|  |  |
| --- | --- |
| 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** **Модели на основе ДУ в частных производных с краевыми условиями II и III рода**  **Студент Якуба Д. В.**  **Группа ИУ7-63Б**  **Оценка (баллы)**  **Преподаватель Градов В. М.** |  |

Москва

2021 г.

Лабораторная работа по теме «Модели на основе ДУ в частных производных с краевыми условиями II и III рода»

Тема:

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

Цель работы:

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

Задание:

1. Задана математическая модель.

Уравнение для функции

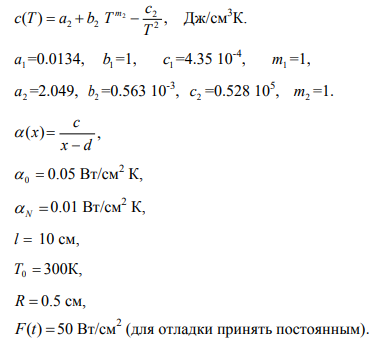
Краевые условия:

В обозначениях уравнения лекции

2. 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 при . Для этого надо проинтегрировать на отрезке выписанное выше уравнение и учесть, что поток , а

3. Значение параметров для отладки:





Результат

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

Пусть , .

В таком случае уравнение

При факте того, что

Примет вид:

Проинтегрируем данное выражение:

То есть

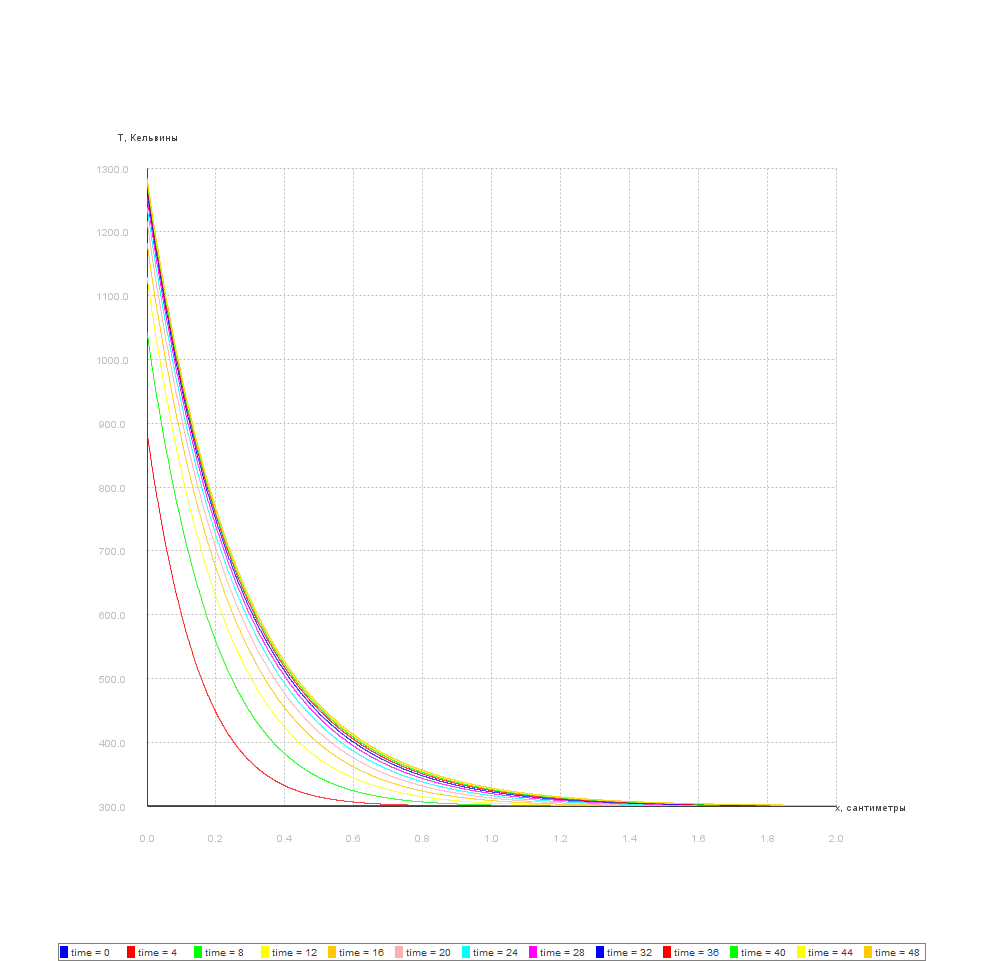
Для первого интеграла в правой части воспользуемся методом правых прямоугольников, а для остальных – методом трапеций.

Учитывая , , , получим:

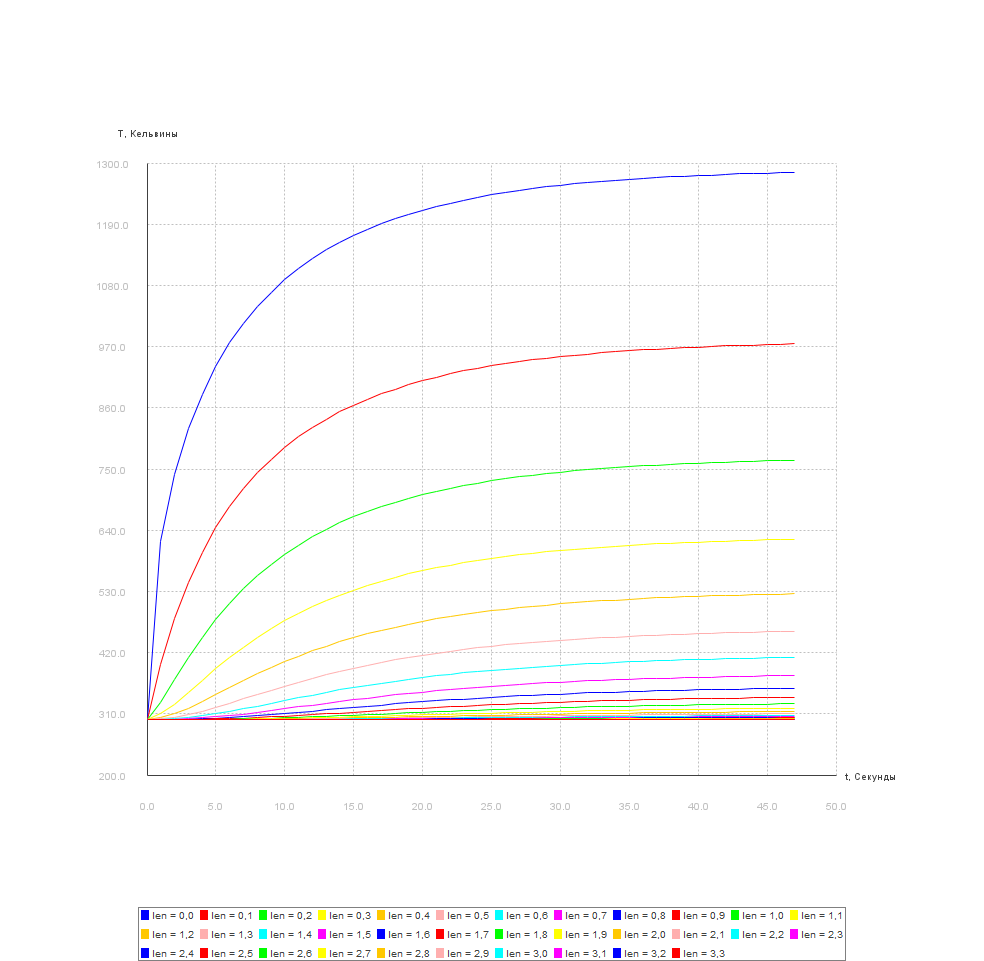
Приведём данное уравнение к разностному аналогу краевого условия при :

Требуется найти левые и правые коэффициенты:

Данная система решается методом итераций:

2. График зависимости температуры от координаты *x* при нескольких фиксированных значениях времени (аналогично рисунку в лекции) при заданных выше параметрах.

На рисунке представлены графики зависимости температуры от координаты при фиксированных *t*.

График зависимости при нескольких фиксированных значениях координаты . Обязательно представить случай , то есть .

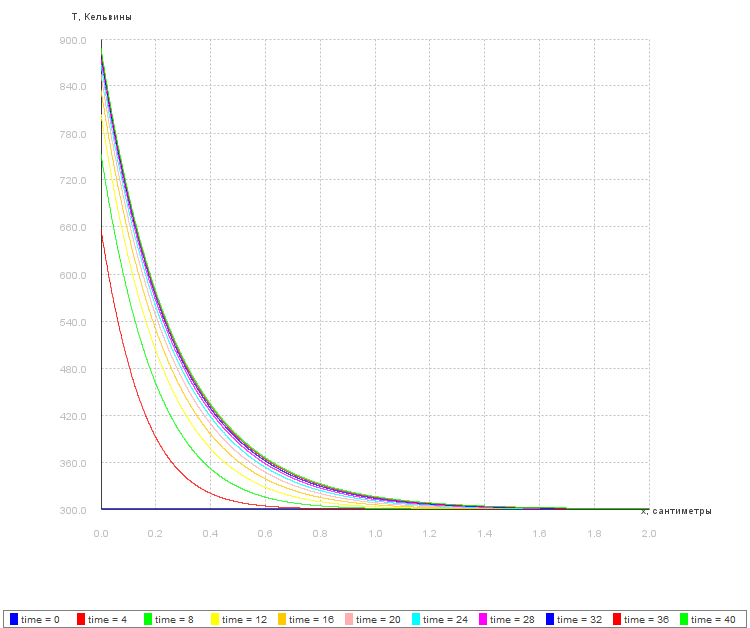
На рисунке представлены графики зависимости температуры от времени при фиксированных *x*.

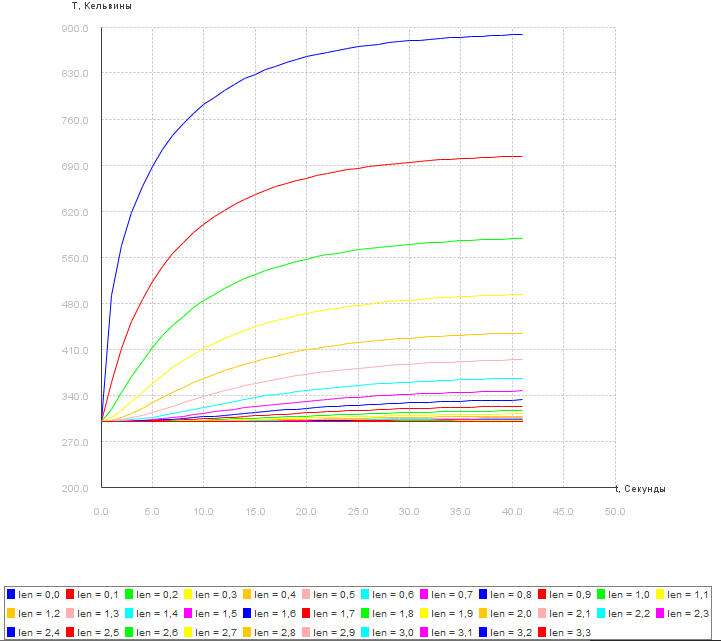
Контрольные вопросы

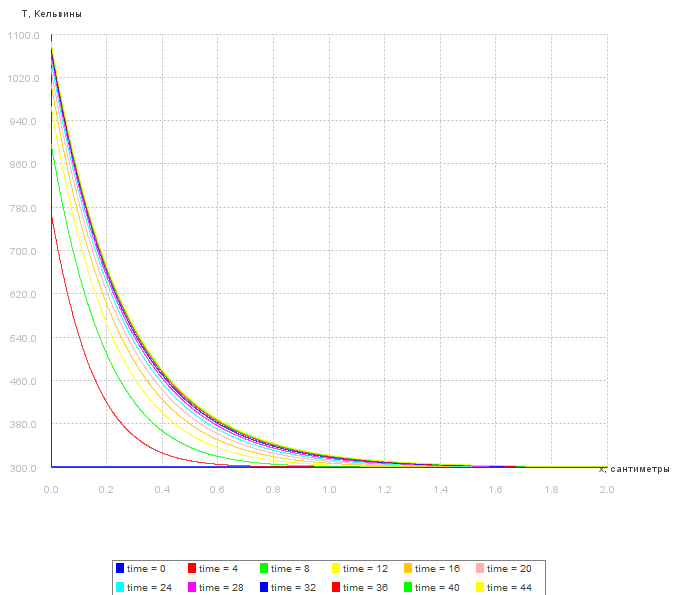
1. Приведите результаты тестирования программы (графики, общие соображения, качественный анализ)

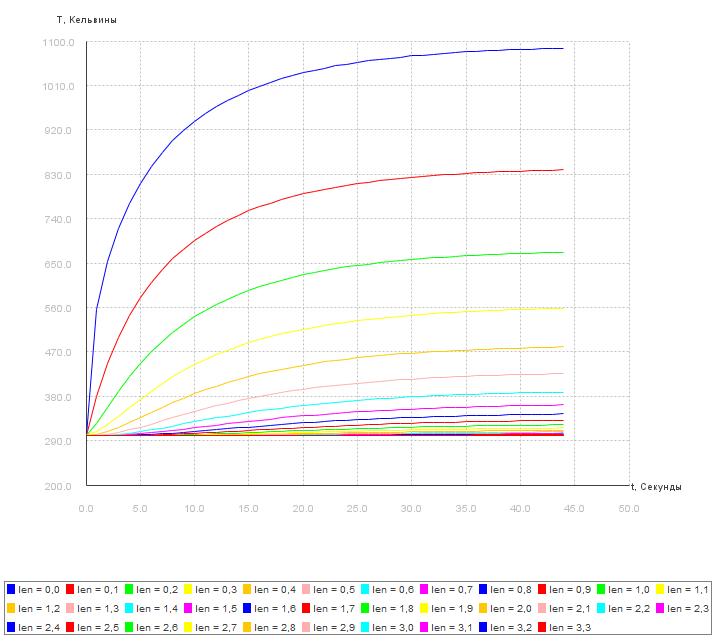
Ответ:

При постепенном увеличении теплового потока можем видеть возрастание значений температур:

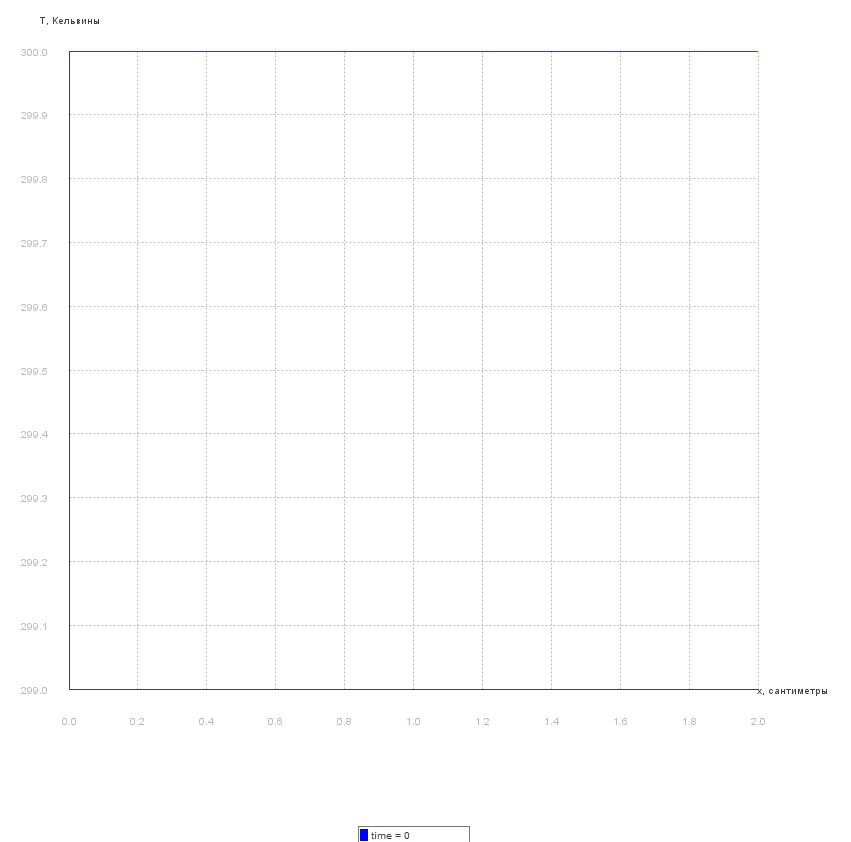






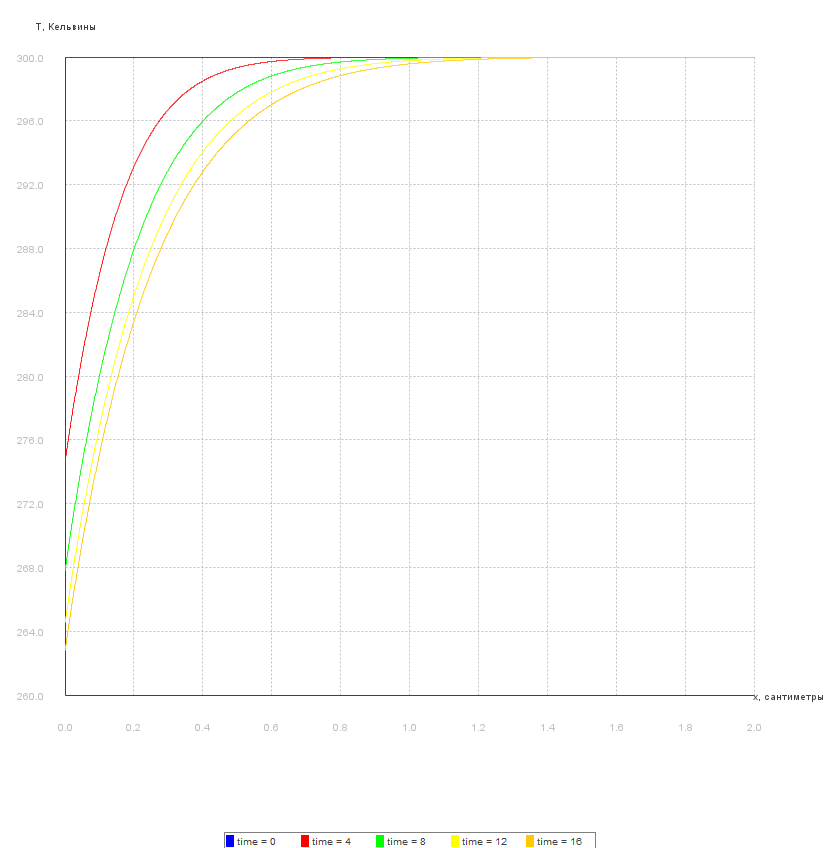


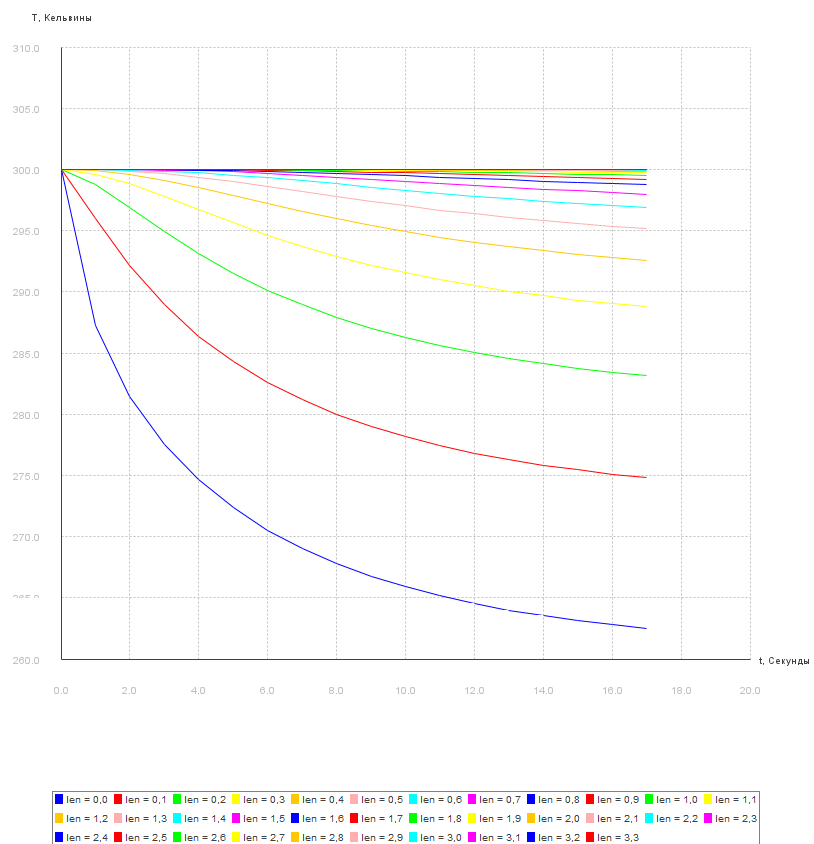
При тепловом потоке, равном нулю (, температура не должна изменяться:



При указании отрицательного значения теплового потока (, например) должно происходить охлаждение слева:

val fZero = -2.0





Код программы

|  |
| --- |
| import kotlin.math.abs import kotlin.math.pow import javax.swing.\* import org.math.plot.Plot2DPanel  class Parameters() {  val a1 = 0.0134  val b1 = 1.0  val c1 = 4.35e-4  val m1 = 1.0  val a2 = 2.049  val b2 = 0.563e-3  val c2 = 0.528e5  val m2 = 1.0  val alphaZero = 0.05  val alphaN = 1e-2  val l = 10.0  val tZero = 300.0  val r = 5e-1  val fZero = 50.0  val h = 1e-3  val t = 1.0  val epsilon = 1e-3 }  val *parameters* = Parameters()  fun plusApprox(function: (Double) -> Double, n: Double, step: Double): Double {  return (function(n) + function(n + step)) / 2 }  fun minusApprox(function: (Double) -> Double, n: Double, step: Double): Double {  return (function(n) + function(n - step)) / 2 }  val *kFun* = **{** x: Double **->** *parameters*.a1 \* (*parameters*.b1 + *parameters*.c1 \* *parameters*.m1.*pow*(x)) **}** val *cFun* = **{** x: Double **->** *parameters*.a2 + *parameters*.b2 \* x.*pow*(*parameters*.m2) - (*parameters*.c2 / x.*pow*(2)) **}** fun alphaFun(x: Double): Double {  val s1 = (*parameters*.alphaN \* *parameters*.l) / (*parameters*.alphaN - *parameters*.alphaZero)  val s2 = -*parameters*.alphaZero \* s1  return s2 / (x - s1) }  val *pFun* = **{** x: Double **->** *alphaFun*(x) \* 2 / *parameters*.r **}** val *fFun* = **{** x: Double **->** *alphaFun*(x) \* 2 \* *parameters*.tZero / *parameters*.r **}** val *aAFun* = **{** x: Double **->** *parameters*.t / *parameters*.h \* *minusApprox*(*kFun*, x, *parameters*.t) **}** val *dDFun* = **{** x: Double **->** *parameters*.t / *parameters*.h \* *plusApprox*(*kFun*, x, *parameters*.t) **}** val *bBFun* =  **{** x: Double, t: Double **->** *aAFun*(t) + *dDFun*(t) + *parameters*.h \* *cFun*(t) + *parameters*.h \* *parameters*.t \* *pFun*(x) **}** val *fFFun* = **{** x: Double, t: Double **->** *parameters*.h \* *parameters*.t \* *fFun*(x) + t \* *parameters*.h \* *cFun*(t) **}** fun leftexitConditions(tList: MutableList<Double>): Triple<Double, Double, Double> {  val c = *plusApprox*(*cFun*, tList[0], *parameters*.t)  val k = *plusApprox*(*kFun*, tList[0], *parameters*.t)   val kZero =  *parameters*.h / 8 \* c + *parameters*.h / 4 \* *cFun*(tList[0]) +  *parameters*.t / *parameters*.h \* k + *parameters*.t \* *parameters*.h / 8 \* *pFun*(  *parameters*.h / 2  ) + *parameters*.t \* *parameters*.h / 4 \* *pFun*(0.0)   val mZero =  *parameters*.h / 8 \* c - *parameters*.t / *parameters*.h \* k +  *parameters*.t \* *parameters*.h / 8 \* *pFun*(*parameters*.h / 2)   val pZero =  *parameters*.h / 8 \* c \* (tList.*first*() + tList[1]) +  *parameters*.h / 4 \* *cFun*(tList.*first*()) \* tList.*first*() +  *parameters*.fZero \* *parameters*.t + *parameters*.t \* *parameters*.h / 8 \* (3 \* *fFun*(  0.0  ) + *fFun*(*parameters*.h))   return Triple(kZero, mZero, pZero) }  fun rightexitConditions(tList: MutableList<Double>): Triple<Double, Double, Double> {  val c = *minusApprox*(*cFun*, tList.*last*(), *parameters*.t)  val k = *minusApprox*(*kFun*, tList.*last*(), *parameters*.t)   val kN =  *parameters*.h / 8 \* c + *parameters*.h / 4 \* *cFun*(tList.*last*()) +  *parameters*.t / *parameters*.h \* k +  *parameters*.t \* *parameters*.alphaN +  *parameters*.t \* *parameters*.h / 8 \* *pFun*(  *parameters*.l - *parameters*.h / 2  ) + *parameters*.t \* *parameters*.h / 4 \* *pFun*(*parameters*.l)   val mN =  *parameters*.h / 8 \* c - *parameters*.t / *parameters*.h \* k +  *parameters*.t \* *parameters*.h / 8 \* *pFun*(*parameters*.l - *parameters*.h / 2)   val pN =  *parameters*.h / 8 \* c \* (tList.*last*() + tList[tList.size - 2]) +  *parameters*.h / 4 \* *cFun*(tList.*last*()) \* tList.*last*() +  *parameters*.t \* *parameters*.alphaN \* *parameters*.tZero +  *parameters*.t \* *parameters*.h / 4 \* (*fFun*(  *parameters*.l  ) + *fFun*(*parameters*.l - *parameters*.h / 2))   return Triple(kN, mN, pN) }  fun formNewTList(list: MutableList<Double>): MutableList<Double> {  val zeroTriple = *leftexitConditions*(list)  val nTriple = *rightexitConditions*(list)   val xiList: MutableList<Double> = *mutableListOf*(0.0, -zeroTriple.second / zeroTriple.first)  val etaList: MutableList<Double> = *mutableListOf*(0.0, zeroTriple.third / zeroTriple.first)   var curX = *parameters*.h  var curN = 1   while (curX + *parameters*.h < *parameters*.l)  {  val curT = list[curN]  val dm = *bBFun*(curX, curT) - *aAFun*(curT) \* xiList[curN]   xiList.add(*dDFun*(curT) / dm)  etaList.add((*fFFun*(curX, curT) + *aAFun*(curT) \* etaList[curN]) / dm)   curX += *parameters*.h  curN++  }   val outT = *mutableListOf*<Double>()  for (i in 0..curN)  outT.add(0.0)   outT[curN] =  (nTriple.third - nTriple.second \* etaList[curN]) / (nTriple.first + nTriple.second \* xiList[curN])   for (i in curN - 1 *downTo* 0)  outT[i] = xiList[i + 1] \* outT[i + 1] + etaList[i + 1]   return outT }  fun simpleIteration(): Pair<MutableList<MutableList<Double>>, Double> {  var tList = *mutableListOf*<Double>()  var newTList = *mutableListOf*<Double>()   for (i in 0..(*parameters*.l / *parameters*.h).toInt())  {  tList.add(*parameters*.tZero)  newTList.add(0.0)  }   val outList = *mutableListOf*(tList)   var curT = 0.0  var exitCondition = true  while (exitCondition)  {  var tempList = tList  var max = 1.0   while (max >= 1)  {  newTList = *formNewTList*(tempList)  max = *abs*((tList.*first*() - newTList.*first*()) / newTList.*first*())   for (ind in tList.*indices*)  {  if (*abs*((tList[ind] - newTList[ind]) / newTList[ind]) > max)  max = *abs*((tList[ind] - newTList[ind]) / newTList[ind])  }   tempList = newTList  }   outList.add(newTList)  curT += *parameters*.t   exitCondition = false  for (ind in tList.*indices*)  {  if (*abs*(tList[ind] - newTList[ind]) / newTList[ind] > *parameters*.epsilon)  {  exitCondition = true  break  }  }   tList = newTList  }   return Pair(outList, curT) }  fun main() {  val out = *simpleIteration*()   val xList = *mutableListOf*<Double>()  var i = 0.0  while (i < 2)  {  xList.add(i)  i += *parameters*.h  }   val plot = Plot2DPanel()  for (curY in out.first.*indices*)  {  if (curY % 4 == 0)  plot.addLinePlot("time = $curY", xList.*toDoubleArray*(), out.first[curY].*toDoubleArray*())  }   val frame = JFrame("T(x)")  plot.addLegend("SOUTH")   plot.setAxisLabel(0, "x, сантиметры")  plot.setAxisLabel(1, "T, Кельвины")  frame.setSize(1000, 1000)  frame.*contentPane* = plot  frame.*isVisible* = true   val secList = *mutableListOf*<Double>()  i = 0.0  while (i < out.second && secList.size != out.first.size)  {  secList.add(i)  i += *parameters*.t  }   val sPlot = Plot2DPanel()   var k = 0.0  while (k < *parameters*.l / 3)  {  val curList = *mutableListOf*<Double>()  for (curF in out.first)  curList.add((curF[(k / *parameters*.h).toInt()]))  sPlot.addLinePlot("len = %.1f".*format*(k), secList.*toDoubleArray*(), curList.*toDoubleArray*())   k += 0.1  }  sPlot.addLegend("SOUTH")  val newFrame = JFrame("T(t)")  newFrame.setSize(1000, 1000)  sPlot.setAxisLabel(0, "t, Секунды")  sPlot.setAxisLabel(1, "T, Кельвины")  newFrame.*contentPane* = sPlot  newFrame.*isVisible* = true } |