|  |  |
| --- | --- |
| 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 №4**

**По курсу «Моделирование»**

|  |  |
| --- | --- |
| **Тема: «Программно-алгоритмическая реализация моделей на основе дифференциальных уравнений в частных производных с краевыми условиями II и III рода»**  **Студент Горячев В. Г.**  **Группа ИУ7-65Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Градов В. М.** |  |

Москва

2021 г

**Цель работы**

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

**Исходные данные**

1. Задана математическая модель.

Уравнение для функции :

Краевые условия:

В обозначениях уравнения лекции

1. Значения параметров для отладки (все значения согласованы):

**Код программы (значимые участки)**

Константы и подготовка данных:

|  |
| --- |
| length = 10  T0 = 300  R = 0.5  F0 = 100  alpha0 = 0.05  alphaN = 0.01  h = 1e-3  N = int(length / h)  t = 0.5  a1 = 0.0134; b1 = 1; c1 = 4.35e-4; m1 = 1  a2 = 2.049; b2 = 0.563e-3; c2 = 0.528e5; m2 = 1  ab1 = a1 \* b1  ac1 = a1 \* c1  # поскольку m1 и m2 == 1, нет смысла явно прописывать  # операцию возведения в степень  def **k**(T):  # a1 \* (b1 + c1 \* T \*\* m1)  return ab1 + ac1 \* T  def **c**(T):  # a2 - (c2 / (T \* T)) + b2 \* T \*\* m2  return a2 - (c2 / (T \* T)) + b2 \* T  d = (alphaN \* length) / (alphaN - alpha0)  c\_local = -alpha0 \* d  def **alpha**(x):  return c\_local / (x - d) |

Составляющие уравнение, разностную схему функции и элементы алгоритма прогонки

|  |
| --- |
| def **p**(x):  return 2 \* alpha(x) / R  def **f**(x):  # 2 \* alpha(x) / R \* T0  return p(x) \* T0  def **cappa\_plus**(x, step, func):  return (func(x) + func(x + step)) / 2  def **cappa\_minus**(x, step, func):  return (func(x) + func(x - step)) / 2  hdiv8 = h / 8  hdiv4 = h / 4  def **left**(Ts):  cappaPlustc = cappa\_plus(Ts[0], t, c)  cappaPlustk = cappa\_plus(Ts[0], t, k)  cTs0 = c(Ts[0])  ph2 = p(h / 2)  K0 = hdiv8 \* cappaPlustc + hdiv4 \* cTs0 + \  cappaPlustk \* t / h + t \* (hdiv8 \* ph2 + hdiv4 \* p(0))  M0 = hdiv8 \* cappaPlustc - cappaPlustk \* t / h + t \* h \* ph2 / 8  P0 = hdiv8 \* cappaPlustc \* (Ts[0] + Ts[1]) + \  hdiv4 \* cTs0 \* Ts[0] + t \* (F0 + hdiv8 \* (3 \* f(0) + f(h)))  return K0, M0, P0  def **right**(Ts):  cappaMinustc = cappa\_minus(Ts[-1], t, c)  cappaMinustk = cappa\_minus(Ts[-1], t, k)  KN = hdiv8 \* cappaMinustc + hdiv4 \* c(Ts[-1]) + \  cappaMinustk \* t / h + t \* alphaN + \  t \* hdiv8 \* p(length - h / 2) + t \* hdiv4 \* p(length)  MN = hdiv8 \* cappaMinustc - cappaMinustk \* t / h + \  t \* h \* p(length - h / 2) / 8  PN = hdiv8 \* cappaMinustc \* (Ts[-1] + Ts[-2]) + \  hdiv4 \* c(Ts[-1]) \* Ts[-1] + t \* alphaN \* T0 + \  t \* hdiv4 \* (f(length) + f(length - h / 2))  return KN, MN, PN  def **progonka**(A, B, C, D, K0, M0, P0, KN, MN, PN):  \*\*\* |

Итерация и отслеживание условий выхода

|  |
| --- |
| def **cond1**(T, T\_new):  for t, ts in zip(T, T\_new):  if abs((t - ts) / ts) < 1e-4:  return False  return True  def **cond2**(T, Ts):  for t, ts in zip(T, Ts):  if abs((ts - t) / ts) > 1e-2:  return True  return False  def **calc\_iteration**(Ts):  K0, M0, P0 = left(Ts)  KN, MN, PN = right(Ts)  A = np.empty(N + 1, dtype=np.double)  B = np.empty(N + 1, dtype=np.double)  C = np.empty(N + 1, dtype=np.double)  D = np.empty(N + 1, dtype=np.double)  x = np.arange(0, length + h, h)  tdivh = t / h  tmulh = t \* h  for n in range(1, N + 1):  Tn = Ts[n]  cTmulh = c(Tn) \* h  A[n] = tdivh \* cappa\_minus(Tn, t, k)  C[n] = tdivh \* cappa\_plus(Tn, t, k)  B[n] = A[n] + C[n] + cTmulh + p(x[n]) \* tmulh  D[n] = f(x[n]) \* tmulh + cTmulh \* Tn  return progonka(A, B, C, D, K0, M0, P0, KN, MN, PN)  def **calc\_slice**(alpha, ys0, maxIter = 25):  i = 0  ys = calc\_iteration(ys0)  while cond1(ys0, ys) and i < maxIter:  ys0 = ys  ys = (1.0 - alpha) \* ys0 + alpha \* calc\_iteration(ys0)  i += 1  return ys  def **calc\_model**(T):  global F0  time = 0.0; count = 0  res = [T]  Ts = calc\_slice(0.7, T)  res.append(Ts)  time += t  while cond2(T, Ts) and count < 50:  # F0 = 100 + 50 \* cos(time) \*\* 2 # для произвольной ф-ии F(t)  T = Ts  Ts = calc\_slice(0.7, T)  res.append(Ts)  time += t  count += 1  return res, time |

**Вывод краевого условия**

Представить разностный аналог краевого условия при и его краткий вывод интегро-интерполяционным методом.

Уравнение для функции :

Краевые условия:

В обозначениях уравнения лекции

Также введём обозначение:

Разностный аналог краевого условия при x = l получается путем интегрирования уравнения на отрезке , с учётом того, что

Получаем:

Интегрируем аналогично разностному аналогу краевого условия при :

Приведём уравнение к виду :

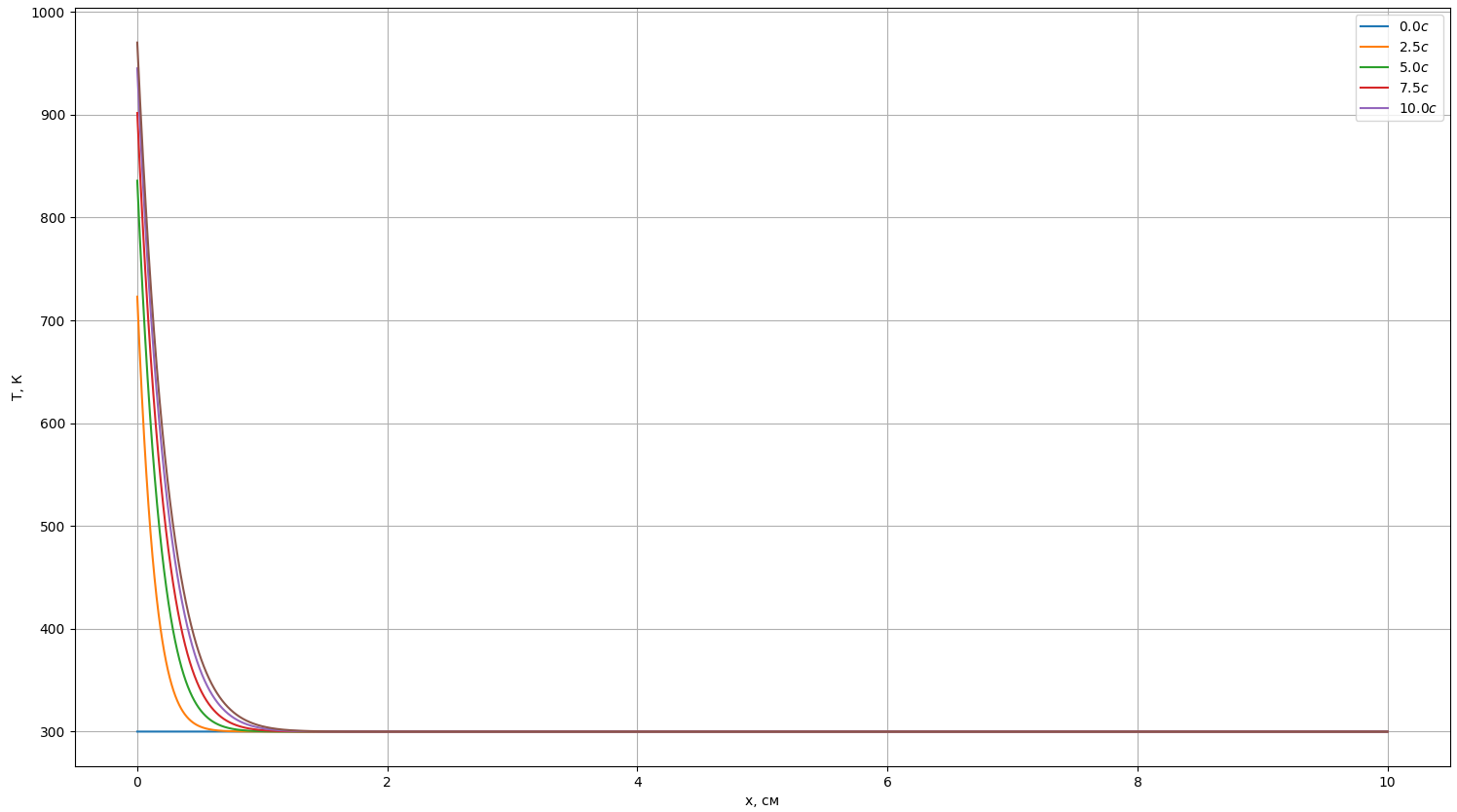
Будет принята простая аппроксимация (для функций ):

Полученное краевое условие записывается в виде

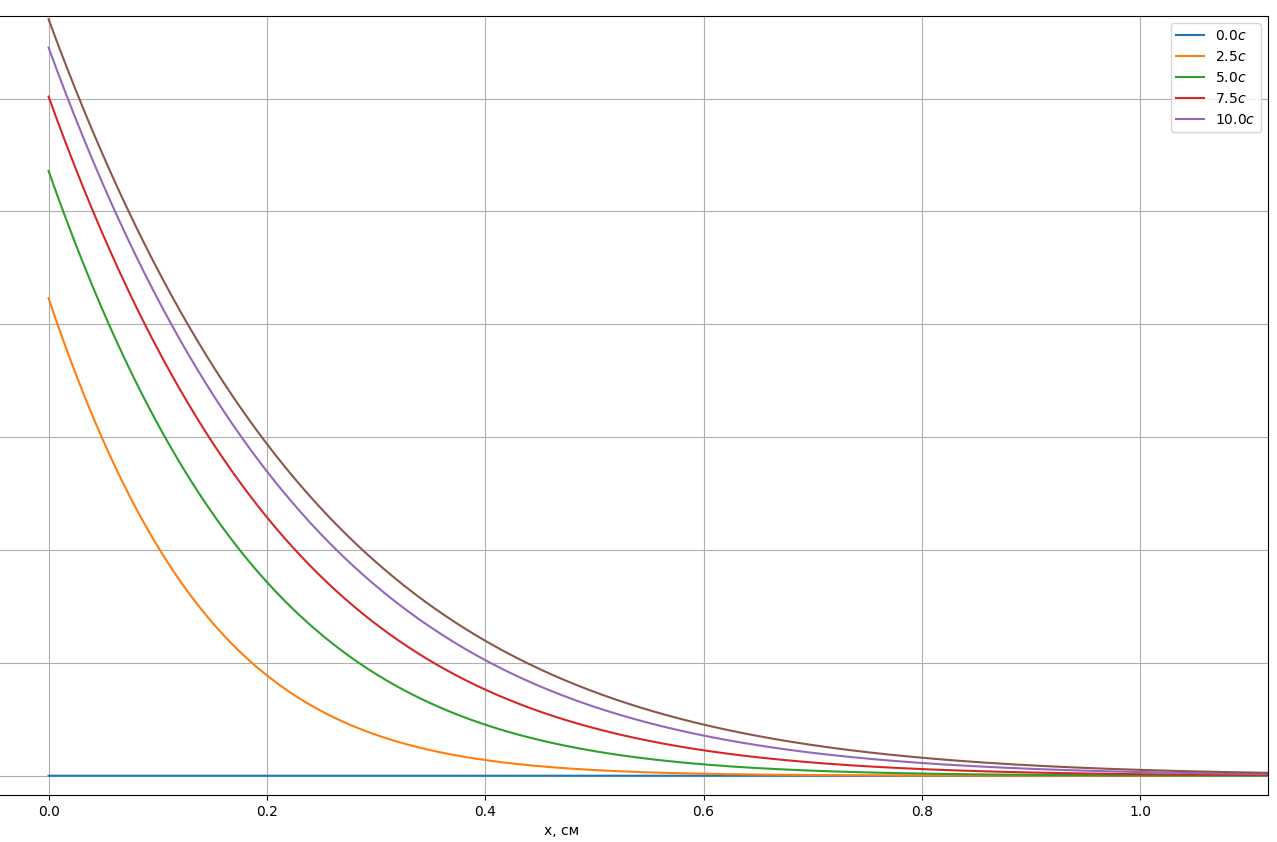
где соответствующие , , сгруппированы выше.

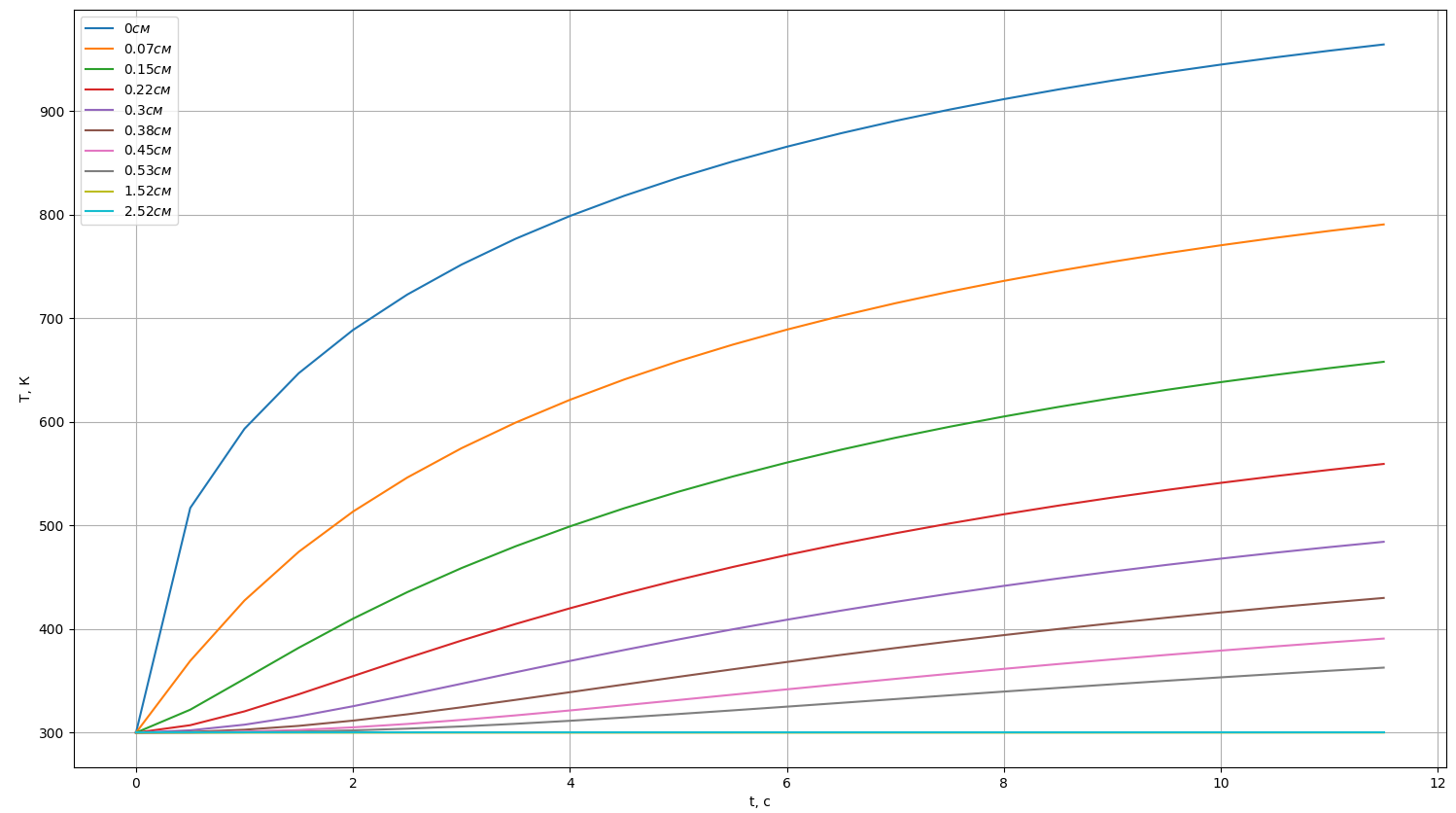
**Результаты работы программы и тестирование**

1. Графики зависимости (при фиксированных значениях времени) и (при фиксированных значениях длины) при заданных отладочных параметрах:



Приближу первый график:

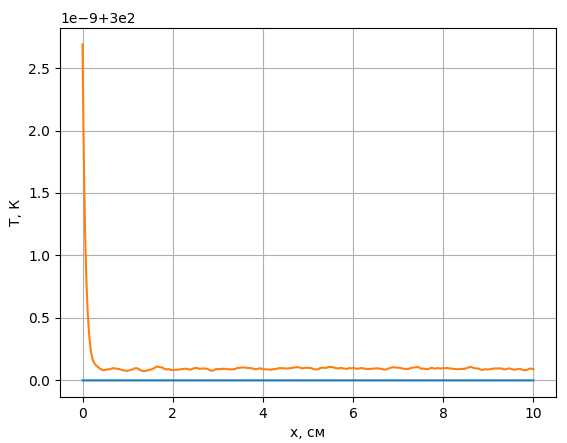




Цвет графика в легенде соответствует «срезу», на котором моделируется температура. Последующие слои не представляют интереса, поскольку их температура близка к температуре окружающей среды.

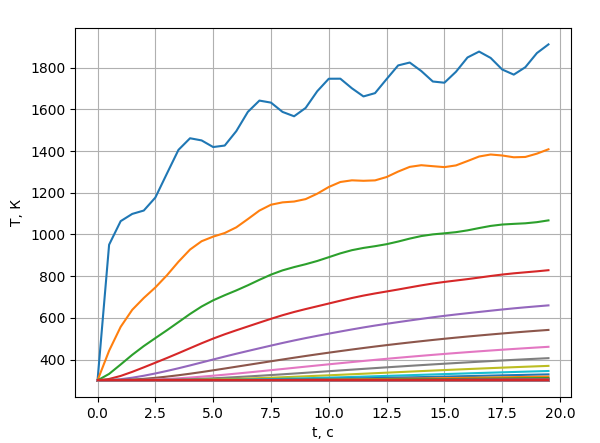
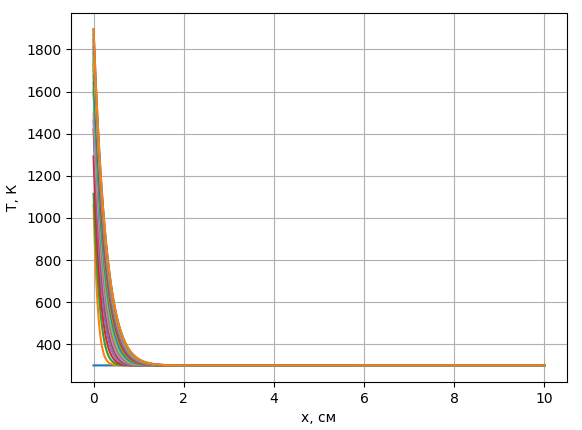
Протестируем другие возможные режимы. Также, поскольку теперь понятно, чего ожидать от вида графиков, уберу вывод легенды, чтобы можно было делать картинки размером поменьше.

Нулевой поток тепла ():



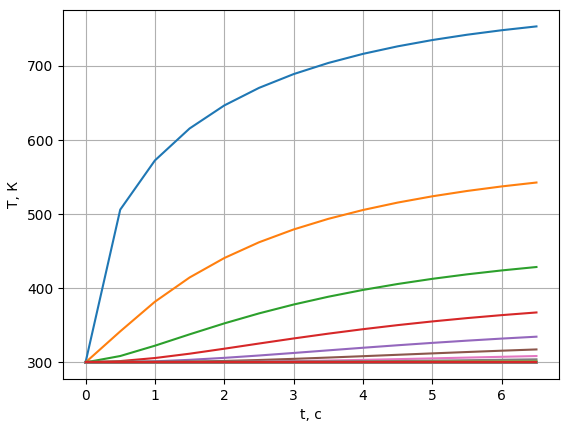
Как и ожидалось, температура стержня равна температуре окружающей среды.

Поток тепла зависит от времени (пусть ):

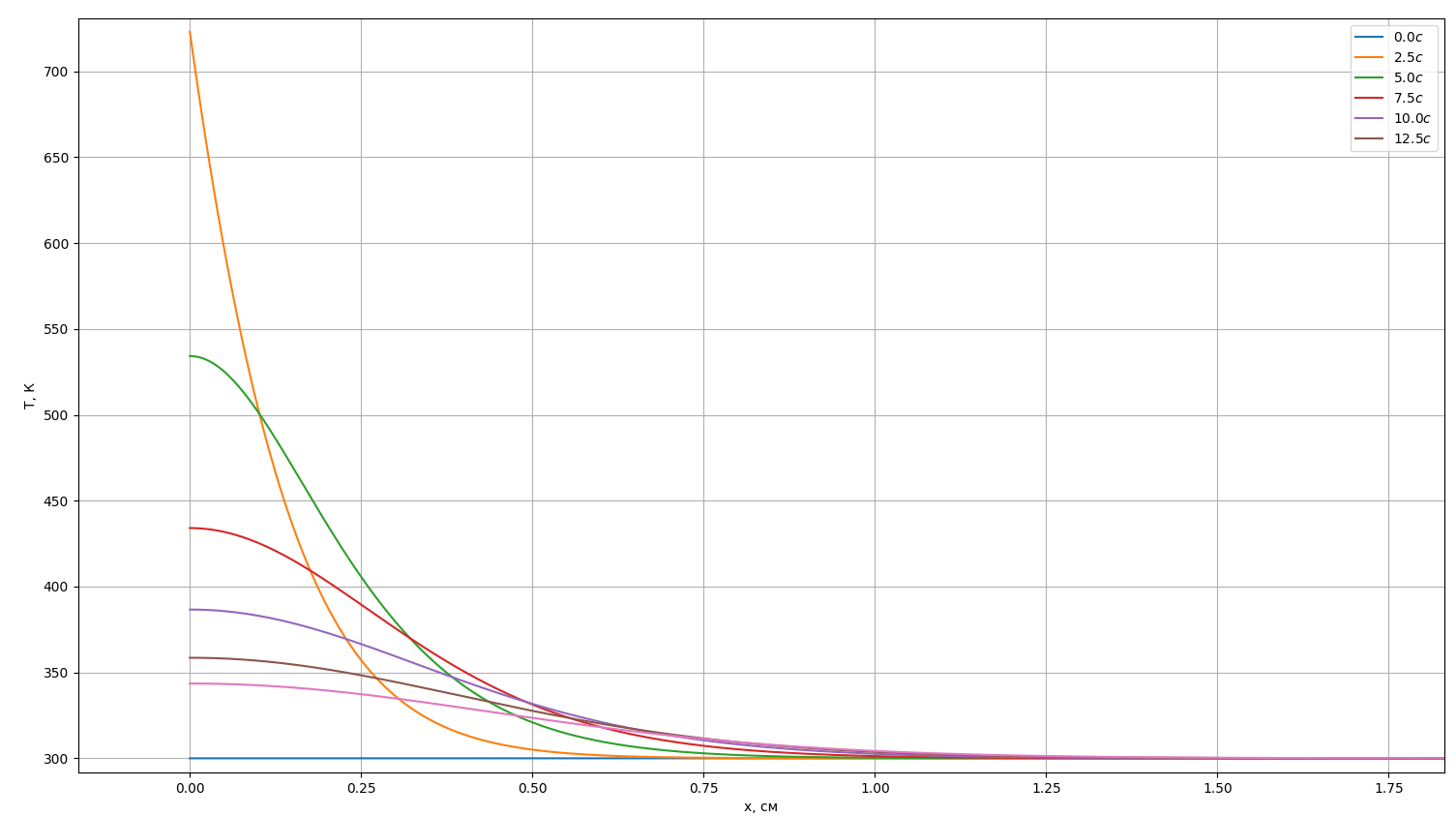


Видно, что температура стержня реагирует на «провалы» в потоке тепла, но продолжает расти в сторону финального устойчивого значения, потому что поток всегда положителен. Чем дальше расстояние от начала, тем меньше колебания потока сказываются на температурном поле.

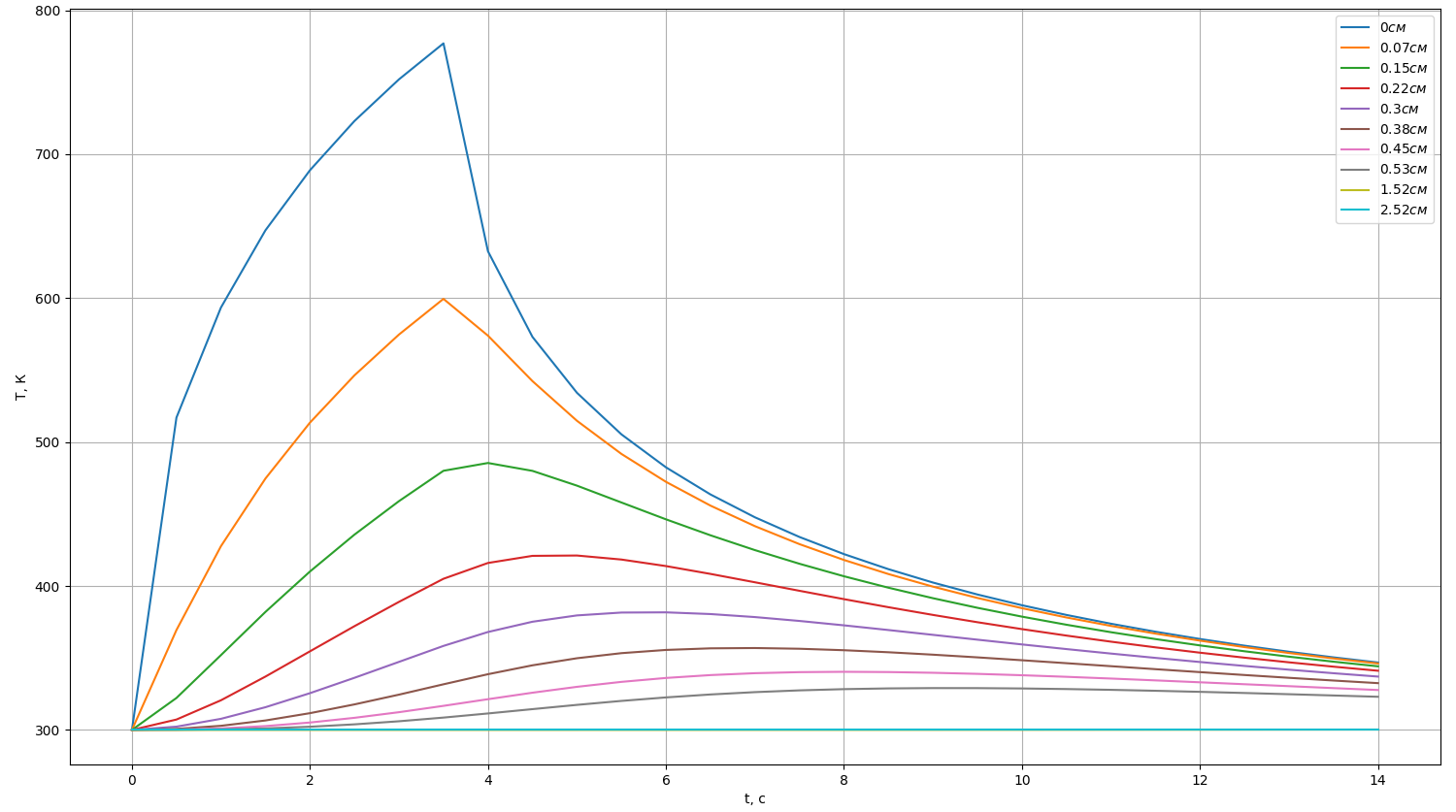
Также можно варьировать значения параметров альфа, например, увеличить в 3 раза. Это должно приводить к снижению температуры в целом, что можно наблюдать на графике (остальные параметры стандартные):

**

Реализуем сценарий, при котором по достижении некоторого времени нагрев стержня прекращается. Он должен остыть до температуры окружающей среды. Поставим обнуление потока тепла на начало 4-й секунды:

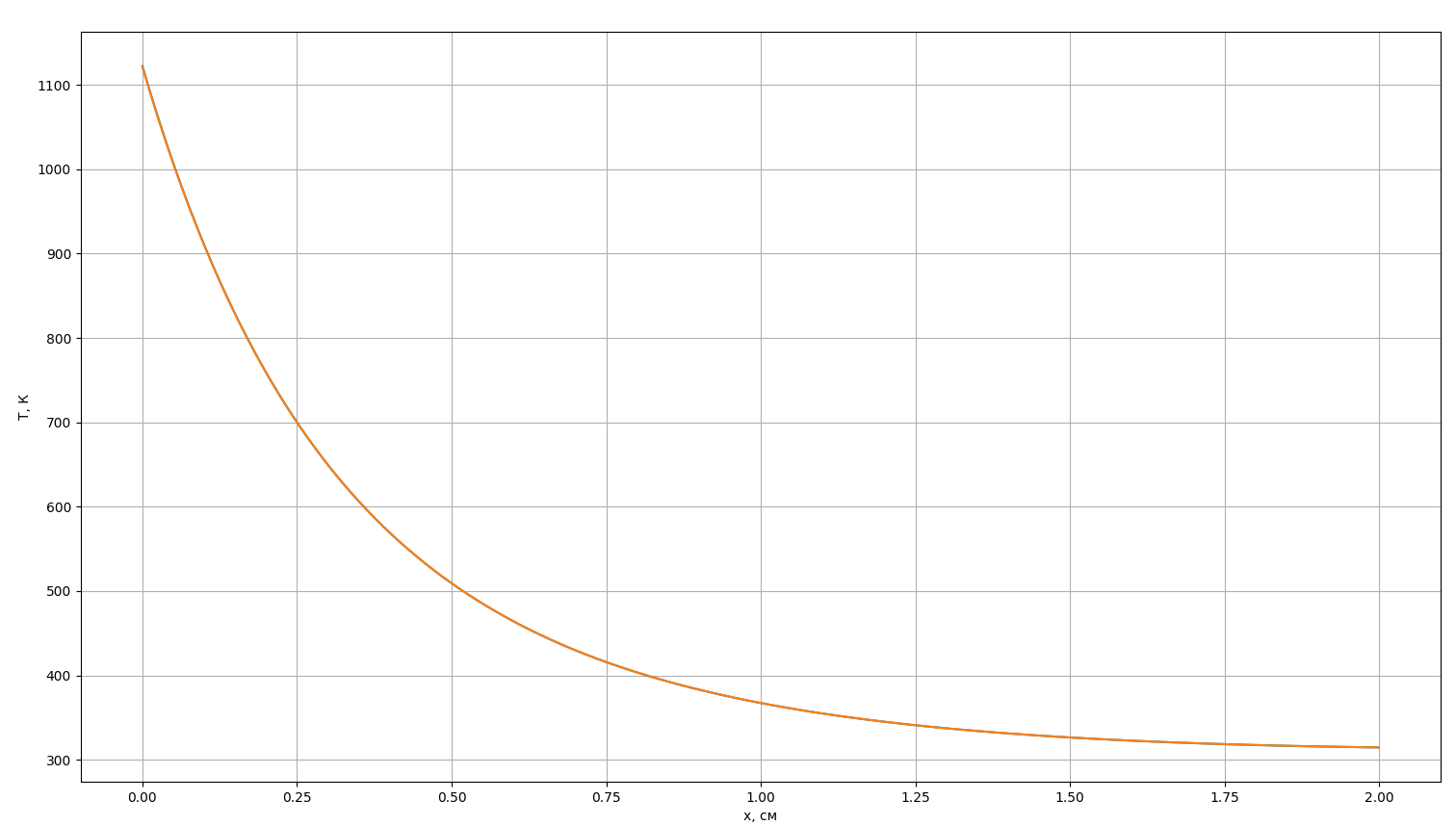


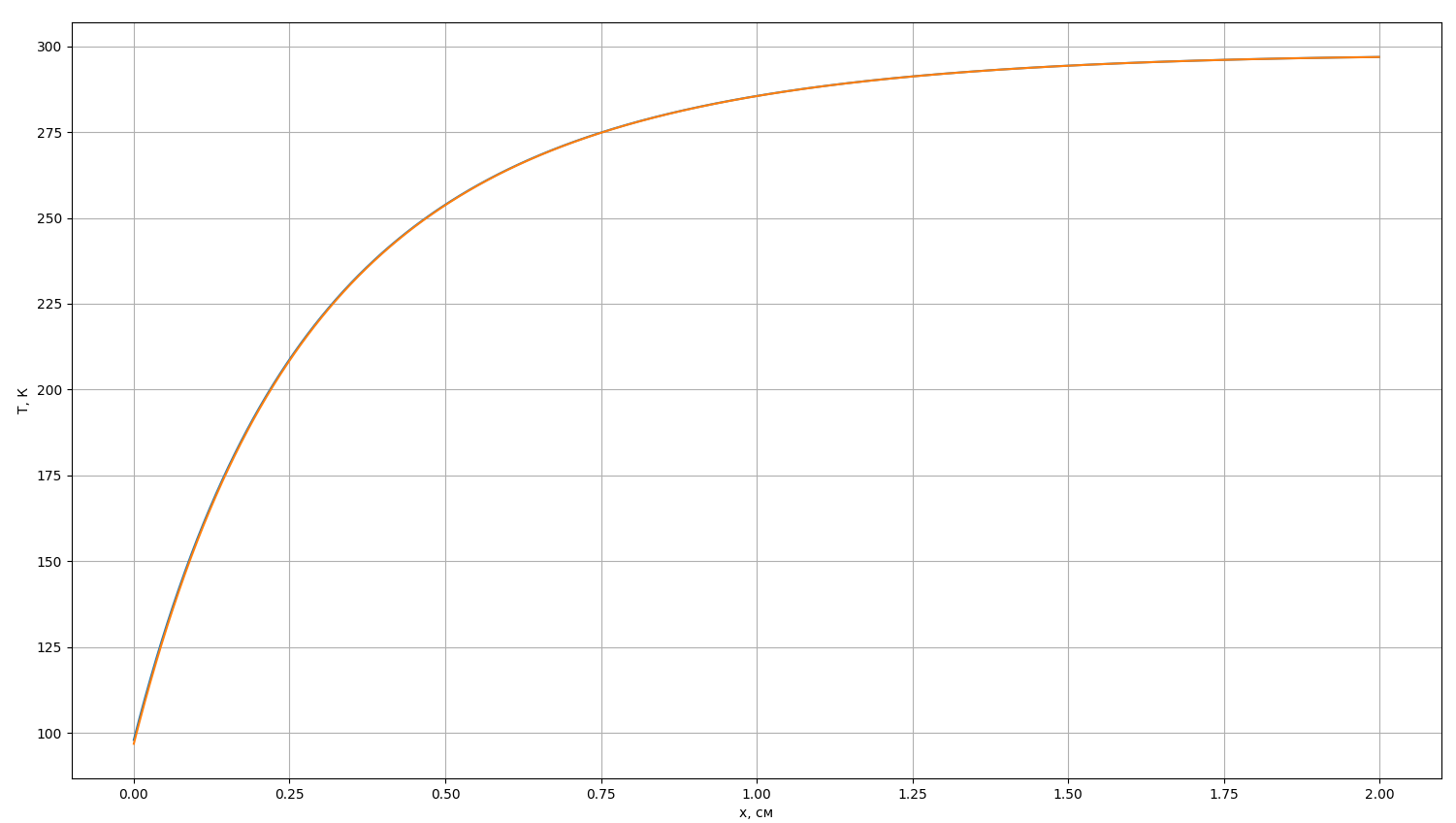
Видно, что уже к 6-й секунде стержень успел остыть. И что мне отдельно нравится – внутренние «слои» остывают плавно, без резкого перелома, как это и происходило бы в настоящем стержне.

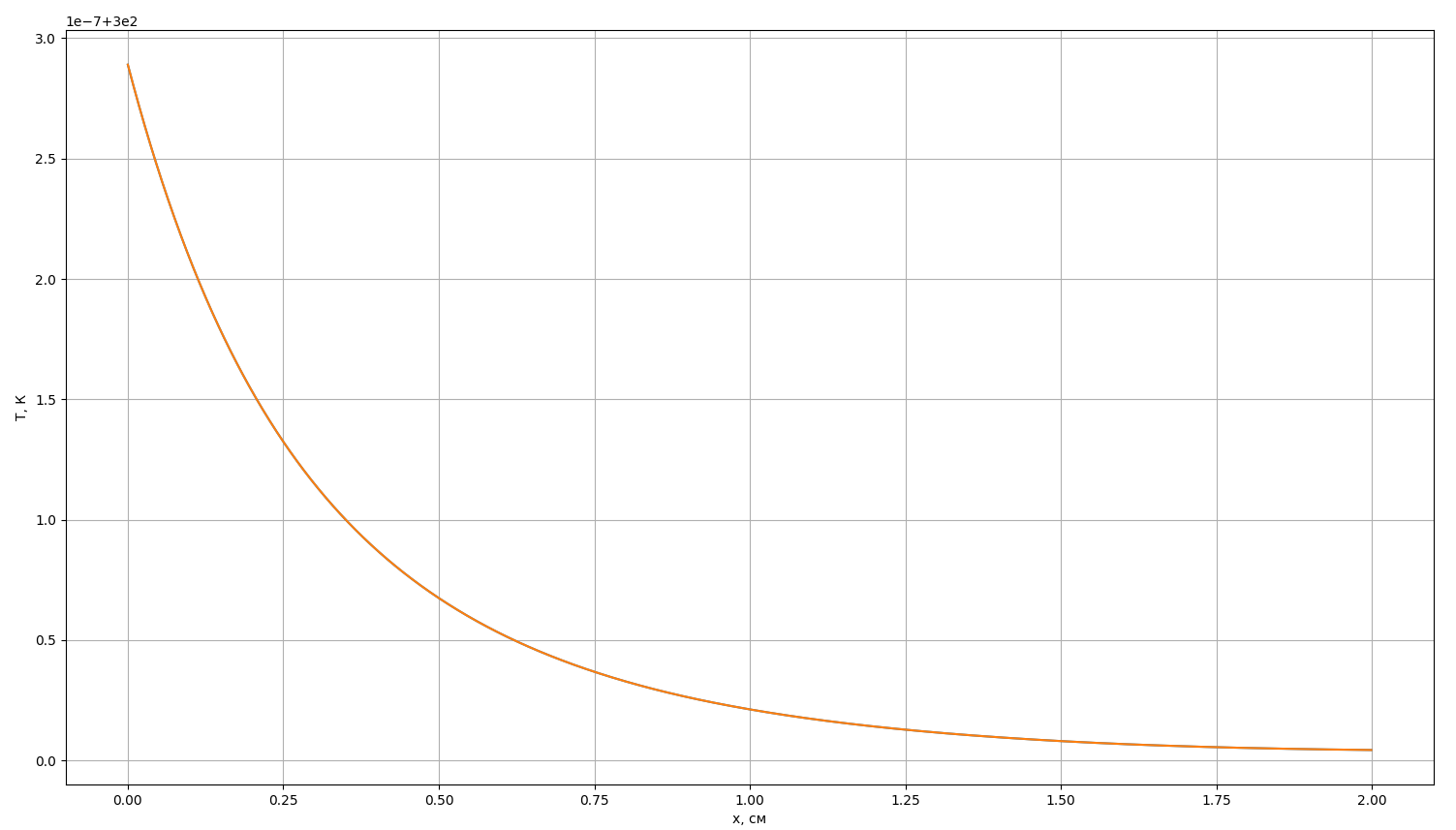


Охлаждение происходит так, как и ожидалось.

Если упростить эту программу (обнулить функцию , уменьшить длину стержня), можно получить модель, в некоторых пределах соответствующую модели из 3-й лабораторной работы. Получаемые графики демонстрируют это:





****

**Вопросы при защите лабораторной работы**

1. Тестирование

Представлено в результатах программы.