Бюджетное учреждение высшего образования
Ханты-Мансийского автономного округа — Югры
«СУРГУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт

Кафедра прикладной математики

РАССАДИН АЛЕКСАНДР АНДРЕЕВИЧ

Дисциплина «Уравнения математической физики»

ОТЧЁТ

Индивидуальное задание №1

Тема «Численное решение начально-краевых задач для одномерного уравнения теплопроводности»

Студент гр. №601-01
Рассадин А.А

Преподаватель
Гореликов А.В

Сургут 2022г.

Оглавление

Глава 1

Введение

На сегодняшний день благодаря техническому прогрессу и развитию точных наук люди научились описывать много вещей, происходящих в мире. В основе этих описаний лежат математические модели. В последнее время математическое моделирование широко используется во многих сферах человеческой деятельности. Оно позволяет достигать высоких показателей в промышленности, экономить природные и денежные ресурсы.

С другой стороны, нельзя недооценивать сложность правильного применения моделирования. Человек, занимающийся этим родом деятельности должен хорошо знать предметную область, обладать соответствующей математической подготовкой. Обязательным этапом при практическом применении компьютерного моделирования является расчёт модели. Следовательно, нужно знать устройство и особенности современных компьютеров и вычислительных систем. Наконец, важно владеть большим числом различных методов для того, чтобы в зависимости от характера задачи выбирать наилучший из них.

Нужно сказать, что моделирование это очень широкое понятие. Если рассматривать только математическое моделирование, то и в этой отрасли можно выделить ещё несколько разных сфер. К ним относятся: теоретические исследования — создание новых моделей, методы решения задач — разработка новых способов решения, а также правильная обработка результатов вычислений.

Данная работа относится к сфере методов решения задач, точнее к численному решению дифференциальных уравнений в частных производных. С помощью таких уравнений описывается множество явлений в физике и химии. Однако среди этих уравнений есть много таких, которые *невозможно решить аналитически*, поэтому применяют *численный* метод. Такие методы основаны на проведении многократных расчётов. Современные ЭВМ дают много возможностей для применения этих методов, поэтому они активно развиваются. Одним из таких методов — является метод контрольного объёма[1]. Будет показано, как он применяется при численном решении одной из классических задач: задаче о распространении тепла в тонком стержне. В результате будет создана программа, способная рассчитать распределение температуры в стержне с течением времени в зависимости от заданных условий. Результаты

[1] Часто в тексте он будет записан кратко как «МКО».

расчётов будут представлены в удобном для восприятия формате: в виде анимации. Основным критерием при тестировании должно быть соответствие точности данных, полученных программой с теоретической оценкой.

1.1 Постановка задачи

Главная цель данной работы — приобретение первичных навыков математического моделирования, а также получение первого опыта моделирования на основе этих знаний. Требуется изучить математическую модель переноса тепла в твёрдом теле. Ознакомиться с методом контрольных объёмов — одним из способов численного решения задач такого вида. Затем выполнить реализацию этого метода в виде расчётной программы. При разработке программы нужно сделать так, чтобы её выходные данные потом можно было визуализировать. На финальном этапе должно быть сделано тестирование созданной программы.

Глава 2

Математическая модель

2.1 Уравнение теплопроводности

Много физических явлений описываются с помощью дифференциальных уравнений с заданными дополнительными условиями. В этой части будет рассмотрено уравнение, которое описывает теплопроводность в твёрдом теле. Рассматривается простейший случай, когда тело это достаточно тонкий[1] стержень. Уравнение теплопроводности в этом случае имеет следующий вид:

$$\rho c \frac{\partial U(x,t)}{\partial t} \;=\; \frac{\partial}{\partial x}(k \frac{\partial U(x,t)}{\partial x}) \;+\; f(x,t) \tag{2.1}$$

Оно является частным случай общего уравнения для трёхмерного тела. Подробный вывод общего уравнения можно найти в [1]. Здесь ρ — плотность материала, из которого сделан стержень, c — удельная теплоёмкость, k — коэффициент теплопроводности (на сколько быстро тело способно менять температуру). U — искомая величина, зависящая от двух значений x и t, $f(x,t)$ - плотность тепловых источников[2], величина T — конечное время наблюдений. Коэффициенты ρ, c, k - записаны как константы, но на самом деле в реальных процессах бывает так, что и плотность ρ, и удельная теплоёмкость c, и коэффициент теплопроводности k могут быть функциями от $x,\ t$ или даже U, или вдобавок зависеть от температуры U. Примером такой ситуации может быть стержень, у которого части сделаны из разных материалов, в нём k принимает значения зависящие от положения в стержне, то есть $k \;=\; k(x)$. В данной работе будет предполагаться, что $\rho,\ c,\ k$ - постоянные величины.

Под решением уравнения (2.1) понимается функция $U(x,t)$, которая при подстановке её в уравнение (2.1) обращает его в тождество на заданном множестве изменения переменных x, t. В данной задаче это множество $x \in (0, l),\ t \in (0, T]$. Функция $U(x,t)$ описывает значение температуры стержня в точке x в момент времени t

Уравнение (2.1) получено из фундаментального закона: закона сохранения энергии. Его левая часть отражает изменение тепловой энергии тела. А слагаемые справа описывают факторы, которые вызывают это изменение. Первое из них возникает из-за теплообмена с внешней средой, который происходит благодаря диффузии, когда

[1]Предполагается, что в пределах поперечного сечения температура одинакова.
[2]Плотность тепловых источников — количество теплоты выделяемой/поглощаемой в единице объёма за 1 секунду.

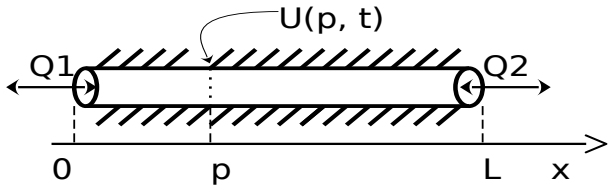


Рис. 2.1: Q_1, Q_2 показывают тепловые потоки на концах стержня. Штрихи обозначают, что через боковую поверхность теплообмена нет.

сталкиваются молекулы разных скоростей. Второе слагаемое — так называемый источниковый член. Он отражает вклад от внутренних источников/поглощателей тепла в теле. Всякая математическая модель учитывает лишь часть свойств объекта или явления. Поэтому говорят, что у модели есть *границы применимости*. В случае одномерного уравнения теплопроводности (2.1) можно говорить о неточности данной модели. В том смысле, что учитывается лишь теплообмен через концы стержня[3], а вся боковая поверхность считается теплоизолированной. Это показано на рис. 2.1.

Если для уравнения (2.1) существует решение $U(x,t)$, то ему в этом случае будет также удовлетворять и функция $U(x,t)+C$, где C — *произвольное* постоянное число. Таким образом, дифференциальное уравнение либо не имеет решений, либо их *бесконечное* число. В математической модели присутствуют кроме уравнений, также и дополнительные условия. С математической точки зрения они нужны, чтобы уравнением (2.1) функция $U(x,t)$ определялась *однозначно*.

2.2 Дополнительные условия

В разных математических моделях дополнительные условия могут отличаться. Для случая с явлением переноса тепла в твёрдом теле нужно задать *начальные* и *граничные* условия.

Начальные условия означают, что известно распределение температуры во всем теле в начальный момент времени и задаются таким образом:

$$U(x,t_0)=\varphi(x), \quad x\in[0;l] \tag{2.2}$$

Аналитически это распределение выражается функцией $\varphi(x)$. В дальнейшем всегда будет предполагаться, что $t_0=0$. Если $\varphi(x)\equiv 0$, то принято нулевые начальные условия называть *однородными*.

Кроме распределения температуры в начальный момент времени, для однозначного определения функции $U(x,t)$ нужно также задать граничные условия (ГУ) протекания процесса. То есть выражения, которые описывают температуру или тепловой поток.[4] на границе исследуемой области в течение всего эксперимента.

[3]Если учесть теплообмен ещё и через боковую поверхность, то в уравнении появится дополнительное слагаемое, содержащее искомую величину $U(x,t)$.

[4]Тепловой поток — физическая величина, численно равная количеству теплоты проходящей через единичную площадку за одну секунду.

Общий вид граничных условий в одномерном случае такой:

$$\alpha(p)\frac{\partial U(p,t)}{\partial x} \ + \ \beta(p)U(p,t) \ = \ \psi(p,t) \tag{2.3}$$

Точка p пробегает множество граничных точек, а $t > 0$. В данном случае граница области состоит из вдух точек - левого и правого концов стержня. $p \in \{0\} \cup \{l\}$. Производная в уравнении понимается к *односторонняя*. В зависимости от значений $\alpha(p),\ \beta(p)$. Выделяют особого три частных случая:

- $\alpha(p) \equiv 0$ — первый род, когда известна температура границы (задача Дирихле)
- $\beta(p) \equiv 0$ — второй род ГУ, когда задан тепловой поток (задача Неймана)
- $\alpha(p) \neq 0$ и $\beta(p) \neq 0$ — третий род ГУ, теплообмен через границу происходит по закону Ньютона-Рихмана (задача Робена)

Также как и в случае с нулевыми начальными условиями, когда выделяют случай однородных граничных условий. Математически доказано, что для уравнения (2.1) с начальными условиями (2.2) и граничными условиями (2.3), в случае существования решения, оно будет *единственным*. Нужно отметить, что необходимым условием существования решения является *сопряжение* граничных условий с начальными при $t \to 0$, поскольку в данной модели функция $U(x,t)$ - непрерывная.

2.3 Математическая модель теплопроводности

Выше были названы условия, при которых можно однозначно определить функцию $U(x,t)$ - описывающую температуру тонкого стержня. Эти условия позволяют *корректно*[5] описать процесс переноса тепла теле. Математическая модель, для теплопроводности в твёрдом теле в случае со стержнем записывается так:

$$\begin{cases} \rho c\dfrac{\partial U(x,t)}{\partial t} \ = \ \dfrac{\partial}{\partial x}(k\dfrac{\partial U(x,t)}{\partial x}) \ + \ f(x,t) \\ \alpha(p)\dfrac{\partial U(p,t)}{\partial x} \ + \ \beta(p)U(p,t) \ = \ \psi(p,t) \\ U(x,0) \ = \ \varphi(x) \\ \psi(p,0) \to \varphi(p), \quad t \to 0 \\ x \in (0;l), \quad t \in (0;T] \end{cases} \tag{2.4}$$

В такой постановке исходными данными задачи являются: длина стержня l, физические величины ρ, c, k или их аналитические выражения, конечный момент измерений T. А также функции $\varphi(x),\ \psi(p,t)$ - известные заранее.

Задачи вида (2.4) довольно хорошо изучены, для них есть известные способы нахождения точного решения. То есть определения функции $U(x,t)$ *аналитически, в виде*

[5]Корректно поставленная задача это такая, для которой решение есть, оно единственно и мало меняется при малом изменении дополнительных условий.

формулы.. Однако такая возможность есть *не всегда,* поэтому применяются *численные методы решения.* В данной работе будет рассмотрен случай, когда в граничные условия таковы, что $\alpha(p) \equiv 0$. Такие граничные условия относятся к первому тип, а задачу принято называть *задачей Дирихле.* Разработанная программа будет использовать метод контрольных объёмов для определения температуры. При этом считается, что величины ρ, c, k - постоянные, а источниковый член $f(x,t) \equiv f(x)$ - стационарный, то есть не зависит от времени.

2.4 Тестовая задача для проверки программы

Для поверки правильности работы программы её вывод будет сравниваться с тестовой задачей, для которой известно точное решение. В качестве такой задачи рассматривается следующая однородная задача Дирихле:

$$\begin{cases} \dfrac{\partial U(x,t)}{\partial t} = \dfrac{\partial^2 U(x,t)}{\partial x^2} + sin(x) \\ U(0,t) = 0 \\ U(\pi,t) = 0 \\ U(x,0) = 0 \\ x \in (0;\pi), \quad t \in (0;T] \end{cases} \tag{2.5}$$

Задачи такого вида решают так называемым «методом разделения переменных». Более подробную информацию о нём можно найти в любом учебнике по математической физике, например в [1]. Основная идея этого метода в том, что уравнение теплопроводности линейное, поэтому можно предположить, что решение представимо в виде $U(x,t) = T(t)X(x)$. Следующий шаг это «редукция» — представление исходной задачи в виде нескольких более простых. Обычно решают вспомогательную задачу, где учитывается только исходное уравнение и *нулевые* граничные условия. Ясно, что такая задача имеет множество решений, поэтому дальше нужно решение получается из требования заданного начальными условиями.

Для одномерной задачи теплопроводности, когда заданы нулевые граничные условия *первого* рода, и уравнение содержит источниковый член известно, что решение задачи (2.5) представляется в виде: $U(x,t) = \sum\limits_{k=1}^{\infty} V_k(t) sin(\frac{\pi k x}{l})$. Но в тестовой задаче $l = 1$, поэтому общий вид её решения такой $U(x,t) = \sum\limits_{k=1}^{\infty} V_k(t) sin(kx)$. Функции $sin(\frac{\pi k x}{l})$ всегда получаются после решения вспомогательной задачи[6]. Чтобы найти неизвестные функции V_k делают ещё одно предположение.

Как известно, (см.[2]) система $\{sin(\frac{\pi k x}{l})\}$ — является системой Фурье, то есть при определённых условиях можно представить функцию рядом вида $g(x) =$

[6]Для задач с другим типом *однородных* граничных условий будут свои тригонометрические функции. Для каждого случая они приведены в [1].

$\sum_{k=1}^{\infty} g_k sin(\frac{\pi k x}{l})$, который называется рядом Фурье функции $g(x)$, g_k - коэффициенты ряда. А так как искомая функция $U(x,t)$ тоже представляется рядом по синусам, то предполагают, что источниковый член можно тоже по ним разложить. Далее получается для каждой неизвестной $V_k(t)$ своё дифференциальное уравнение с дополнительным условием $V_k(0) = 0$ поэтому каждая функция определяется однозначно.

В случае с задачей (2.5) всё довольно просто. Разложение функции $sin(x)$ содержит единственный коэффициент равный единице. Единственное уравнение на функции $V(t)$, которое получается в этом случае такое: $V(t)' + V(t) = 1$ — линейное обыкновенное дифференциальное уравнение, которое решается, например, методом вариации постоянной. Этот метод состоит из двух шагов, решение однородного уравнение и определение неизвестной функции $\beta(t)$:
$V(t)' + V(t) = 0 \Rightarrow \frac{V(t)'}{V(t)} = -dt \Rightarrow |V(t)| = e^{-t}\beta(t)$
(Функция $V(t) = 0$ не подходит, так как она даёт такую $U(x,t)$, которая не является решением уравнения в (2.5).)
Второй этап - определение $\beta(t)$. Для этого подставляют полученную $V(t)$ в исходное неоднородное уравнение.
$-\beta(t)e^{-t} + \beta(t)'e^{-t} + e^{-t}\beta(t) = 1 \Rightarrow \beta(t)'e^{-t} = 1 \Rightarrow$
$\beta(t) = e^t dt \Rightarrow \beta(t) = e^t + C_1, \quad C_1 = const$
Константа C_1 определяется из условия:
$V(0) = 0 \Rightarrow e^{-0}(e^0 + C_1) = 1 + C_1 = 0 \Rightarrow C_1 = -1$ В силу однородности условий, видно, что $C_1 = 0$. В итоге $V(t) = 1 - e^{-t}$. Значит решением задачи (2.5) будет следующая функция:

$$U(x,t) = (1 - e^{-t})sin(x) \tag{2.6}$$

Это выражение позднее будет применяться при вычислении погрешностей счёта программы.

Глава 3

Численное решение дифференциальных уравнений. Метод контрольных объёмов для уравнения теплопроводности.

3.1 Преимущества и недостатки численных методов

Как можно видеть, в основе различных моделей лежат математические соотношения. Зачастую, возникают ситуации, когда необходимо решить какую-то практическую задачу, но найти решение *аналитически* никак не получается. В этих случаях прибегают к *численному* решению. Это значит, что в ходе решения будет многократно происходить обработка больших объёмов *числовых* данных, причём всё, что с ними будет сделано сводится в конечном счёте к арифметическим операциям. Поэтому, окончательным результатом применения численного метода является *набор числовых* значений. Одним из этапов математического моделирования является расчёт модели, и часто он делается численным методом. Основным преимуществом численного метода перед решением аналитическим способом является то, что класс задач, которые *можно* решить расширяется.

Следует сказать, что недостатки, перечисленные далее, не относятся напрямую к численным методам, а скорее обусловлены тем, что эти методы реализуются на компьютерах. Сегодня компьютеры позволяют быстро оперировать с внушительными объёмами данных, что даёт возможность рассчитывать всё более сложные модели. Но не нужно считать быстродействующие ЭВМ чем-то всемогущим. Всегда важно помнить о том, что при расчётах на компьютерах появляется масса трудностей. Среди них: неизбежные погрешности представления чисел, ошибки округления, возможные ошибки при реализации алгоритмов. Кроме того, после расчёта появляется большой массив данных, а правильная интерпретация результатов моделирования это отдельная не простая задача.

3.2 Основные идеи метода контрольных объёмов

Для решения задач теплопроводности существует *метод контрольных объёмов*. Подробно этот метод описан в книге [3]. Здесь будут рассмотрены некоторые основные

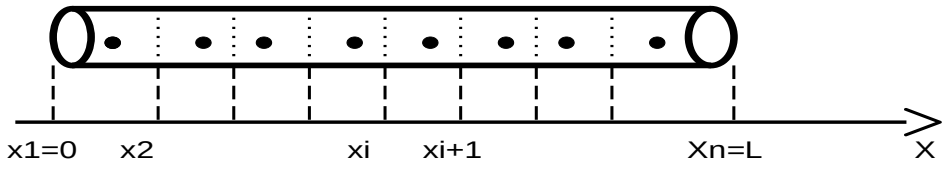


Рис. 3.1: Пример разбиения непрерывной области - $(0; l)$ на конечное число точек $X(i)$

положения как МКО, так и численного решения задач уравнений математической физики в целом.

Первым отличием численного решения от аналитического является то, что исходная задача заменяется на её аналог, который «представим» в ЭВМ. Переходят от непрерывной расчётной области в пространстве к её дискретному аналогу его ещё называют расчётной сеткой. То есть исходная область разделяется на *конечное* число подобластей. Эти подобласти называют контрольными объёмами, в каждом из них выбирается контрольная (узловая, расчётная) точка, она располагается в центре объёма. То же самое относится и ко времени: на заданном непрерывном отрезке времени выбирают промежуточные точки - временные слои. Ставится задача определить значение искомой величины уже в *конечном* наборе точек пространства на каждом временном слое. Итоговым решением является *набор сеточных значений искомой величины.*

Вторая идея всех численных методов и МКО в частности, состоит в том, что дифференциальное уравнение приводят в вид, который компьютер может обработать. Так как значения искомой величины определяются в заданных точках, то в итоге из исходного ДУ получается система линейных уравнений, по одному уравнению на каждую точку. Уравнения получаются за счёт того, что производные искомой функции определённым образом заменяют на конечные разностные отношения. Этот этап называется дискретизацией уравнения. Получить дискретный аналог ДУ можно разными способами, всё зависит от того какой вид разностного отношения выбрать для приближения производной. В таких ситуациях чаще всего руководствуются или качественными рассуждениями о протекании процесса, или получают наиболее точное представление математически. Причём бывает так, что для аппроксимации[1] разных (по разным переменным) производных от одной и той же функции используют различные выражения. Поскольку численные методы применяются для нахождения какого-то частного решения задачи, то всегда заранее известны некоторые дополнительные условия. Именно они и позволяют решить полученную систему *линейных* алгебраических уравнений.

Можно сказать, что метод контрольных объёмов имеет нечто общее с натурными экспериментами. Ведь в них исследователи также обрабатывают результаты измерений полученные в какие-то заданные моменты времени в выбранных точках. Одна-

[1] Аппроксимация — приближение

ко *существенная разница здесь в том, что результаты эксперимента представляют собой приближенные измерения реально происходящего процесса, в то время как всякий численный метод в результате отражает некоторую математическую модель.* В случае применения МКО к явлению теплопроводности в твёрдых телах считается, что при достаточно густом разбиении расчётной области все решения сходятся к одной общей величине вне зависимости от выбора способа аппроксимации ДУ. Кроме этого, в данном случае независимая переменная t будет так называемой *односторонней координатой.* Это значит, что величина температуры зависит лишь от того, какой она была на предыдущем временном слое. Поэтому можно последовательно двигаться от одного временного слоя к следующему и так далее, и в итоге определить температуру в нужный момент времени. Это обстоятельство позволяет эффективнее расходовать память при вычислениях.

3.3 Дискретизация расчётной области

В данной работе сетка будет строиться с равномерным шагом как по координатам, так и по времени. Исходными данными для построения дискретного аналога исследуемой области в одномерном случае будут значения:

- N_v — Число контрольных объёмов
- l — Длина стержня
- T_{end} — Конечное значение времени
- Nt — Количество временных слоёв

Пространственная область, т.е. отрезок $[0;\ l]$ разбивается на N_v контрольных объёмов[2]. В результате можно определить набор значений координат граней контрольных объёмов (в данном случае грани это точки). В каждом контрольном объёме располагается одна узловая точка, в ней определяется значение температуры. На граничные точки тоже приходятся соответствующие узловые точки, чтобы знать температуру границы. Таким образом общее число граней контрольных объёмов N_f[3] равно $N_v + 1$, а узловых точек N_p[4] составляет $N_v + 2$. В МКО предполагается, что каждая узловая точка располагается по середине контрольного объёма. Разбиение временной области тоже делается с равномерным шагом. Его значение можно определить равномерно, например, по конечному времени T_{end} и числу N_t, тогда шаг по времени $dt = \frac{T_{end}}{N_t}$. Либо другим способом, для неравномерной стеки.

[2]Можно считать, что объёмы имеют вид $a \times 1 \times 1$, где a - это расстояние между гранями контрольных объёмов, причём $a \gg 1$.
[3]N_f выбрано для обозначения числа граней, от слова «faces» - в одном из значений это грань.
[4]N_p индекс $_p$ выбран как сокращение от «points» - точки

3.4 Дискретный аналог дифференциального уравнения

Рассматривается способ получения дискретного аналога следующего уравнения.

$$\rho c \frac{\partial U(x,t)}{\partial t} \;=\; \frac{\partial}{\partial x}\left(k \frac{\partial U(x,t)}{\partial x}\right) \;+\; f(x,t) \tag{3.1}$$

На рисунке 3.2 изображен один из *внутренних* контрольный объёмов. В дальнейшем

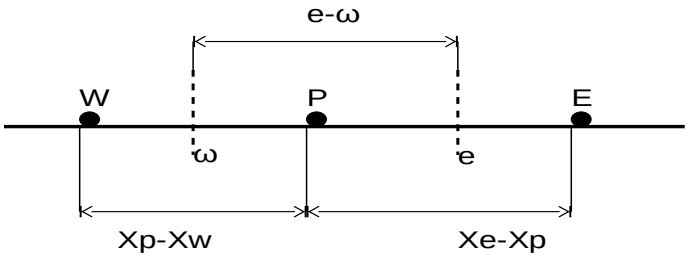


Рис. 3.2: Изображён типичный внутренний объем с границами $\omega,\ e$ и узловой точкой p.
Обозначения точек выбраны специально: W «west» - западная точка, E «east» - восточная.

будет удобно принять обозначения: $T(x) := T(x,t),\ T^0(x) := T(x,t_0)$. Значение температуры во всех узловых точках на предыдущем временном слое считается известными[5]. Уравнение (**??**) можно проинтегрировать в пределах рассматриваемого объёма, то есть при $x \in [\omega, e]$ и за промежуток времени от t_0 до t. Удобнее начать с левой части уравнения(**??**):

$$\rho c \frac{\partial T(x,\tau)}{\partial x} \;\Rightarrow\; \rho c \int\limits_{\omega}^{e} dx \int\limits_{t_0}^{t} \frac{\partial T(x,\tau)}{\partial x}\, d\tau$$

При вычислении внутреннего интеграла $\int\limits_{t_0}^{t} \frac{\partial T(x,\tau)}{\partial x}\, d\tau$ переменной интегрирования является τ она принимает множество значений от t_0 до t. При этом пространственная координата x *фиксированная*. Интегрирование избавляет от производной, поэтому получается выражение:

$$\rho c \int\limits_{\omega}^{e} T(x,\tau)|_{t_0}^{t} dx \;=\; \rho c \int\limits_{\omega}^{e} \left(T(x,t) - T(x,t_0)\right) dx$$

В полученном выражении стоит разность двух функций от переменной x: $T(x,t),\ T(x,t_0)$, где $x \in [\omega;\ e]$. На этом этапе делается предположение, что в *фиксированный момент времени* температура в пределах всего рассматриваемого объёма равна значению температуры в узловой точке p. Такая ситуация изображена на рисунке (а). В результате получается:

$$\rho c \int\limits_{\omega}^{e} dx \int\limits_{t_0}^{t} \frac{\partial T(x,\tau)}{\partial x}\, d\tau \;=\; \rho c \int\limits_{\omega}^{e} T(x,\tau)|_{t_0}^{t} dx \;=\; \rho c(e-\omega)\left(T(p,t) - T(p,t_0)\right)$$

[5]Потому что известны начальные условия.

Таким образом, интегрирование левой части (3.1) приводит к такому результату:

$$\rho c(e-\omega)\left(T(p)-T^0(p)\right) \tag{3.2}$$

На этапе интегрирования правой части можно, поменять порядок взятия интегралов, так как $T(x,t)$ — непрерывная функция.

$$k\frac{\partial}{\partial x}\left(\frac{T(x,t)}{\partial x}\right) \Rightarrow k\int\limits_{t_0}^{t} d\tau \int\limits_{\omega}^{e}\left(\frac{\partial}{\partial x}(\frac{\partial T(x,\tau)}{\partial x})+f(x)\right)dx =$$

$$= k\int\limits_{t_0}^{t} d\tau \int\limits_{\omega}^{e}\left(\frac{\partial T(x,\tau)}{\partial x}\right)dx + \int\limits_{t_0}^{t} d\tau \int\limits_{\omega}^{e} f(x)dx$$

Интегрирование первого слагаемого правой части:

$$k\int\limits_{t_0}^{t} d\tau \int\limits_{\omega}^{e}\left(\frac{\partial}{\partial x}(\frac{\partial T(x,\tau)}{\partial x})\right)dx = k\int\limits_{t_0}^{t}\left(\frac{\partial T(x,\tau)}{\partial x}\Big|_{\omega}^{e}\right)d\tau$$

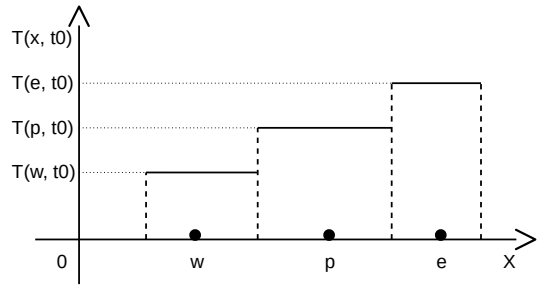
В этом случае изменяться будет значение x, то есть значения температуры в фиксированный момент времени в разных точках объёма. На этот раз считается, что в заданный момент времени при переходе от одной точки к другой температура меняется линейно. Это изображено на рис.(b) Поэтому значения производной на границах контрольного объёма можно приблизить разностным отношением температуры в узловых точках в определённый момент времени τ:

$$\frac{\partial T(e,\tau)}{\partial x} = \frac{T(e,\tau)-T(p,\tau)}{x_e-x_p}; \quad \frac{\partial T(\omega,\tau)}{\partial x} = \frac{T(p,\tau)-T(\omega,\tau)}{x_p-x_w}$$

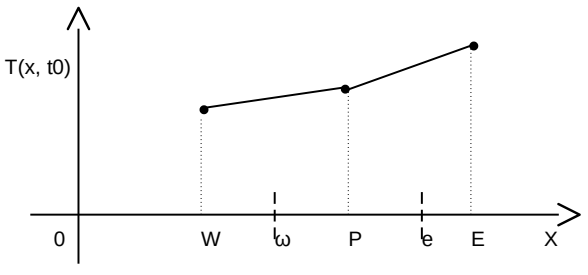
Первое слагаемое правой части принимает вид:

$$k\int\limits_{t_0}^{t}\left(\frac{\partial T(x,\tau)}{\partial x}\Big|_{\omega}^{e}\right)d\tau = k\int\limits_{t_0}^{t}\left(\frac{T(e,\tau)-T(p,\tau)}{x_e-x_p}-\frac{T(p,\tau)-T(\omega,\tau)}{x_p-x_w}\right)d\tau$$

Получена разность четырех функций от одной переменной - времени. Есть несколько



(a) Вид функции $T(x,t)$ при интегрировании по объёму, когда t - фикс.



(b) Вид функции $T(x,t)$ при интегрировании по времени, когда x - фикс.

способов решить полученный интеграл. Все они основаны на применении *теоремы о среднем* и различаются предположением о профиле $T(p_0, \tau), p_0$-фикс. Особо известны схемы: явная, неявная, Кранка-Николсона. В данной работе используется схема Кранка-Николсона. Согласно этой схеме температура в фиксированной точке меняется линейно, поэтому средним значением будет среднее арифметическое из значений температуры в моменты t_0, t. График изменения температуры в таком случае показан на рис.3.3. Выбор схемы влияет не только на способ решения, но и на требования к сетке. (Подробнее написано в [3].) Считается, что выбранная схема на равномерной сетке, при соотношении N_v к N_t равном один к одному обеспечивает *второй* порядок точности. Это значит, что когда число объёмов и временных слоёв удваивают, то ошибка решения уменьшается в *четыре* раза. Результат интегрирования правой части:

$$k \int_{t_0}^{t} \left(\frac{\partial T(x,\tau)}{\partial x}\bigg|_{\omega}^{e} \right) d\tau \;=\; \frac{k(t-t_0)}{2} \left(\frac{T(e)-T(p)}{x_e - x_p} + \frac{T^0(e)-T^0(p)}{x_e - x_p} \right) \;- \\ \frac{k(t-t_0)}{2} \left(\frac{T(p)-T(w)}{x_p - x_w} + \frac{T^0(p)-T^0(w)}{x_p - x_w} \right) \tag{3.3}$$

Наконец, дошло дело и до второго слагаемого правой части:

$$f(x) \;\Rightarrow\; \int_{t_0}^{t} d\tau \int_{\omega}^{e} f(x)dx$$

Так как здесь случай со стационарным источником, то при интегрировании снова можно воспользоваться теоремой о среднем, выбирая постоянное значение $\overline{f_p}$, для данного объёма. Получается выражение вида:

$$(t-t_0)(e-\omega)\overline{f_p} \tag{3.4}$$

В условии тестовой задачи (на стр.8) известно аналитическое выражение для источникового члена уравнения, поэтому самым лучшим вариантом будет использовать точное выражение интеграла, то есть:

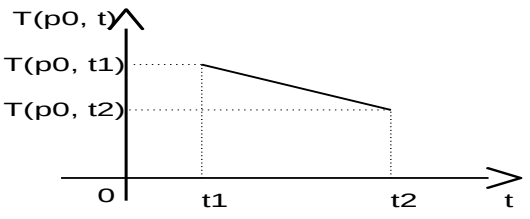


Рис. 3.3: Изменение температуры T в фикс. точке p_0 за время от t_0 до t согласно схеме Кранка-Николсона.

$$\int_{t_0}^{t} d\tau \int_{\omega}^{e} sin(x)dx \;=\; \int_{t_0}^{t} d\tau \left(-cos(e)+cos(\omega)\right) \;=\; (t-t_0)\left(-cos(e)+cos(\omega)\right) \qquad (3.5)$$

Ранее было сказано, что при переходе от дифференциального уравнения к его дискретному представлению получается система *линейных* уравнений. Как видно сейчас, после интегрирования получаются выражения, содержащие известные значения и искомую величину T. Приравнивание результатов интегрирования левой (3.2) и правой частей (3.3, 3.4) уравнения (3.1) позволяет выразить температуру в узловой точке данного контрольного объёма. Перед этим удобно сразу разделить обе части на $(t-t_0)$.

$$\frac{\rho c(e-\omega)}{t-t_0}\left(T(p)-T^0(p)\right) \;=\; \frac{k}{2}\left(\frac{T(e)-T(p)}{x_e-x_p} + \frac{T^0(e)-T^0(p)}{x_e-x_p}\right) -$$

$$-\frac{k}{2}\left(\frac{T(p)-T(w)}{x_p-x_w}+\frac{T^0(p)-T^0(w)}{x_p-x_w}\right) - cos(e)+cos(\omega) \;\Rightarrow$$

$$\Rightarrow \left(\frac{\rho c(e-\omega)}{t-t_0}+\frac{k}{2(x_e-x_p)}+\frac{k}{2(x_p-x_w)}\right)T(p) \;=$$

$$= \frac{kT(e)}{2(x_e-x_p)}+\frac{kT(w)}{2(x_p-x_w)}+\frac{T^0(e)-T^0(p)}{x_e-x_p}+\frac{T^0(p)-T^0(w)}{x_p-x_w} - cos(e)+cos(\omega) \qquad (3.6)$$

Видно, что в последнем выражении (3.6) получена зависимость в температуры в точке p от значений её значений в ближайших соседних точках. Для сокращения записей удобно использовать обозначения коэффициентов, как на рис. 3.2. со стр.13. Буквенные индексы в обозначениях выбраны для удобства восприятия, так w (от англ. «west») - западная точка, аналогично e (от англ. «east») - восточная точка.

$ae(i) \;=\; \frac{k}{2(x_e-x_p)}$

$aw(i) \;=\; \frac{k}{2(x_p-x_w)}$

$ap(i) \;=\; \frac{\rho c(e-\omega)}{t-t_0} + ae(i) + aw(i)$

$bp(i) \;=\; ae(i)\frac{T^0(e)-T^0(p)}{x_e-x_p} - aw(i)\frac{T^0(p)-T^0(w)}{x_p-x_w} + \frac{\rho c(e-\omega)T^0(p)}{t-t_0} - cos(e)+cos(\omega)$

В итоге получается компактная запись (3.7) для уравнения на температуру во *внутренней* точке p на следующем временном слое со значением времени равным t. Индексом i нумеруются узловые точки. Левой границе соответствует $i=1$, а правой $i=N_v+2$. (см. обозначения на стр.12)

$$ap(i)T(i) \;=\; ae(i)T(i+1)+aw(i)T(i-1)+bp(i), \quad i=\overline{2,n-1} \qquad (3.7)$$

Осталось получить выражения для температуры в граничных точках. Для левой границы его можно получить из (3.6):

$ap(1)T(1) \;=\; ae(1)T(2)+aw(1)T(0)+bp(1)$

Но значения $T(0)$ не существует, поэтому всегда $aw(1)\equiv 0$. Тогда выражение для

температуры на левой границе такое:
$ap(1)T(1) \;=\; ae(1)T(2) + bp(1).$
Аналогично для правой границы:
$ap(n)T(n) \;=\; aw(n)T(n-1) + bp(n).$
Таким образом, получено N_p уравнений для значений температуры в каждой из N_p расчётных точек. Величины $aw(i), ap(i), ae(i)$ и $bp(i)$, где $i = \overline{1,n}$ известны. Поэтому расчёт температуры на следующем временном слое при значении времени равном t сводится к решению системы *линейных* уравнений.

3.5 Решение системы уравнений

В процессе дискретизации дифференциального уравнения мы получили вот такую систему *линейных* алгебраических уравнений.

$$\begin{cases} ap(1)T(1) \;=\; ae(1)T(2) + bp(1) \\ ap(2)T(2) \;=\; ae(2)T(3) + aw(2)T(1) + bp(2) \\ ap(3)T(3) \;=\; ae(3)T(4) + aw(3)T(2) + bp(3) \\ \dots \\ ap(n-2)T(n-2) \;=\; ae(n-2)T(n-1) + aw(n-2)T(n-3) + bp(n-2) \\ ap(n)T(n) \;=\; aw(n)T(n-1) + bp(n) \end{cases} \tag{3.8}$$

Видно, что её матрица довольно своеобразная. Ненулевые значения сконцентрированы рядом с главной диагональю. Существует *прямой* метод решения СЛАУ такого вида. У его принято называть методом *прогонки*[6]. Из первого уравнения системы (3.8) видно, что значение $T(1)$ можно выразить через $T(2)$. То есть:
$ap(1)T(1) \;=\; ae(1)T(2) + bp(1) \Rightarrow$
$T(1) = \frac{ae(1)T(2)}{ap(1)} + \frac{bp(1)}{ap(1)} \;=\; P(1)T(2) + Q(1), \quad P(1) \;=\; \frac{ae(1)}{ap(1)}, \quad Q(1) \;=\; \frac{bp(1)}{ap(1)}$
Из второго уравнения температура $T(2)$ определяется значениями в соседних точках $T(1),\ T(3)$, но можно $T(1)$ представить через $T(2)$. Связь $T(1)$ и $T(2)$ даёт следующее:
$ap(2)T(2) \;=\; ae(2)T(3) + aw(2)T(1) + bp(2) \;=\; aw(2)\left(P(1)T(2) + Q(1)\right) + bp(2) \Rightarrow$
$\left(ap(2) - aw(2)P(1)\right)T(2) \;=\; ae(2)T(3) + aw(2)Q(1) + bp(2) \Rightarrow$
$T(2) \;=\; P(2)T(3) + Q(2)$, где $P(2) \;=\; \frac{ae(2)}{ap(2)-aw(2)P(1)},\ Q(2) \;=\; \frac{aw(2)Q(1)+bp(2)}{ap(2)-aw(2)P(1)}$
Ясно, что продолжая такие же рассуждения можно получить зависимость температуры

$$T(i)T(i+1T(i) = P(i)T(i+1) + Q(i) \tag{3.9}$$

Уравнение выше позволяет выразить температуру $T(i)$ для*каждой внутренней* точки $i = \overline{2 = n-1}$ температуру справа, то есть $T(i+1)$.
С другой стороны известно представление $T(i)$ в виде, вкотором оно записано в системе (3.8)
$ap(i)T(i) \;=\; ae(i)T(i+1) + aw(i)T(i-1) + bp(i)$

[6]В иностранной литературе из-за трёхдиагонального вида матрицы этот алгоритм называется TDMA

В последнем уравнении можно исключить $T(i-1)$

$$ap(i)T(i) = ae(i)T(i+1) + aw(i)\left(P(i-1)T(i) + Q(i-1)\right) + bp(i) \Rightarrow$$
$$\left(ap(i) - aw(i)P(i-1)\right)T(i) = ae(i)T(i+1) + aw(i)Q(i-1) + bp(i) \Rightarrow$$

$$\Rightarrow T(i) = \frac{ae(i)T(i+1)}{ap(i) - aw(i)P(i-1)} + \frac{aw(i)Q(i-1) + bp(i)}{ap(i) - aw(i)P(i-1)} \tag{3.10}$$

Для температуры $T(i)$ найдено два представления, значит из (3.10) можно найти формулы для коэффициентов $P(i), Q(i)$ в (3.9) при $i = \overline{2, n-1}$

В этом состоит первая часть алгоритма прогонки — прямой ход, когда вычисляются коэффициенты $P(i), Q(i)$. Но решить выражение не получается, так как неизвестно значение $T(i+1)$. В общем случае, при численном решении уравнения теплопроводности после того, как выполнен этап прямого хода значение $T(n)$. В случае для *первой* начально-краевой задачи, получается, что известны значения температуры на границах. Тестовая задача (2.5, стр.(8)) однородная, что означает $T(n) \Rightarrow ap(n) = P(n) = Q(n) = 0$. Теперь, можно найти значение $T(n-1)$, через него получается $T(n-2)$ ну и так далее до $T(2)$, а значение $T(1)$ задано из граничных условий. Такая процедура называется *обратным* ходом и с её помощью вычисляются значения температуры в каждой узловой точке *одного* временного слоя. Всего временных слоёв N_t.

3.6 Алгоритм метода прогонки

1. Пользуясь граничными условиями, вычислить: $P(1),\ Q(1)$.
2. Прямой ход — определение величин:
   $P(i) = \frac{ae(i)}{ap(i) - aw(i)P(i-1)}, \quad Q(i) = \frac{aw(i)Q(i-1) + bp(i)}{ap(i) - aw(i)P(i-1)}$
3. Получить значение $T(N)$, используя граничные условия.
4. Выполнить обратный ход, находя $T(i)$ по формуле:
   $T(i) = P(i)T(i+1) + Q(i)$

Глава 4

Описание программной реализации МКО

4.1 Общая схема программы

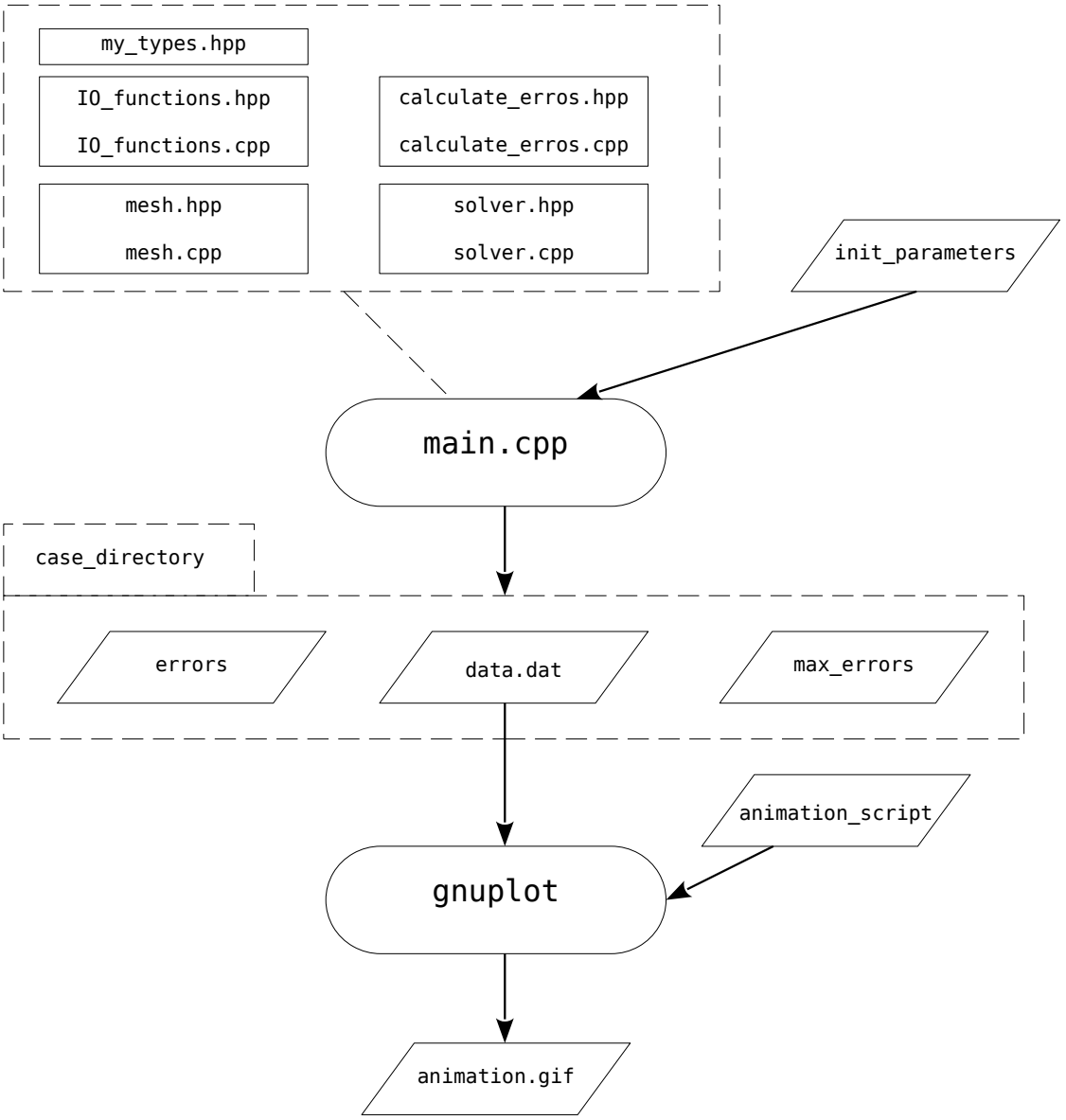


Рис. 4.1: Структура вычислительной программы, её взаимодействие с утилитой gnuplot.

Описание взаимодействия пользователя с программой. Для того, чтобы провести расчёт первой начально-краевой задачи, нужно передать входные данные. Это делается с помощью текстового файла. В файле необходимо перечислить параметры задачи в *соответствии со следующим форматом:*

- «значение длины»
- «количество контрольных объёмов»
- «конечное время»
- «число временных слоёв»
- «плотность материала»
- «удельная теплоёмкость»
- «коэффициент теплопроводности»

При этом, для удобства в начале каждой строки можно подписать, какой именно параметр в ней записан, например $\rho = 1000$ — допустимая запись. После этого нужно вызвать исполняемый файл программы передав ей в качестве первого аргумента имя файла с исходными данными, а вторым аргументом ввести имя директории, где будут сохранены результаты расчётов. Для визуализации вычислений используется свободно распространяемая программа gnuplot. Специально для создания анимаций на основе расчётных данных был создан специальный скрипт[1] для gnuplot. По задумке этот скрипт поставляется вместе с вычисляющей программой. Поэтому пользователю, находящемуся в каталоге с результатами расчётов, нужно лишь перенаправить в gnuplot команды из скрипта. Это делается очень просто: вызывается gnuplot и аргументом указывается имя скрипта. В итог будет сформирован файл с анимацией в формате «.gif». Для изменения параметров анимации нужно всего лишь отредактировать нужные поля в текстовом файле скрипта. Теперь, когда ясно как с этим *работать,* нужно объяснить как это *работает.*

Вся расчётная программа написана на языке C++ с использованием стандартных библиотек языков C и C++. Выбор этого языка объсяется тем, что C++ хорошо подходит для расчётов, благодаря своей быстроте. Он позволяет создавать достаточно оптимизированные программы, так как в нём есть возможность прямого контроля памяти. В данном случае эффективное взаимодействие с памятью имеет значение. Поскольку для представления дискретных наборов значений хорошо подходят массивы, то они и применяются для представления расчётной сетки, набора значений численного и эталонного решений, величин погрешностей. Но *пользователь может указывать разное* число расчётных точек, поэтому статические массивы не пригодны. В созданной программе память под все массивы выделяется *динамически.* А C++ позволяет напрямую контролировать этот процесс: выделять ровно столько, сколько нужно, и освобождать память, как только она больше не нужна. С самого начала было решено,

[1]скрипт - в данном случае это файл с заготовленным набором команд для программы gnuplot.

что программа будет состоять из модулей. В верхней части схемы на странице 19 в пунктирной рамке изображены 5 модулей, которые подключаются к модулю «`main`», содержащему главную функию. В самом первом модуле «`my_types`» описаны структуры данных, которые используются при ведении вычислений. Объединение нескольких базовых типов данных в один составной - структуру позволяет сократить объём кода, а также сделать его восприятие более доступным.

Основное удобство представления программы в виде нескольких обособленных частей состоит в том, что в этом случае существенно упрощается модернизация программы, а следовательно, расширяются её возможности. Например, если будет решено сделать неравномерную расчётную сетку, то потребуется лишь изменить содержимое модуля «`mesh`»[2], а лучше сделать две версии и в разных ситуациях использовать наиболее подходящую. Кроме этого в таком формате код лучше воспринимается, ведь каждый модуль отвечает за конкретную задачу. Если код программы большой, то неудобно просматривать его. А когда программа разбита на модули, то программисту нужно смотреть лишь код интересующего его модуля, что заметно проще. Нужно отметить, что модуль может состоять как из одного, так и из двух файлов. Если в модуле реализованы функции, которые будут использовать другими частями программы вне его, то синтаксис языка C++ требует, объявления этих переменных в тех частях программы, где они используются, а их реализации могут располагаться в другом файле. В этой программе «парных модулей четыре».
Краткое описание предназначения модулей.

- «`my_types`» — В нём введены новые структуры данных, представляющие: сетку, физические константы, массивы с погрешностями.
- «`IO_functions`» — Этот модуль отвечает за обработку файла с исходными данными задачи, а также вывод результатов работы программы.
- «`mesh`» — В нём реализованы функции для создания равномерной расчётной сетки.
- «`solver`» — В данном модуле реализован алгоритм[3] решения системы уравнений, которая позволяет определить температуру на следующем временном слое.
- «`calculate_errors`» — Эта часть программы отвечает за вычисление абсолютных и относительных погрешностей. После этапа тестирования её можно не подключать.
- «`main`» — «Главная подпрограмма» в ней происходит вызов функций из всех остальных функций, описанных в подключаемых модулях.

[2] от англ. *mesh* - сеть
[3] Имеется ввиду описанный выше алгоритм ТДМА.

4.2 Модуль «my_types»

```
1  /* В данном файле описаны структуры данных,
2  используемые при рассчётах. */
3  #ifndef __MY_TYPES__
4  #define __MY_TYPES__
5
6  //uns-сокращенное имя для неотрицательных целых чисел
7  typedef unsigned uns;
8
9  //Структура для хранения физических констант
10 struct physConsts {
11   double ro; //плотность
12   double c; //удельная теплоёмкость
13   double k; //коэффициент теплопроводности
14 };
15
16 //Дискретный аналог фазового пространства
17 //В нём хранится рассчётная сетка, а также её параметры
18 struct discrMesh {
19   double len; //Длина стержня (расчётной области)
20   uns Nv; //Число контрольных объёмов
21   uns Np; //Число контрольных (узловых) точек
22   uns Nt; //Количество временных слоёв
23   double *Faces; //Координаты граней конт. объёмов
24   double *Nodes; //Координаты узловых точек
25   //Массивы с координатами имеют общую длину.
26   //В нек. из них есть неиспользуемые значения, они равны NAN
27   double T_end; //Конечное время
28   double *cur_t; //Значение времени на текущем слое
29   double dt; //Шаг по времени
30 };
31
32 //Структура для хранения данных о погрешностях вычислений
33 struct errors {
34 //Знач. абс. погр. на тек. врем. слое для каждой узловой точки
35   double *abs;
36
37 //Макс. знач. абс. погрешности для теку врем. слоя
38   double maxAbs;
39
40 //Знач. относит. погр. на тек. врем. слое для каждой узловой точки
41   double *rel;
42
43 //Макс. знач. относит. погрешности для теку врем. слоя
44   double maxRel;
45 };
46 #endif
```

Далее будет приведён исходный код каждого модуля, а также скрипта для анимации. Поясняющие комментарии написаны прямо в коде.

4.3 Модуль «IO_functions»

```
/* Данный Заголовочный файл содержит объявления (прототипы) функций,
которые используются для получения исходных данных задачи
из файла, заданного пользователем. А также функции вывода
результатов расчёта и погрешностей в отдельные файлы */

#ifndef __IO_FUNCTIONS__
#define __IO_FUNCTIONS__
#include "my_types.hpp"

//Проверяет корректность выделения блока памяти
void checkMemAlloc(void *pointer, const char *ErrMsg);

//Вспомогательная функция, используется
//при обработке файла с исх. данными задачи (кейса)
bool isNotDigitOrSign(const char ch);

//Получения значений физических констант (ro, c, k)
//и параметров сетки (Nv, Tend, Nt, len)
//Файл с исходными данными задачи передаётся пользователем как аргумент
void getCaseParameters(const char *fileName,
                       physConsts & consts, discrMesh & mesh);

//Выводит рассчитанные (методом МКО) и точные значения температуры
// в файл "data" (по всем слоям в один)
//Между каждым временным слоем разделение в виде двух пустых строк
void printDataToFile(const discrMesh &mesh,
                     double *curT_exact, double *curT);

//Выводит значения для каждого погрешностей в файл "errors"
void printErrorsOnTimeLayerToFile(const discrMesh &mesh,
                                  const errors &errs, uns t);

//Выводит максимальные значения погрешностей в файл "MaxErrors"
//(по одному значению с каждого временного слоя)
void printMaxErrorsToFile(const discrMesh &mesh,
                          const errors &errs, uns t);
#endif
```

```
/* В данном файле реализованы функции для обработки
файла с иходными данными задачи.
А также функции для вывода результатов в файлы */

#include "my_types.hpp"
#include <cstdlib>
#include <cstdio>

void checkMemAlloc(void *pointer, const char *ErrMsg);

//Вспомогательная функция, используемая при обработке вход.о файла.
//Она определяет чем является символом, цифрой или нет.
bool isDigitOrSign(const char ch) {
```

```
14    char digits[13] = { '-', '+', '.', '0', '1',
15                        '2', '3', '4', '5', '6',
16                        '7', '8', '9'};
17    bool result = false;
18    for(int i = 0; i < 13; i++)
19      if(ch == digits[i]) {
20        result = true;
21        break;
22      }
23    return result;
24  }
25
26  //В этой процедуре реализовано заполнение входных парам. задачи
27  //из заданного пользователем файла.
28  void getCaseParameters(const char *fileName,
29                         physConsts &consts, discrMesh &mesh) {
30    uns maxSize = 256; //Максимальная длина для считывания строки
31    uns linesCounter = 0; //Счётчик числа считанных строк
32    uns paramsNum = 7; //Сколько параметров должно быть файле
33
34    FILE *fp = fopen(fileName, "r");
35    if(fp == nullptr) {
36      printf("Error with opening file \"%s\"\n", fileName);
37      exit(EXIT_FAILURE);
38    }
39
40    //Обработка файла происходит построчно
41    char *line = new char [maxSize];
42    checkMemAlloc(line, "Error in function getCaseParameters\n");
43    char *numStr; //В неё заносится число в виде набора символов
44
45
46    while(1) {
47      //fgets() - возвращает nullptr в двух случаях!
48      //1)если содерж. файла закончилось и считывать больше нечего
49      //2)если возникла ошибка считывания
50      if(fgets(line, maxSize-1, fp) == nullptr) {
51        //Считывание прервано
52        //Либо файл закончился, либо неправильный файл
53        if(linesCounter != paramsNum) {
54        //Если в файле не верное число строк,
55        //то в этом случае программа завершается.
56          printf("Error with reading file \"%s\"!\n", fileName);
57          exit(EXIT_FAILURE);
58        }
59        break;
60      }
61
62      linesCounter += 1; //Увеличение счётчика считанных строк
63      int j = 0; //Индекс символа в текущей считанной строке
64
65  //В массив numStr будут записаны численые знач. параметров посимвольно
```

```
66      numStr = new char [maxSize];
67      checkMemAlloc(numStr, "Error in func \"getCaseParameters\"\n");
68
69      //Отсечение лишних символов, до первой цифры или знака (+/-/.)
70      while(isDigitOrSign(line[j]) == false)
71        j++;
72
73
74      int i = 0; //Индекс символа в массиве numStr
75      //j - индекс символа в считанной строке line
76      while(line[j]) {
77        if(line[j] == '\n') {
78          numStr[i] = '\0';
79          break;
80        }
81        numStr[i] = line[j];
82        i++; j++;
83      }
84
85      switch (linesCounter) {
86  //Заполнение соответсвующих полей структур.
87  //В такой реализации входной файл должен быть оформлен в спец. виде.
88  //Чтобы записанные в нём значения вводились в подходящие поля.
89        case 1: { mesh.len = atof(numStr); break; }
90        case 2: {
91          mesh.Nv = atoi(numStr);
92          mesh.Np = mesh.Nv + 2;
93          break;
94        }
95        case 3: { mesh.T_end = atof(numStr); break; }
96        case 4: { mesh.Nt = atoi(numStr); break; }
97        case 5: { consts.ro = atof(numStr); break; }
98        case 6: { consts.c = atof(numStr); break; }
99        case 7: { consts.k = atof(numStr); break; }
100 //Если строк больше, чем предполагается, то обработка завершается
101       default: {
102         delete [] line;
103         delete [] numStr;
104         fclose(fp);
105         printf("Error with reading file \"%s\"!\n", fileName);
106         exit(EXIT_FAILURE);
107        }
108     }
109     delete [] numStr;
110   }
111   delete [] line;
112   fclose(fp);
113 }
114
115
116 //Функция для вывода расчётных данных файл "data.dat"
117 //Они используются для визуализации результатов.
```

```
118 void printDataToFile(const discrMesh &mesh,
119                       double *curT_exact, double *curT) {
120   static bool first = true;
121   FILE * fp = fopen("data.dat", "a+");
122   if(fp == nullptr) {
123     printf("Erorr with opening file to write data\n");
124     exit(EXIT_FAILURE);
125   }
126
127   if(first) {
128     //"Шапка", чтобы было понятно какая колонка за что отвечает
129     fprintf(fp, "#   X      T      T_exact\n");
130     first = false;
131   }
132
133   for(uns i = 0; i < mesh.Np; i++)
134     fprintf(fp, "%lf  %lf   %lf\n", mesh.Nodes[i], curT[i], curT_exact[i]);
135   //Разделение между данными на каждом слое в две пустых строки
136   fprintf(fp, "\n\n");
137   fclose(fp);
138 }
139
140 //Процедура, выводящая погрешности в каждой расчётной точке
141 //в отдельный файл "errors"
142 void printErrorsOnTimeLayerToFile(const discrMesh &mesh,
143                                   const errors &errs, uns t) {
144   static bool first = true;
145   FILE * fp = fopen("errors", "a+");
146   if(fp == nullptr) {
147     printf("Erorr with opening file to write errors\n");
148     exit(EXIT_FAILURE);
149   }
150
151   if(first) {
152     //"Шапка", чтобы было понятно какая колонка за что отвечает
153     fprintf(fp, "#   X  AbsErr  RelErr [%%]\n");
154     first = false;
155   }
156
157   for(uns i = 0; i < mesh.Np; i++) {
158     if (i == 0)
159       fprintf(fp,"#time: %.3lf\n", mesh.cur_t[t]);
160     fprintf(fp,"%.3lf   %.2lf   %.2lf\n", mesh.Nodes[i],errs.abs[i],errs.rel[i]);
161   }
162   fprintf(fp, "\n\n"); //Разделение между слоями в две пустые строки
163   fclose(fp);
164 }
165
166 //Выводит максимальную погрешность на каждом слое в файл "MaxErrors"
167 void printMaxErrorsToFile(const discrMesh &mesh,
168                           const errors &errs, uns t) {
169   static bool first = true;
```

```
170     FILE * fp = fopen("MaxErrors", "a+");
171     if(fp == nullptr) {
172       printf("Erorr with opening file to write errors\n");
173       exit(EXIT_FAILURE);
174     }
175
176     if(first) {
177       fprintf(fp, "#time  MaxAbs  MaxRel [%%]\n");
178       first = false;
179     }
180
181     fprintf(fp, "%.3lf   %.2lf   %.2lf\n", mesh.cur_t[t], errs.maxAbs, errs.maxRel);
182     fclose(fp);
183   }
```

4.4 Модуль «mesh»

```
1   /* Данный заголовочный файл содержит объявления (прототипы) функций,
2   используемых для построения дискретного аналога расчётной области.
3   В данной версии этот модуль строит
4   только равномерные сетки по пространству и времени. */
5
6   #ifndef __MESH_HPP__
7   #define __MESH_HPP__
8   #include "my_types.hpp"
9
10  //Проверяет корректность динамического выделения памяти.
11  void checkMemAlloc(void *pointer, const char *ErrMsg);
12
13  //Заполняет массив с координатами граней контрольных объёмов.
14  //Грани расположены с одинак. уд. друг от друга на всей области.
15  double * getUniformFaces(double len, uns Nv);
16
17  //Заполняет массив с координатами контрольных точек.
18  //Каждая точка располагается в центре контрольного объёма.
19  double * getUniformNodes(double *xf, double len, uns Np);
20
21  //Заполняет массив со значениями временных слоёв.
22  double * getArray_cur_t(double *dt, uns Nt);
23
24  //Формирует равномерную расчётную сетку.
25  //Заполняются массивы: граней, узловых точек, временных слоёв,
26  //расстояний между гранями, расстояний между узловыми точками.
27  //Вычисляются раномерные шаги по времени.
28  void getUniformMesh(discrMesh &mesh);
29
30  //Функция для освобождения выделенных участков памяти.
31  void freeMesh(discrMesh &mesh);
32  #endif
```

```
1   /* В данном файле выполнены реализации функций,
2   необходимых для формирования одномерного дискретного
```

```
3  аналога фазового пространства - расчётной сетки.
4  В такой реализации сетка равномерная.
5  (Как по координатам, так и по времени.)
6
7  Все массивы относящиеся к пространству имеют одинаковую длину,
8  которая равна Np - общему числу узловых точек.
9  В массивах:
10 Faces, FacesIntervals, dFaces, dNodes - фактических значений меньше
11 Поэтому неиспользуемым ячейкам присваивается значение NAN. */
12
13 #include "mesh.hpp"
14 #include "my_types.hpp"
15 #include <cstdio>
16 #include <cstdlib>
17 #include <cmath>
18
19 //Заполняет массив с координатами граней контрольных объёмов.
20 //Первое значение не используется
21 double * getUniformFaces(double len, uns Nv) {
22   double *xf = new double [Nv+2];
23   checkMemAlloc(xf, "Error in function - getUniform_xf\n");
24   double dx = len / Nv;
25   xf[0] = NAN;
26   xf[Nv+1] = len;
27   for(uns i = 1; i <= Nv; i++)
28     xf[i] = (i-1)*dx;
29   xf[Nv+1] = len;
30   return xf;
31 }
32
33
34 //Массив узловых точек
35 //Узел помещается в посередине между гранями
36 double * getUniformNodes(double *xf, double len, unsigned Np) {
37   double *x = new double [Np];
38   checkMemAlloc(xf, "Error in function - getUniform_x\n");
39   x[0] = 0.0f; x[Np-1] = len;
40   for(uns i = 1; i <= Np-2; i++)
41     x[i] = (xf[i+1] + xf[i]) / 2.0f;
42   return x;
43 }
44
45 //Создаёт равномерный массив временных интервалов
46 double * getUniformArray_dt(double T_end, uns Nt) {
47   double *dt = new double [Nt+1];
48   checkMemAlloc(dt, "Error in function getUniform_dt\n");
49   double UnsformTimeStep = T_end / Nt;
50   for(uns i = 0; i <= Nt; i++)
51     dt[i] = UnsformTimeStep;
52   return dt;
53 }
54
```

```
55  //Формирует массив, хранящий значения времени
56  double * getArray_cur_t(double T_end, uns Nt) {
57    double *cur_t = new double [Nt+1];
58    checkMemAlloc(cur_t, "Error in function get_cur_t\n");
59    double h = T_end / Nt;
60    for(uns i = 0; i <= Nt; i++)
61      cur_t[i] = i*h;
62    return cur_t;
63  }
64
65  //В этой функции заполняются все массивы, которые
66  //представляют рассчётную сетку
67  void getUniformMesh(discrMesh &mesh) {
68    mesh.Faces = getUniformFaces(mesh.len, mesh.Nv);
69    mesh.Nodes = getUniformNodes(mesh.Faces, mesh.len, mesh.Np);;
70    mesh.dt = mesh.T_end / mesh.Nt;
71    mesh.cur_t = getArray_cur_t(mesh.T_end, mesh.Nt);
72  }
73
74  void freeMesh(discrMesh &mesh) {
75    delete [] mesh.Faces;
76    delete [] mesh.Nodes;
77    delete [] mesh.cur_t;
78  }
```

4.5 Модуль «solver»

Пожалуй самый «интересный» из всех модулей. Собственно в нём реализован метод решения системы линейных уравнений, решение которой даёт значения температуры. В данной программе вычислительная часть настроена на решение задачи с нулевыми граничными условиями первого рода. Но можно изменить несколько строк когда, где вычисляется прогоночные коэффициенты и получится, что программа способна решать и *ненулевые* первые начально-краевые задачи.Ниже приведён код заголовочного файла модуля «solver».

```
1   /* В этом модуле объявляны функции,
2   выполняющие вычислительную часть программы. */
3   #ifndef __CV_SOLVER__
4   #define __CV_SOLVER__
5   #include "my_types.hpp"
6   void checkMemAlloc(void *pointer, const char *ErrMsg);
7
8   //Вычисляет сеточные значения точного решения.
9   double **getExactSolution(const discrMesh &mesh);
10
11  //Реализация алгоритма ТДМА
12  double *TDMA(double *T0,const discrMesh &mesh,const physConsts &consts);
13  #endif
```

Далее идёт листинг с исходным файлом данного модуля. Нужно сказать, что при необходимости максимально ускорить вычисления данную версию решателя можно переделать. Сейчас при вычислениях в нескольких местах локальным переменным

переприсваиваются те же самые данные, что и в структуре mesh. Это сделано для того, чтобы упростить восприятие кода.

```
1  /* Данный модуль отвечает за вычислительную часть программы.
2  В нём реализован алгоритм трёдиагональной матрицы для случая
3  однородной задачи Дирихле, когда известно аналитическое выражение
4  стационарного источника. */
5
6  #include <cstdlib>
7  #include <cstdio>
8  #include "solver.hpp"
9  #include "my_types.hpp"
10 #include <cmath>
11
12 void checkMemAlloc(void *pointer, const char *ErrMsg);
13
14 double **getExactSolution(const discrMesh &mesh) {
15 //Массив хранит знач. аналитич. решения во всех рассчётных точках
16   double ** T_exact = new double *[mesh.Nt+1];
17   checkMemAlloc(T_exact, "Erorr in function T_exact\n");
18   for(uns t = 0; t <= mesh.Nt; t++) {
19     T_exact[t] = new double [mesh.Np];
20     checkMemAlloc(T_exact[t], "Erorr in function T_exact\n");
21     T_exact[t][0] = 0.0f; //Однородные граничные условия
22     T_exact[t][mesh.Np-1] = 0.0f; //Однородные граничные условия
23     for(uns i = 1; i <= mesh.Np-2; i++)
24       T_exact[t][i] = (1.0f-exp(-mesh.cur_t[t]))*sin(mesh.Nodes[i]);
25   }
26   return T_exact;
27 }
28
29 double *TDMA(double *T0,const discrMesh &mesh,const physConsts &consts)
30 {
31   //mesh - Структура, где хранятся все данные расчётной сетки
32   //consts - Структура, где хранятся физические константы ro, c, k
33   //t - Значение очередного времееного слоя
34   double dxW; //Расстояние между узловыми точками (P-W)
35   double dxE; //Расстояние между узловыми точками (E-P)
36   double dFaces; //Размер очередного контр. объема (e-w)
37   double fW, fE; //Координаты западной и восточной граней
38   double aW, aP, aE, bP; //Коэф. уравнений на кажд. врем. слое
39   //Прямой ход метода ТДМА заполняет массивы P, Q.
40   //На этапе обратного хода с их помощью вычисляется температура
41   //в каждой узловой точке
42   double *P, *Q;
43   double *T;
44   T = new double [mesh.Np];
45   P = new double [mesh.Np];
46   Q = new double [mesh.Np];
47   if(!(P && Q && T)) {
48     fprintf(stderr, "%s\n", "Mem Error!\n In function \'TDMA\'\n");
49     exit(EXIT_FAILURE);
50   }
```

```
51
52      //Расстояние между узловыми точками на равномерной сетке одинаково
53      //В данном случае можно вычислить коэффициенты один раз.
54      dxW = dxE = mesh.Nodes[1] - mesh.Nodes[0];
55      aW = aE = (0.5f*consts.k) / dxW;
56      //Прямой ход
57      T[0] = P[0] = Q[0] = 0.0f; //Граничное условие первого рода
58      for(uns i = 1; i <= mesh.Np-2; i++) {
59        //Вычисляем P[i], Q[i] для внутренних объёмов
60        //dFaces - Расст. между гранями очередного контрольного объема
61        dFaces = mesh.Faces[i+1] - mesh.Faces[i];
62        aP = (consts.ro*consts.c*dFaces)/mesh.dt + aW + aE;
63        fW = mesh.Faces[i]; //Западная грань
64        fE = mesh.Faces[i+1]; //Восточная грань
65        //bP - Вклад из левой части
66        bP = (consts.ro*consts.c*dFaces*T0[i])/mesh.dt;
67
68    //Вклад после интегрирования конвективного члена
69    //При интегрировании по времени исп. - схема Кранка-Николсона
70        bP += aE*(T0[i+1] - T0[i]) - aW*(T0[i] - T0[i-1]);
71        bP += cos(fW) - cos(fE); //Вклад стационарного источника
72        P[i] = aE / (aP - aW*P[i-1]); //Заполнение массивов P, Q
73        Q[i] = (aW*Q[i-1] + bP) / (aP - aW*P[i-1]);
74      }
75      //Задано однородное граничное условие первого рода
76      T[mesh.Np-1] = P[mesh.Np-1] = Q[mesh.Np-1] = 0.0f;
77
78      //Обратный ход (определение температуры во внутренних точках)
79      for(uns i = mesh.Np-2; i >= 1; i--)
80        T[i] = P[i]*T[i+1] + Q[i];
81
82      delete [] P; delete [] Q;
83      return T;
84    }
```

4.6 Модуль «calculate_errors»

```
1   /* В этом модуле записаны прототипы фукнций, используемых для
2   вычисления погрешностей.
3   Получается 4 вида информации о погрешности вычисления.
4   1)Абсолютная погрешность для каждого временного слоя.
5   2)Относительная погрешность для каждого временного слоя.
6   3)Значение максимальной абсолютной погрешности по временному слою.
7   4)Значение максимальной относительной погрешности по временному слою.
8   Эти величины записываются в специальные текстовые файлы.
9   1,2 - в файл errors, 3,4 - в файл MaxErrors. */
10
11  #ifndef __CALCULATE_ERRORS__
12  #define __CALCULATE_ERRORS__
13  #include "my_types.hpp"
14  #include <cmath>
15  void checkMemAlloc(void *pointer, const char *ErrMsg);
```

```
16
17  //Абсолютная погрешность для каждого временного слоя.
18  double * getAbsErr(double *curT_exact, double *curT, uns Np);
19
20  //Значение максимальной абсолютной погрешности по временному слою.
21  double getMaxAbsErr(double *curAbsErr, uns Np);
22
23  //Относительная погрешность для каждого временного слоя.
24  //Погр. выч. с исп. усреднённого значения точного решения.
25  double *getRelErr(double *curT_exact, double *curT,
26                    const discrMesh &mesh, uns time);
27
28  //Знач. максимальной относительной погрешности по временному слою.
29  double getMaxRelErr(double *curRelErr, uns Np);
30  #endif
```

```
1   /* В этом модуле реализованы функции для определения погрешностей */
2   #ifndef _USE_MATH_DEFINES
3   #define _USE_MATH_DEFINES
4   #endif
5   #include <cmath>
6   #include "my_types.hpp"
7   void checkMemAlloc(void *pointer, const char *ErrMsg);
8
9   //Определение абсолютной погрешности
10  //(применяется для каждого временного слоя)
11  double *getAbsErr(double *curT_exact, double *curT, uns Np) {
12    double *curAbsErr = new double [Np];
13    checkMemAlloc(curAbsErr, "Mem Error!\n In func \'getAbsErr\'\n");
14    for(uns i = 0; i < Np; i++)
15      curAbsErr[i] = fabs(curT_exact[i] - curT[i]);
16    return curAbsErr;
17  }
18
19  //Определение максимальной абсолютной погрешности.
20  //Получается по одному значению с каждого временного слоя.
21  double getMaxAbsErr(double *curAbsErr, uns Np) {
22    double maxAbsErr = curAbsErr[0];
23    for(uns i = 1; i < Np; i++)
24      if(curAbsErr[i] > maxAbsErr)
25        maxAbsErr = curAbsErr[i];
26    return maxAbsErr;
27  }
28
29  //Определение относительной погрешности.
30  //Применяется для каждого временного слоя.
31  double *getRelErr(double *curT_exact,
32                    double *curT, const discrMesh &mesh, uns time) {
33    //Получение усреднённого значения точного решения
34    double Texact_Avg=-cos(mesh.Nodes[mesh.Np-1])+cos(mesh.Nodes[0]);
35    Texact_Avg = (1.0f - exp(-mesh.cur_t[time])) / mesh.len;
36
37    double *curRelErr = new double [mesh.Np];
```

```
38    checkMemAlloc(curRelErr, "Mem Error!\n In func \'getRelErr\'\n");
39    for(uns i = 0; i < mesh.Np; i++)
40      curRelErr[i] = 100.0f*(fabs(curT_exact[i]-curT[i])/Texact_Avg);
41    return curRelErr;
42  }
43
44  //Определение максимальной относительной погрешности.
45  //Получается по одному значению с каждого временного слоя.
46  double getMaxRelErr(double *curRelErr, uns Np) {
47    double maxRel = curRelErr[0];
48    for(uns i = 1; i < Np; i++)
49      if(curRelErr[i] > maxRel)
50        maxRel = curRelErr[i];
51    return maxRel;
52  }
```

4.7 Модуль «main»

Главная функция «main» вызывает большинство процедур из описанных выше модулей. В ней подготавливается директория для сохранения файлов с данными расчётов и инициализируются нужные структуры данных. Потом запускается расчётный цикл, в котором вызывается процедура «TDMA»[4], итерации идут для каждого временного слоя. На каждой итерации происходит расчёт температуры в узловых точках на текущем временном слое, вывод расчётных значений и погрешностей в соответствующие файлы. Ниже приведён исходный код.

```
1   #define _USE_MATH_DEFINES //Для доступа к константе - числу Пи
2   #include <cmath>
3   #include <cstdlib>
4   #include <cstdio>
5   #include <unistd.h>
6   #include <sys/stat.h>
7
8   //Подключение пользовательских структур данных
9   #include "my_types.hpp" //mesh, consts, errors, сокр. для unsigned
10
11  //Функции для обработки входных параметров задачи
12  //и для вывода результатов в файлы
13  #include "IO_functions.hpp"
14
15  #include "mesh.hpp" //Процедуры для создания расчётной сетки
16
17  //Реализация алгоритма трёх. диаг. матрицы - ТДМА
18  //В данном случае для однород. задачи Дирихле со стац. источником
19  //И вспомогат. функ. заполн. сеточ. знач. эталонного решения
20  #include "solver.hpp"
21
22  #include "calculate_errors.hpp" //Функции для вычисления погрешностей
23
24  //Проверяет корректность динамического выделения памяти
```

[4]Код программы написан с учётом того, что его может посмотреть иностранный программист. Поэтому название выбрано от слов Tridiagonal matrix algorithm.

```
25  //В случае ошибки выводит "отладочное сообщение" ErrMsg
26  //И завершает работу программы
27  void checkMemAlloc(void *pointer, const char *ErrMsg) {
28    if(nullptr == pointer) {
29      fprintf(stderr, "%s\n", ErrMsg);
30      exit(EXIT_FAILURE);
31    }
32  }
33
34  //Создаёт в тек. директории каталог с именем, заданным польз.
35  //в качестве первого аргумента при запуске программы
36  //В созд. каталоге будут текст. файлы с результатами.
37  //Они содержат данные для визуализации, значения погрешностей.
38  void mkdirSafely(const char *newDir) {
39    if(mkdir(newDir, 0777) == -1) {
40      printf("Error with making directory \'%s\'\n", newDir);
41      exit(EXIT_FAILURE);
42    }
43  }
44
45  //Безопсный переход в директорию для записи результатов
46  void chdirSafely(const char *destDir) {
47    if(chdir(destDir) == -1) {
48      printf("Error with changing directory to \"%s\"\n", destDir);
49      exit(EXIT_FAILURE);
50    }
51  }
52
53  int main(const int argc, const char **argv) {
54    //Проверка правильного запуска программы
55    //Обязательные аргументы при запуске:
56    //имя директории, имя файла со входными данными задачи:
57    //Это длина, число объёмов, конеч. время, ro, c, k
58    //Именно в таком порядке они должны быть в конфигурац. файле.
59    if(argc != 3) {
60      printf("Error! Need two arguments: \n");
61      printf("1 --- Name of file with initial values\n");
62      printf("2 --- Name of directory to save results\n");
63      return 1;
64    }
65
66    discrMesh mesh; //Хранит все параметры сетки
67    physConsts consts; //Хранит физические константы (ro, c, k)
68    errors errs; //Хранит массивы с погрешностями
69
70    //Получение исходных данных для формирования сетки:
71    //длину, число контр об., кон. время, число врем. слоём
72    //В этом же файле задаются изические константы (ro, c, k)
73    getCaseParameters(argv[1], consts, mesh);
74
75    //Создание сетки с заданными параметрами:
76    //формируются массивы координат и временных слоёв.
```

```
77    getUniformMesh(mesh);
78
79    //Создание директории для вывода результатов расчёта
80    mkdirSafely(argv[2]);
81
82    //Переход в созданную директорию
83    chdirSafely(argv[2]);
84
85  //T_exact - набор знач. аналит. реш. в узловых т. по всем врем. слоям
86  //T0 - массив темп. (известных) в узловых т. на тек. врем. слое
87  //T - массив темп. (неизвестных) в узловых т. на след. врем. слое
88  //Заполняем таблицу сеточных значений точного решения
89    double **T_exact = getExactSolution(mesh);
90    double *T0 = new double [mesh.Np];
91    double *T = new double [mesh.Np];
92    checkMemAlloc(T0, "Error with memory allocation \"(T0) main\"\n");
93    checkMemAlloc(T, "Error with memory allocation \"(T) main\"\n");
94    for(uns i = 0; i < mesh.Np; i++)
95      T0[i] = T_exact[0][i];
96
97    //Основной цикл состоит из двух этапов:
98    //1) Рассчёт температуры на каждом слое;
99    //2) Вывод данных и погрешностей в файлы
100   for(uns t = 1; t <= mesh.Nt; t++) {
101   //Вычисл. темп. на тек. врем. слое с помощью алгоритма TDMA
102     T = TDMA(T0, mesh, consts);
103
104     //Вычисление погрешностей для данного временного слоя
105     errs.abs = getAbsErr(T_exact[t], T, mesh.Np);
106     errs.maxAbs = getMaxAbsErr(errs.abs, mesh.Np);
107     errs.rel = getRelErr(T_exact[t], T, mesh, t);
108     errs.maxRel = getMaxRelErr(errs.rel, mesh.Np);
109
110     //Вывод значений темп. и погр. в соответсвующие файлы
111     printDataToFile(mesh, T_exact[t], T);
112     printErrorsOnTimeLayerToFile(mesh, errs, t);
113     printMaxErrorsToFile(mesh, errs, t);
114
115     //Обновление значений - переход к новому слою
116     for(uns i = 0; i < mesh.Np; i++)
117       T0[i] = T[i];
118     delete [] T; delete [] errs.abs; delete [] errs.rel;
119   }
120
121   //Очистка выделенной памяти
122   for(uns t = 0; t < mesh.Nt; t++)
123     delete [] T_exact[t];
124   delete [] T_exact;
125   delete [] T0;
126   freeMesh(mesh);
127   return 0;
128 }
```

4.8 Скрипт для визуализации результатов расчётов в gnuplot

Утилита gnuplot поддерживает два режима работы. Один из них это выполнение команд из заданного файл. Это и применяется для построения анимации по результатам расчётов. При необходимости можно изменить настройки анимации, например, изменив задержку между кадрами или цвет графика.

```
#Установка режима анимации со скоростью 2 кадра в сек.
set terminal gif animate delay 50

#Выбор имени выходного файла
set output "HeatTransAnim.gif"

set font "Times New Roman,12" #Шрифт для подписей
set xlabel "x" #Подпись оси Ox
set xrange [0:pi] #Диапозон изменения x
set ylabel "T(x)" #Подпись оси Oy
set yrange [0:1] #Диапозон изменеия y = T(x)
set grid
stats "data.dat" name "File" #Получение информации о файле с данными
#%set nokey #Отключение легенды
set title "Evalution of temperature" #Подпись к анимации

#Цикл для покадрового построения анимации
do for [i=1:int(File_blocks)] { \
plot "data.dat" \
using 1:2 index i with linespoints \
linewidth 2 \
linecolor "red" \
title sprintf("frame = %d", i) }
#Анимация получается следующим образом:
#Выбраются данные из первых двух столбцов файла.
#Первый столбец это коорд. узловой точки, а второй знач. темп.
#Выбранный тип линий такой, что два соседних значения температуры
#в узловых точках соединяются прямой.
#Толщина линии равна 2 единицам.
#Цвет линии - красный.
#Опция title выводит номер кадра

```

4.9 Результаты тестирования

Таблица 4.1:
Число контрольных объёмов - 5.
Число временных слоёв - 5.

погрешность		
время сек.	абс.	относ. в %
0.200	0.001898	3.289
0.400	0.002584	2.463
0.600	0.002432	1.694
0.800	0.001720	0.981
1.000	0.000652	0.324

Таблица 4.2:
Число контрольных объёмов - 10.
Число временных слоёв - 10.

погрешность		
время сек.	абс.	относ. в %
0.100	0.000275	0.908
0.200	0.000462	0.800
0.300	0.000573	0.695
0.400	0.000623	0.594
0.500	0.000622	0.497
0.600	0.000579	0.403
0.700	0.000501	0.313
0.800	0.000397	0.226
0.900	0.000271	0.144
1.000	0.000129	0.064

Таблица 4.3:
Число контрольных объёмов - 20.
Число временных слоёв - 20.

погрешность		
время сек.	абс.	относ. в %
0.050	0.000038	0.243
0.100	0.000069	0.229
0.150	0.000095	0.215
0.200	0.000116	0.201
0.250	0.000132	0.188
0.300	0.000144	0.175
0.350	0.000152	0.162
0.400	0.000156	0.149
0.450	0.000158	0.137
0.500	0.000156	0.124
0.550	0.000151	0.112
0.600	0.000145	0.101
0.650	0.000136	0.089
0.700	0.000125	0.078
0.750	0.000112	0.067
0.800	0.000098	0.056
0.850	0.000083	0.046
0.900	0.000066	0.035
0.950	0.000049	0.025
1.000	0.000030	0.015

Когда производилось интегрирование уравнения (на стр.13) было сказано, что существует несколько подходов при интегрировании по времени. Была выбрана схема Кранка-Николсона. Согласно теоретическим рассуждениям, при её использовании должен быть *второй* порядок точности. Это означает, что при *удвоении* числа расчётных точек ошибка должна уменьшаться в *четыре* раза. Выше приведены таблицы. Один из столбцов содержит абсолютные погрешности, в а другой относительные. В этих таблицах в записаны *наибольшие* значения соответствующих погрешностей среди *всех узловых точек* для каждого временного слоя, значение времени указано в первом столбце. Ошибки вычислялись при сравнении температуры, посчитанной программой с точным значением, которое определяется решением задачи на (стр. 8).

Нужно уточнить, что для вычисления абсолютной погрешности использовалась стандартная формула: $\Delta T = |T^* - T|$, а для относительной погрешности δT значение в знаменателе было заменено на *среднее* по всему временному слою. Среднее значение $\overline{T}$ определялось интегрированием: $\overline{T} = \frac{1}{\pi}\int\limits_0^{pi} T(x,t_0)dx$. И итоговая формула для относительной погрешности, которая реализована в коде такая: $\frac{|T-T^*|}{\overline{T}}$

Результаты, представленные в таблицах подтверждают хорошую точность приближенного вычисления температуры на основании численного метода - метода контрольных объёмов. Кроме того, видно, что соблюдается выполнение второго порядка точности. Таким образом, при реализации процедуры разбиения расчётной области и вычислительного алгоритма ошибок нет.

Глава 5

Заключение

В ходе работы была проведена теоретическая подготовка. Изучена математическая модель описываемого процесса теплопроводности, а также численный метод решения задач из этой области — метод контрольных объёмов. В ходе работы удалось реализовать этот метод решения в виде расчётной программы, состоящей из нескольких частей. Результаты тестирования подтвердили теоретический прогноз о том, что схема Кранка-Николсона даёт второй порядок точности. В результате появился первый опыт компьютерного моделирования. Также удалось развить навыки программирования, так как в при создании программы в этом проекте активно использовалась раздельная сборка программы, работа с файлами, взаимодействие нескольких программ. Во время работы на программой часто помогала информация из [4]. Для разработки программы был выбран модульный подход. Это позволит с лёгкостью модернизировать начальную версию. Выходные данные создаваемые в процессе вычисления программой сделаны в формате, который поддерживается свободно распространяемой программой gnuplot, это позволяет визуализировать результаты расчётов. Модель взаимодействия конечного пользователя с программой, подробно описанная на (стр. 20) подходит к использованию даже тем людям, кто не владеет навыками программирования.

Литература

[1] Лекции по математической физике. — Издательство Московского Университета, 1993.

[2] Основы математического анализа часть 2. — Физматлит, 2015.

[3] Численные методы решения задач теплообмена и динамики жидкост. — Энергоатомиздат, 1984.

[4] Программированиие: введение в профессию. — ДМК Пресс, 2021.