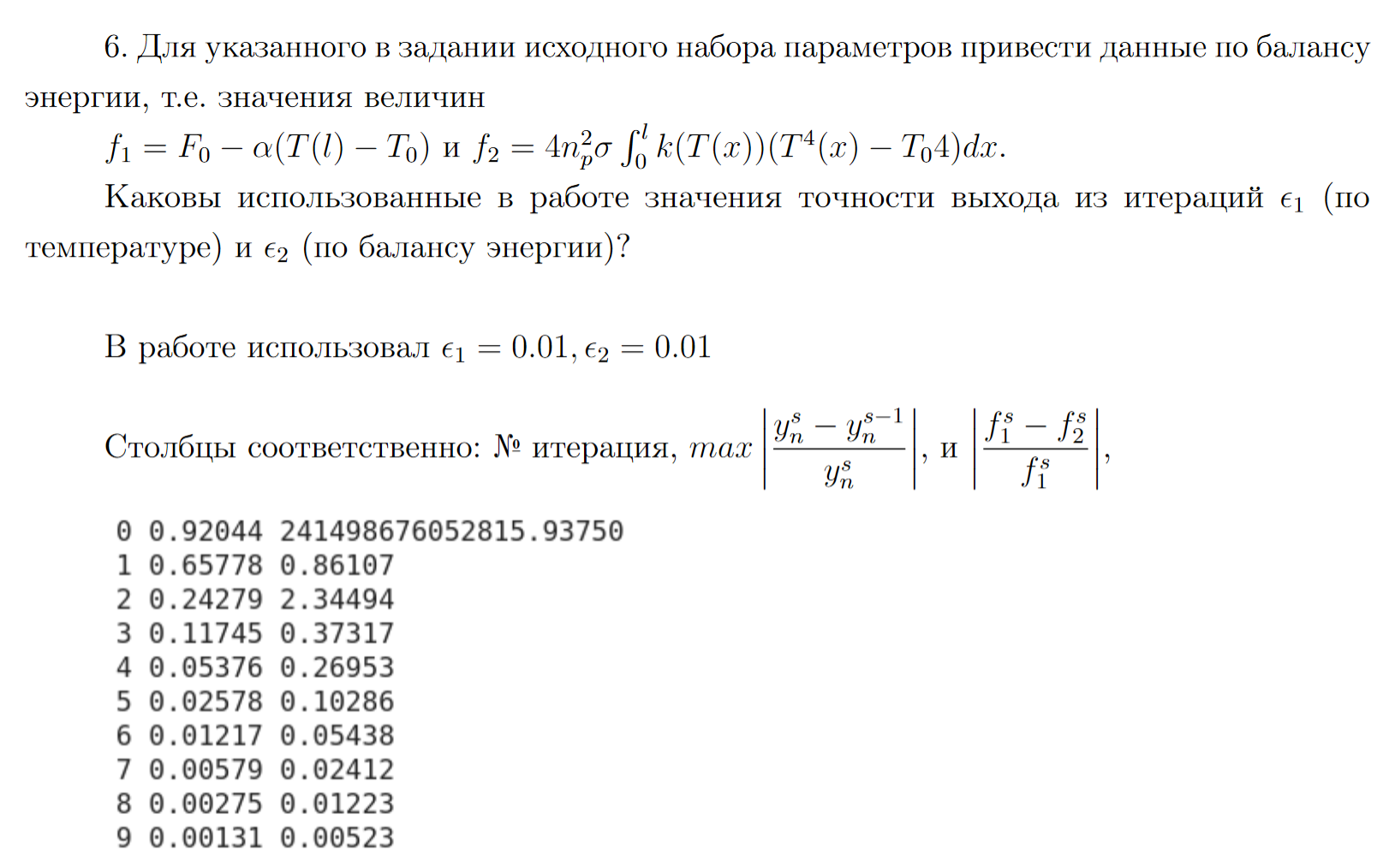
4. График зависимости T(x) при увеличенных значениях α (например, в 3 раза). Сравнить с п.2.

Коэффициент теплоотдачи больше, тепло быстрее передается наружу, более низкая температура

5. График зависимости T(x) при F0 = 0.

**Справка.** В данных условиях тепловое нагружение отсутствует, причин для нагрева нет, температура стержня должна быть равна температуре окружающей среды T0 (разумеется, с некоторой погрешностью, определяемой приближенным характером вычислений)





**IV. ОТВЕТЫ НА ВОПРОСЫ**

