|  |  |
| --- | --- |
| **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** Сушина А.Д.  **Группа** ИУ7-61б  **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. Задана математическая модель.

Уравнение для функции

(1)

Краевые условия

(2)

В обозначениях уравнения (14.1) лекции №14

(3)

2. Разностная схема с разностным краевым условием при получена в Лекции №14 (14.6),(14.7) и может быть использована в данной работе.

Разностная схема:

(4)

(5)

(6)

(7)

(8)

Разностные аналоги краевых условий при х=0:

(9)

Самостоятельно надо получить интегро-интерполяционным методом разностный аналог краевого условия при . Для этого надо проинтегрировать на отрезке [xN-1/2, xN] выписанное выше уравнение (1) и учесть, что поток

(10)

(11)

3. Значения параметров для отладки (все размерности согласованы)

, Вт/см К,

, Дж/см3К.

=0.0134, =1, =4.35 10-4, =1,

=2.049, =0.563 10-3, =0.528 105, =1.

,

0.05 Вт/см2 К,

0.01 Вт/см2 К,

10 см,

300К,

0.5 см,

50 Вт/см2 (для отладки принять постоянным).

**Физическое содержание задачи**

Постановки задач в данной лабораторной работе и работе №3 во многом совпадают. Отличия заключаются в следующем:

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

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

3. При цилиндр нагружается тепловым потоком , в общем случае зависящим от времени, а в работе №3 поток был постоянный.

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

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

При произвольной зависимости потока 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.

# Код программы

Код программы представлен на листингах 1-2.

|  |
| --- |
| Листинг 1. Functions.py  import numpy as np  from data import data  from math import fabs  a1 = data[**'a1'**]  b1 = data[**'b1'**]  c1 = data[**'c1'**]  m1 = data[**'m1'**]  a2 = data[**'a2'**]  b2 = data[**'b2'**]  c2 = data[**'c2'**]  m2 = data[**'m2'**]  alpha0 = data[**'Alpha0'**]  alphaN = data[**'AlphaN'**]  l = data[**'l'**]  T0 = data[**'t0'**]  R = data[**'R'**]  F0 = data[**'F0'**]  h = data[**'h'**]  t = data[**'t'**]  def alpha(x):  d = (alphaN \* l) / (alphaN - alpha0)  c = - alpha0 \* d  return c / (x - d)    def k(T):  return a1 \* (b1 + c1 \* T \*\* m1)  def c(T):  return a2 + b2 \* T \*\* m2 - (c2 / T \*\* 2)  def p(x):  return 2 \* alpha(x) / R  def f(x):  return 2 \* alpha(x) \* T0 / R  def func\_plus\_half(x, step, func):  return (func(x) + func(x + step)) / 2  def func\_minus\_half(x, step, func):  return (func(x) + func(x - step)) / 2  def A(T):  return t / h \* func\_minus\_half(T, t, k)  def D(T):  return t / h \* func\_plus\_half(T, t, k)  def B(x, T):  return A(T) + D(T) + c(T) \* h + p(x) \* h \* t  def F(x, T):  return f(x) \* h \* t + c(T) \* T \* h  def left\_condition(y):  c0 = c(y[0])  c12 = func\_plus\_half(y[0], t, c)  k12 = func\_plus\_half(y[0], t, k)  p0 = p(0)  p12 = p(h / 2)  K0 = h / 8 \* c12 + h / 4 \* c0 + \  k12 \* t / h + \  t \* h / 8 \* p12 + t \* h / 4 \* p0  M0 = h / 8 \* c12 - \  k12 \* t / h + \  t \* h \* p12 / 8  P0 = h / 8 \* c12 \* (y[0] + y[1]) + \  h / 4 \* c0 \* y[0] + \  F0 \* t + t \* h / 8 \* (3 \* f(0) + f(h))  return K0, M0, P0  def right\_condition(y):  cn12 = func\_minus\_half(y[-1], t, c)  cn = c(y[-1])  kn12 = func\_minus\_half(y[-1], t, k)  pn12 = p(l - h / 2)  pn = p(l)  fn = f(l)  fn12 = f(l - h / 2)  KN = h / 8 \* cn12 + h / 4 \* cn + \  kn12 \* t / h + t \* alphaN + \  t \* h / 8 \* pn12 + t \* h / 4 \* pn  MN = h / 8 \* cn12 - \  kn12 \* t / h + \  t \* h \* pn12 / 8  PN = h / 8 \* cn12 \* (y[-1] + y[-2]) + \  h / 4 \* cn \* y[-1] + t \* alphaN \* T0 + \  t \* h / 4 \* (fn + fn12)  return KN, MN, PN  def calculate(prev):  K0,M0,P0 = left\_condition(prev)  KN,MN,PN = right\_condition(prev)  eps = [0, -M0 / K0]  eta = [0, P0 / K0]    x = h  n = 1  while (x + h < l):  new\_eps = D(prev[n]) / (B(x, prev[n]) - A(prev[n]) \* eps[n])  new\_eta = (F(x, prev[n]) + A(prev[n]) \* eta[n]) / (B(x, prev[n]) - A(prev[n]) \* eps[n])  eps.append(new\_eps)  eta.append(new\_eta)  n += 1  x += h  y = [0] \* (n + 1)  y[n] = (PN - MN \* eta[n]) / (KN + MN \* eps[n])  for i in range(n - 1, -1, -1):  y[i] = eps[i + 1] \* y[i + 1] + eta[i + 1]  return y  def get\_result():  step1 = int(l / h)  T = [T0] \* (step1 + 1)  T\_new = [0] \* (step1 + 1)  ti = 0  res = []  res.append(T)  lent = len(T)  while True:  prev = T  while True:  T\_new = calculate(prev)  max = fabs((T[0] - T\_new[0]) / T\_new[0])  for step2, j in zip(T, T\_new):  d = fabs(step2 - j) / j  if d > max:  max = d  if max < 1:  break    prev = T\_new  res.append(T\_new)  ti += t  check\_eps = 0  for i, j in zip(T, T\_new):  if fabs((i - j) / j) > 1e-2:  check\_eps = 1  if check\_eps == 0:  break  T = T\_new  x = [i for i in np.arange(0, l, h)]  te = [i for i in range(0, ti, t)]  return res, x, te |
| Листинг 2. main.py  import matplotlib.pyplot as plt  import data as const  from functions import get\_result  from tkinter import \*  root = Tk()  varList = {  **'a1'** : StringVar(),  **'b1'** : StringVar(),  **'c1'** : StringVar(),  **'m1'**: StringVar(),  **'a2'**: StringVar(),  **'b2'**: StringVar(),  **'c2'**: StringVar(),  **'m2'**: StringVar(),  **"Alpha0"**: StringVar(),  **"AlphaN"**: StringVar(),  **"l"**: StringVar(),  **"t0"**: StringVar(),  **"R"**: StringVar(),  **"F0"**: StringVar(),  **"h"**: StringVar(),  **'t'**: StringVar(),  }  def create\_grid(root):  i = 0  for var in varList.keys():  label = Label(root, text=var)  label.grid(row=i, column=0, sticky=**"e"**)  entry = Entry(root,width=10,textvariable=varList[var])  entry.grid(row=i, column=1)  entry.insert(0, str(const.resetData[var]))  i+=1  def check\_is\_num():  for var in varList.values():  try:  float(var.get())  except ValueError:  return False  return True  def start\_work(Event):  if not check\_is\_num():  print(**"WARNING NOT DIGIT"**)  return  for var in varList.keys():  const.data[var] = float(varList[var].get())  res, x, te = get\_result()  plt.subplot(1, 2, 1)  step1 = 0  for i in res:  if (step1 % 2 == 0):  plt.plot(x, i[:-1])  step1 +=1  plt.title(**'T(x)'**)  plt.plot(x , res[-1][:-1])  plt.xlabel(**"x, sm"**)  plt.ylabel(**"T, K"**)  plt.grid()  plt.subplot(1, 2, 2)  h = const.data[**'h'**]  step2 = 0  while (step2 < 1/3):  point = [j[int(step2 / h)] for j in res]  plt.plot(te, point[:-1])  step2 += 0.1  plt.xlabel(**"t, sec"**)  plt.ylabel(**"T, K"**)  plt.grid()  plt.show()    if \_\_name\_\_ == **'\_\_main\_\_'**:    btn = Button(root, text=**"START"**)  create\_grid(root)  btn.bind(**"<Button-1>"**, start\_work)  btn.grid(column=1, padx=10, pady=10)  root.mainloop() |

# Результаты работы.

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

Для получения разностного аналога краевого условия при , надо проинтегрировать на отрезке [xN-1/2, xN] выписанное выше уравнение (1) и учесть (10) и (11)

Обозначим (12)

Тогда (1) можно записать в виде:

(13)

где

Проитегрируем (13):

(14)

Вычисляем интегралы

(15)

Приведем к виду . Получим:

(16)

Для функций c, X, p будет принята простая аппроксимация.

(17)

Из (9) и (16) получим .

Получим систему:

(18)

Эту систему можно решить методом итераций. Пусть i — номер итерации

2. График зависимости температуры от координаты при нескольких фиксированных значениях времени (аналогично рисунку в лекции №14) при заданных выше параметрах. Обязательно представить распределение в момент времени, соответствующий установившемуся режиму, когда поле перестает меняться с некоторой точностью (например, ), т.е. имеет место выход на стационарный режим. На этой стадии левая часть дифференциального уравнения близка к нулю, и на самом деле решается уравнение из лабораторной работы №3 (отличие только в том, что там было линейное уравнение).

|  |
| --- |
| Рис 1. График зависимости температуры от координаты при нескольких фиксированных значениях времени |

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

Синий график — x=0

|  |
| --- |
| Рис 2. График зависимости при нескольких фиксированных значениях координаты |

# Вопросы при защите лабораторной работы.

Ответы на вопросы дать письменно в Отчете о лабораторной работе.

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

Если принять F0=-10. При отрицательном тепловом потоке слева идет съем тепла.

Для ЛР4:

|  |
| --- |
| Рис 3. Результат работы программы при f0=-10 |

Для ЛР3:

|  |
| --- |
| Рис 4. Результат работы ЛР3 при f0=-10 |

Если принять F0 =0.Тепловое нагружение отсутствует, причин для нагрева нет, температура стержня должна быть равна температуре окружающей среды T0.

ЛР4:

|  |  |
| --- | --- |
| Рис 5. Результат работы программы | Рис 6. Погрешность |

2. 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.

Система уравнений:

(19)

Коэффициенты зависят от . Исходя из этого получим:

(20)

Канонический вид:

, (21)

где:

(22)

(23)

(24)

(25)

Подставим краевые условия и получим их в каноническом виде:

где

(26)

(27)

(28)

(29)

(30)

(31)

Получаем систему:

(32)

Для решения системы необходимо найти все . Зная приближение (i-1), можно найти приближение (i). Найдём значение искомой функции в узлах:

(33)

Условие завершения:

(34)