|  |  |
| --- | --- |
| 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**\_\_**

|  |  |
| --- | --- |
| **Тема \_Программно- алгоритмическая реализация моделей на основе дифференциальных уравнений в частных производных с краевыми условиями II и III рода.**  **Студент \_**Уласик Е. А.**\_\_\_\_\_\_\_\_\_\_\_**  **Группа \_**ИУ7-61Б**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_**Градов В. М.**\_\_\_\_** |  |

Москва.

2020 г.

**Введение**

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

**Задача**

Задана математическая модель.

Уравнение для функции *T(x, t):*

|  |  |
| --- | --- |
|  | *(1)* |

Краевые условия:

Функция задана своими константами:

*,*

Разностная схема с разностным краевым условием при *x = 0* (14.7) была получена в лекции №14, и может быть использована в данной работе.

Получим интегро-интерполяционным методом разностный аналог при *x = l.*

С учётом (14.2):

|  |  |
| --- | --- |
|  | (14.2) |

а также, что поток:

, a ,

проинтегрируем на отрезке [ уравнение (1):

Для интегралов справа применим метод правых прямоугольников:

Интеграл по t вычислим методом правых прямоугольников, а интегралы по x – методом трапеций:

Подставим выражения для потока:

Приведём подобные, получим:

В формуле можно принять простейшую аппроксимацию:

**Значения параметров для отладки**

* , Вт/см К
* ,

**Физическое содержание задачи**

1. Сформированная 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, то есть теплоёмкость и коэффициент теплопроводности *c(T), k(T)* зависят от *T.*

3. При *x = 0* цилиндр нагружается тепловым потоком *F(t),* в общем случае зависящим от времени.

**Результаты работы**

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

Вывод разностного аналога краевого условия при *x = l* и представлен на страницах 2-3.

1. График зависимости температуры  от координаты  при нескольких фиксированных значениях времени  при заданных выше параметрах.

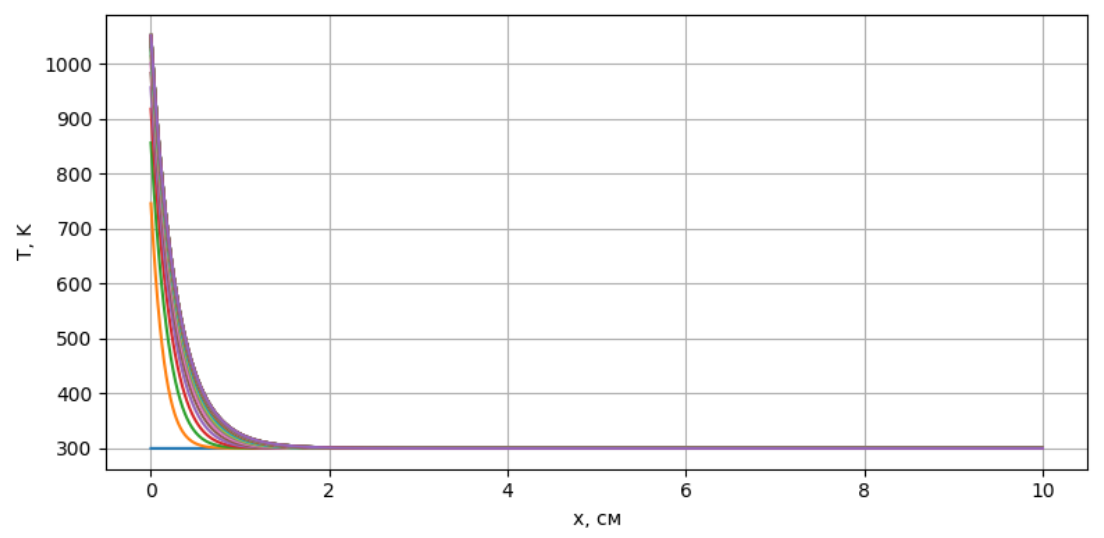


Рис 1. График зависимости температуры при фиксированных значениях времени по всей длине стержня



Рис 2. График зависимости температуры при фиксированных значениях времени до 1.65 см длины стержня

Как видно из рисунков 1-2, нижний график представляет собой график температуры в нулевой момент времени, когда температура стержня равна Верхний график является графиком температуры, соответствующий установившемуся режиму.

1. График зависимости  при нескольких фиксированных значениях координаты . Обязательно представить случай *n*=0, т.е..



Рис 3. График зависимости при фиксированном

На рис. 3 верхний график соответствует случаю , а нижний график соответствует случаю .

Контрольные вопросы

1. Приведите результаты тестирования программы (графики, общие соображения, качественный анализ). Учесть опыт выполнения лабораторной работы №3.

Для тестирования программы можно использовать график, получившийся при выходе на стационарный режим с графиком, полученным в предыдущей лабораторной работе. Для получения такого результата обнулим теплоёмкость, а также уберём зависимость коэффициента теплопроводности от T.

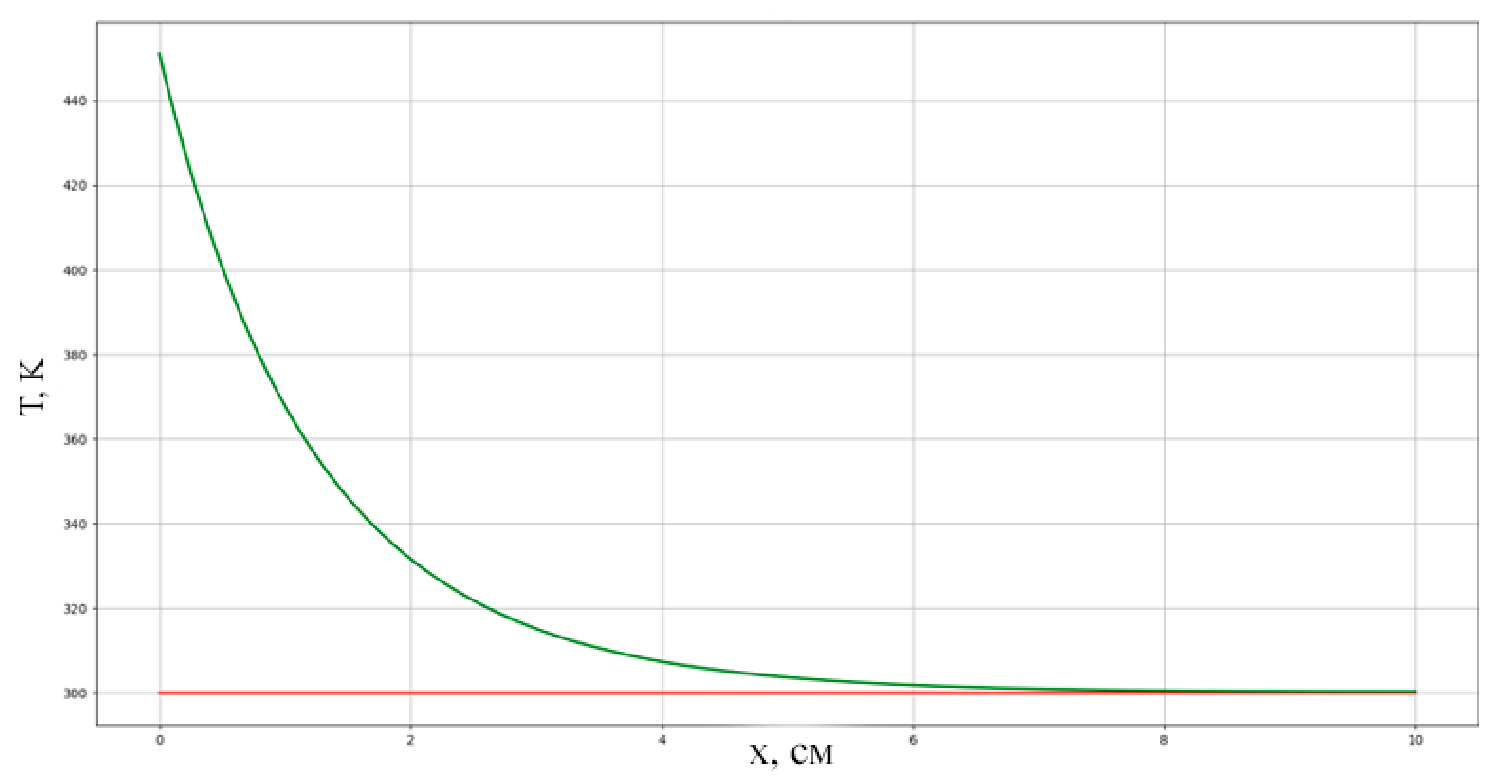


Рис. 4 График температуры при параметрах предыдущей ЛР

Также в качестве тестов можно принять , тогда нагрев будет отсутствовать и стержень будет иметь температуру окружающей среды.

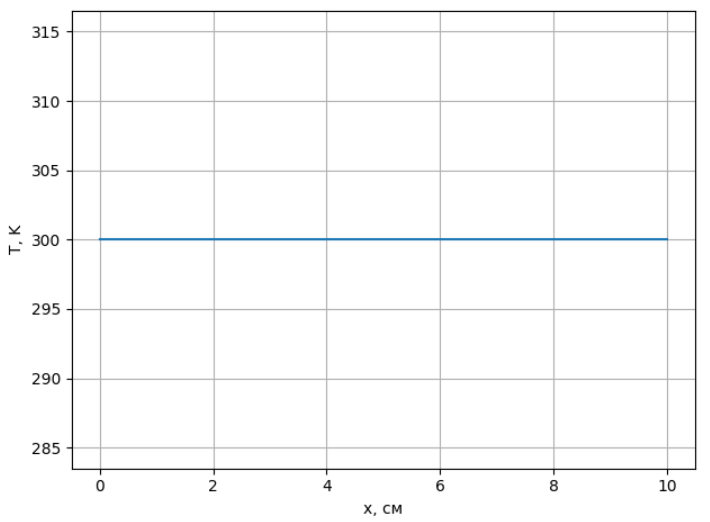
**

Рис. 5 График температуры при отсутствии нагрева

Можно задать отрицательным. Например, . Тогда будет происходит съём тепла.

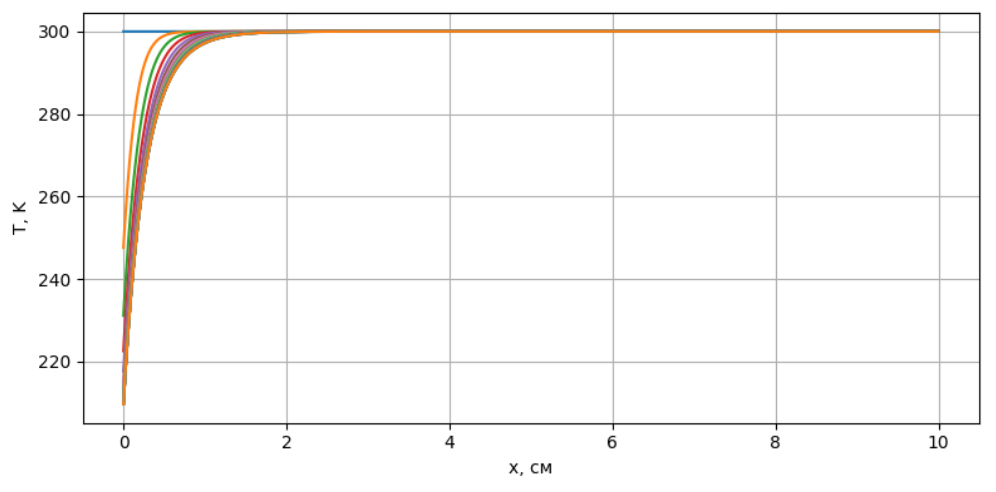
**

Рис. 6 График температуры при съёме тепла

Если после некоторого разогрева стержня принять , то нагретый стержень начнёт остывать, пока температура не станет равно температуре .

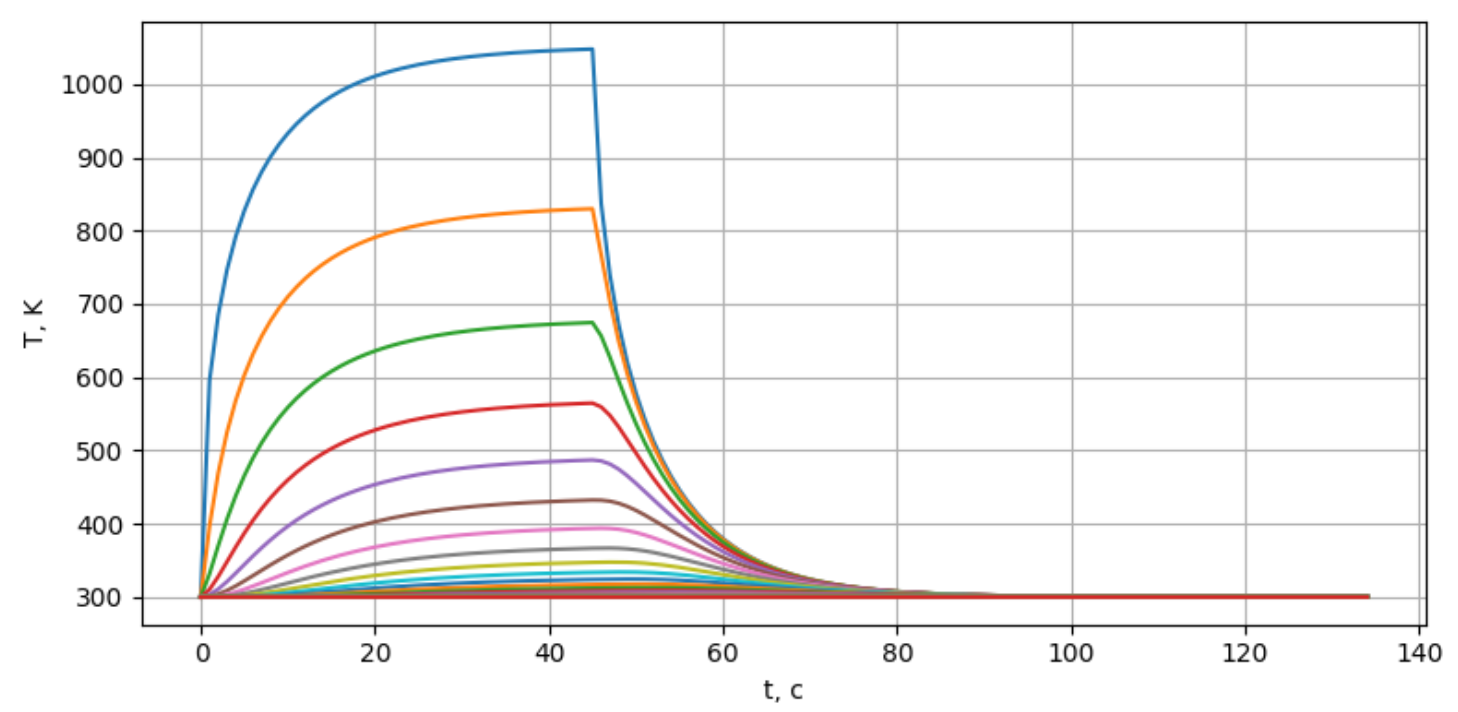


Рисунок 7. График температуры при обнулении после разогрева

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

Выполним линеаризацию уравнения (14.8)

|  |  |
| --- | --- |
|  | (14.8) |

методом Ньютона по переменным , и :

Приведём к каноническому виду:

|  |  |
| --- | --- |
|  | (2) |

где

Уравнение (2) решается методом прогонки при заданных краевых условиях.

Зная приближение s – 1, вычислим приближение s:

Итерационный процесс завершится, когда выполнится условие:

**Листинг программы**

1. **def** k(T):
2. **return** a1 \* (b1 + c1 \* T \*\* m1)
4. **def** c(T):
5. **return** a2 + b2 \* T \*\* m2 - (c2 / T \*\* 2)
7. **def** alpha(x):
8. d = (alphaN \* l) / (alphaN - alpha0)
9. c = - alpha0 \* d
10. **return** c / (x - d)
12. **def** p(x):
13. **return** (2 / R) \* alpha(x)
15. **def** f(x):
16. **return** (2 \* T0 / R) \* alpha(x)
18. **def** A(T):
19. **return** t / h \* approc\_minus\_half(k, T, t)
21. **def** D(T):
22. **return** t / h \* approc\_plus\_half(k, T, t)
24. **def** B(x, T):
25. **return** A(T) + D(T) + h \* c(T) + h \* t \* p(x)
27. **def** F(x, T):
28. **return** h \* t \* f(x) + T \* h \* c(T)
30. **def** approc\_plus\_half(func, n, step):
31. **return** (func(n) + func(n + step)) / 2

34. **def** approc\_minus\_half(func, n, step):
35. **return** (func(n) + func(n - step)) / 2
37. **def** left\_boundary\_condition(T\_prev):
38. T\_prev\_0 = T\_prev[0]
39. c\_plus = approc\_plus\_half(c, T\_prev\_0, t)
40. k\_plus = approc\_plus\_half(k, T\_prev\_0, t)
41. c0 = c(T\_prev\_0)
43. K0 = h / 8 \* c\_plus + h / 4 \* c0 + t / h \* k\_plus + \
44. t \* h / 8 \* p(h / 2) + t \* h / 4 \* p(0)
46. M0 = h / 8 \* c\_plus - t / h \* k\_plus + t \* h / 8 \* p(h / 2)
48. P0 = h / 8 \* c\_plus \* (T\_prev\_0 + T\_prev[1]) + \
49. h / 4 \* c0 \* T\_prev\_0 + F0 \* t + t \* h / 8 \* (3 \* f(0) + f(h))
51. **return** K0, M0, P0
53. **def** right\_boundary\_condition(T\_prev):
54. T\_prev\_N = T\_prev[-1]
55. c\_minus = approc\_minus\_half(c, T\_prev\_N, t)
56. k\_minus = approc\_minus\_half(k, T\_prev\_N, t)
57. cN = c(T\_prev\_N)
59. KN = h / 8 \* c\_minus + h / 4 \* cN + t / h \* k\_minus + t \* alphaN + \
60. t \* h / 8 \* p(l - h / 2) + t \* h / 4 \* p(l)
62. MN = h / 8 \* c\_minus - t / h \* k\_minus + t \* h / 8 \* p(l - h / 2)
64. PN = h / 8 \* c\_minus \* (T\_prev\_N + T\_prev[-2]) + \
65. h / 4 \* cN \* T\_prev\_N + t \* alphaN \* T0 + t \* h / 4 \* (f(l) + f(l - h / 2))
66. **return** KN, MN, PN
67. **def** calculate(T\_prev):
68. K0, M0, P0 = left\_boundary\_condition(T\_prev)
69. KN, MN, PN = right\_boundary\_condition(T\_prev)
71. eps = [0, -M0 / K0]
72. eta = [0, P0 / K0]
74. x = h
75. n = 1
76. **while** (x + h < l):
77. T\_prev\_n = T\_prev[n]
78. denominator = (B(x, T\_prev\_n) - A(T\_prev\_n) \* eps[n])
80. next\_eps = D(T\_prev\_n) / denominator
81. next\_eta = (F(x, T\_prev\_n) + A(T\_prev\_n) \* eta[n]) / denominator
83. eps.append(next\_eps)
84. eta.append(next\_eta)
86. n += 1
87. x += h
89. T\_new = [0] \* (n + 1)
90. T\_new[n] = (PN - MN \* eta[n]) / (KN + MN \* eps[n])
92. **for** i **in** range(n - 1, -1, -1):
93. T\_new[i] = eps[i + 1] \* T\_new[i + 1] + eta[i + 1]
95. **return** T\_new
97. **def** calculate\_result():
98. step1 = int(l / h)
99. T = [T0] \* (step1 + 1)
100. ti = 0
101. res = []
102. res.append(T)
103. lent = len(T)
104. kj = 0
105. **while** True:
106. T\_prev = T
107. **while** True:
108. kj += 1
109. T\_new = calculate(T\_prev)
111. cur\_max = fabs((T[0] - T\_new[0]) / T\_new[0])
112. **for** i **in** range(lent):
113. d = fabs(T[i] - T\_new[i]) / T\_new[i]
114. **if** d > cur\_max:
115. cur\_max = d
117. **if** cur\_max < 1:
118. **break**
119. T\_prev = T\_new
121. res.append(T\_new)
122. ti += t
124. flag\_eps\_ok = True
125. **for** i **in** range(lent):
126. **if** fabs((T[i] - T\_new[i]) / T\_new[i]) > 1e-5:
127. flag\_eps\_ok = False
128. **if** flag\_eps\_ok:
129. **break**
130. T = T\_new
131. **return** res, ti