МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФИЛИАЛ ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
В Г. НАБЕРЕЖНЫЕ ЧЕЛНЫ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Специальность: 010501.65 — Прикладная математика и информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(Дипломная работа)

Численное моделирование плоского взрыва закруглённого заряда эллиптической формы с использованием технологии многопоточности

Регистрационный номер

Работа завершена:
«____» __________ 2011 г. __________ М. А. Сафронов, гр. 4606

Работа допущена к защите:
Научный руководитель
д. ф.-м. н., профессор
«____» __________ 2011 г. __________ Л. М. Котляр

Зав. кафедрой
д. ф-м. н., профессор
«____» __________ 2011 г. __________ Р. Х. Латыпов

Набережные Челны — 2011 год

Содержание

Реферат

Сафронов М. А. ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ ПЛОСКОГО ВЗРЫВА ЗАГЛУБЛЁННОГО ЗАРЯДА ЭЛЛИПТИЧЕСКОЙ ФОРМЫ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ МНОГОПОТОЧНОСТИ, дипломная работа: стр. 41, рис. 8, табл. 1, библ. назв. 27.

Ключевые слова: МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ, КРАЕВАЯ ЗАДАЧА, ТЕОРИЯ СТРУЙ, ТВЁРДО-ЖИДКОСТНАЯ МОДЕЛЬ, ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ, HASKELL.

Рассматривается плоский взрыв заглублённого заряда эллиптической формы в импульсной постановке. Решение находится с использованием твёрдо-жидкостной модели Лаврентьева модифицированным методом Чаплыгина.

Разработана программа, численно рассчитывающая координаты точек границы воронки взрыва заглублённого эллиптического заряда, использующая возможности параллельных вычислений. Показана эффективность программирования в стиле, предусматривающем возможность выполнения программы в многоядерной среде, в том числе многопроцессорной.

Условные обозначения и сокращения

ПО	—	програмное обеспечение;
ТЖМ	—	твёрдо-жидкостная модель;
GHC	—	Glasgow Haskell Compiler, название традиционного известнейшего компилятора Haskell;
ОС	—	операционная система;
MB	—	мегабайт;
$\chi(u)$	—	функция Жуковского;
ϕ_0	—	значение потенциала течения на поверхности заряда;
v_0	—	критическое значение скорости течения на свободной поверхности;
τ	—	параметр, требуемый для полного задания тета-функций;
α	—	угол в точках A и D области 2.1 течения;
$\varkappa$	—	функция кривизны на границе заряда;
c_n	—	постоянные множители из разложения корректирующей функции (2.23) $f(u)$ в ряд Лорана, $n \in [0; \infty)$;
n_{theta}	—	число слагаемых в рядах (3.1) и (3.4), позволяющих вычислить значение тета-функций численно;
$n_{integral}$	—	частота разбиения отрезка интегрирования на сетку;
n_{cn}	—	количество постоянных множителей c_n в разложении $f(u)$, а, следовательно, и количество слагаемых в нём;
ε	—	желаемая точность при определении коэффициентов c_n численными методами.

Введение

В последнее время при производстве различного рода работ всё более широкое применение находят технологии, основанные на использовании энергии взрыва. Взрыв представляет собой сложную цепь физико-химических процессов, включающую в себя детонацию взрывчатого вещества, распространение ударных волн, фазовые превращения, пластическое течение среды, деформацию и разрушение. Для полного всестороннего изучения взрыва в сплошной среде применяются такие теории, как теория детонации, газовой динамики, твёрдого деформируемого тела, баллистики и многие другие.

Одной из основных задач, возникающих при использовании взрывов на выброс, является задача определения размеров и формы воронок (при взрыве сосредоточенных зарядов) или выемок (при взрыве шнуровых зарядов) выброса в зависимости от геометрии области, свойств грунта, энергетической характеристики заряда, а также его формы и расположения. Важность этой задачи обусловлена широким применением удлинённых линейно-распределённых или шнуровых зарядов при строительстве каналов, дамб и т. п.

Существуют различные подходы к исследованию этой задачи. Один из них состоит в обобщении опыта применения взрыва и экспериментальных данных и в построении на этой основе эмпирических формул (см., напр., [15]). Получающиеся при этом формулы довольно просты и удобны в использовании. Однако применимость их ограничена, т. к. они в основном позволяют рассчитывать взрывы на выброс в однородном полупространстве.

Другой подход заключается в создании математических моделей, описывающих основные процессы, происходящие при взрыве. Ряд авторов (см., напр., [1], [4]) при построении таких моделей стремятся учесть, по возможности, все стороны явления. При этом появляется возможность изучения влияния отдельных факторов на действие взрыва, но исследование взрыва сводится к решению сложных математических задач, требующих привлечения самых современных ЭВМ. Вследствие этого применимость таких моделей также ограничена, и используют их лишь для расчёта единичных взрывов, в основном, ядерных.

Из сказанного ясно, что побуждало многих исследователей к созданию различных моделей взрыва в грунте. При этом стремились, чтобы модель позволяла достаточно полно изучать основные явления взрыва и была сравнительно проста. По-видимому, наиболее теоретически обоснованной моделью, позволяющей в ряде случаев находить форму выемки выброса при взрыве шнурового заряда, является твёрдо-жидкостная модель (ТЖМ) взрыва на выброс, предложенная М. А. Лаврентьевым [11] и применённая впервые к решению конкретных задач В. М. Кузнецовым [9], [10]. Согласно этой модели, основанной на импульсно-гидродинамической постановке [2], граница выемки выброса находится как линия тока, вдоль которой скорость постоянна и равна критической, характеризующей среду.

За последние 20 лет, прошедших с момента опубликования монографии [6], судя по всему, не было выпущено больше ни одной работы, в которой применялась бы ТЖМ как средство решения прикладных задач использования взрывов на выброс.

Задача, рассмотренная в данной работе, является обобщением задачи, решённой в других постановках Котляром Л. М. [8], Ильинским Н. Б. [6] и Мартынюком П. А. [12]. В работе [8] рассматривается криволинейный незаглублённый заряд (расположенный на поверхности грунта), в работе [6] заряд полигональной формы, а [12] сводит взрыв кругового заряда к взрыву точечного заряда, теряя общность.

Постановка задачи. *Требуется найти геометрическое место точек, расположенных на границе воронки взрыва заряда эллиптической формы, заглублённого в грунт. Для простоты взрыв рассматривается на двухмерной плоскости, что позволяет экстраполировать результат двояко: как взрыв заряда в форме эллипсоида вращения, или как взрыв шнурового заряда эллиптического сечения.*

Работа будет состоять из двух этапов. На первом этапе мы построим чисто математическую модель взрыва, состоящую из одной функции, которая позволяет найти искомую границу воронки взрыва исходя из его физических параметров. На втором этапе математическая модель будет транслирована в программный код, позволяющий исследовать модель средствами персонального компьютера.

В последние годы (2006–2010 и далее) наблюдается значительный тренд на использование многопроцессорных (в настольных системах — многоядерных) технологий в компьютерной индустрии [25], обусловленный технологическими трудностями, препятствующими развитию традиционных однопроцессорных/одноядерных решений. Многопроцессорный подход предполагает разработку программного обеспечения, явно предусматривающего его возможное выполнение в среде, способной выполнять некоторые вычисления параллельно. Для разработки такого рода «параллельного» ПО требуется использовать вполне определённые методики программирования, однако, преимущества в виде многократного увеличения производительности, полученные при их использовании, перевешивают сложности разработки.

Таким образом, **дополнительные требования** к решению задачи следующие. *Запрограммировать вычисления полученной на первом* *этапе математической модели таким образом, чтобы в наиболее возможно полной мере воспользоваться возможностями многопроцессорной/многоядерной архитектуры современной вычислительной техники.*

Целью настоящей работы является:

1 решить задачу, являющуюся обобщением более ранних работ в данной области;

2 предоставить средство расчётов реальных границ воронок от взрывов, масштабируемое и пригодное для выполнения на высокопроизводительной вычислительной технике, эксплуатирующей параллельность вычислений.

Глава 1. Аналитический обзор

1.1 Обзор исследований по математическому моделированию взрывов

Исследованию механизма взрыва в грунтах в середине XX века посвящено много работ. Интерес к этой теме был вызван как практическими, так и теоретическими соображениями. Сложность явления, большой диапазон изменений параметров (напряжений, скоростей и др.), разнообразие условий и назначения взрыва представляют благоприятную и увлекательную тему для исследователя, для его изобретательности и интуиции. Большое значение при этом имеет и практическое использование взрывов в грунтах и горных породах. Взрывы применяются при разработке полезных ископаемых, при проходке траншей, при сооружении плотин. Например, большая плотина в Медео (близ Алма-Аты) и гидротехнический комплекс в Нуреке (Таджикистан) были сооружены с использованием энергии взрыва.

После работ О. Е. Власова, применившего импульсно-гидродинамическую постановку (ИГП) к изучению взрыва, и пионерских работ М. А. Лаврентьева и В. М. Кузнецова, поставивших и решивших ряд краевых задач теории взрыва, появилось большое количество статей, посвящённых исследованию задач взрыва в ИГП (к 1990 г. их библиография насчитывала свыше 150 наименований; большая их часть была выполнена в Новосибирске, Киеве, Казани и Москве). Из монографий, в которых рассматриваются задачи взрыва в ИГП, ближе всего к теме настоящей работы книга В. М. Кузнецова [10] и книга Ильинского [6].

Использующаяся в данной работе твёрдо-жидкостная модель сильно схематизирует явление, на зато она приводит его к сравнительно простому математическому описанию. Основное упрощение в импульсной модели состоит в том, что среда предполагается несжимаемой, в результате чего исчезает из поля зрения волновой процесс и математически задача сводится к краевой задаче для эллиптических уравнений (в случае несжимаемой среды — к задаче для уравнения Лапласа). При помощи такой модели, конечно, нельзя ответить на все вопросы; однако на некоторые (не только качественные, но и количественные) ответ получается с достаточной степенью приближения.

В данной работе решается самая очевидная задача, которая может быть решена при помощи ТЖМ — задача определения границы воронки выброса. В импульсной модели остался недостаточно обоснованным выбор критерия, определяющего эту границу. Причина возникающей при этом трудности понятна — здесь ставится вопрос о критерии прочности, критерии разрушения, т. е. возникает новый физический вопрос, который не может быть решён простым моделированием среды несжимаемой жидкостью. Обычным решением данной проблемы является выбор границы воронки выброса как линии тока, на которой функция тока принимает определённое «критическое» значение, зависящее от параметров среды.

Метод решения поставленной задачи основан на методе, успешно применённом в работе Л. М. Котляра [8] для незаглублённого заряда. Отличие от работы [8] в заглублении заряда и приданию ему эллиптической формы, т. е., фактически, в переходе к более обобщённой задаче. Метод Л. М. Котляра основывается на некоторых модификациях метода Чаплыгина [3] и использовании тета-функций [16] для построения искомых функций.

Работа [6], по-видимому, была последней из опубликованных по теме использования ТЖМ во взрывном деле. Никаких работ с 1986 года больше не издавалось, поэтому, данная работа приобретает особую актуальность как возобновляющая прерванные по тем или иным причинам исследования.

В настоящее время (начало 2011 года) из общедоступных и относительно известных средств моделирования взрывов являются специализированные программные пакеты для персонального компьютера, такие, как LS-DYNA [13]. Расчёт в этих программах ведётся при помощи различных сеточных методов, рассчитывающих взрыв как определённую деформацию сплошной среды. Такой подход исключительно ресурсозатратный, особенно при необходимости соблюдать высокую точность расчётов, и расчёты по более простой математической модели, такой как ТЖМ, в некоторых случаях будет эффективнее, чем использование подобных программных пакетов.

1.2 Обзор технологии параллельных вычислений и современных техник её применения

Все современные микропроцессоры имеют два или больше ядер и относительно скоро можно ожидать, что количество ядер вырастет до десятков или сотен. Больше нельзя рассчитывать на то, что производительность каждого отдельного ядра будет расти и дальше. Единственный способ достичь увеличения производительности от каждого следующего поколения чипов — разделить работу программы на множество обрабатывающих данные ядер.

Разделить приложение на множество обрабатывающих ядер возможно, если каким-либо образом автоматически распараллелить последовательный код. Данный подход является текущей областью исследований в области компьютерных наук. Другой подход заключается в написании полу-явно или явно параллельных программ, которые затем будут распределены на множество ядер операционной системой и это именно тот тот подход, который будет использоваться в данной работе.

Следует разделить понятия «параллельные», «*parallel*», вычисления и «одновременные»[1], «*concurrent*» вычисления. Параллельная программа написана с конкретной целью воспользоваться потенциалом истинно параллельного вычислительного ресурса, такого, как многоядерного процессора. От параллельной программы мы ожидаем истинно одновременного выполнения. Одновременность же является техникой структурирования программного обеспечения, которая позволяет смоделировать вычисления в виде гипотетически независимых действий, которые могут синхронизироваться и связываться друг с другом.

Написание одновременных и параллельных программ — гораздо более сложная задача, чем написание последовательных программ. Однако, существуют серьёзные причины для написания одновременных и параллельных программ:

1 **Производительность**. Чтобы получить увеличение производительности с каждым новым поколением многоядерных процессоров, нам следует писать параллельные программы.

2 **Сокрытие задержек**. Даже на одноядерных процессорах мы можем использовать одновременные программы для того, чтобы скрыть от пользователя ожидание медленных операций ввода/вывода на диски и сетевые устройства.

3 **Структурирование ПО**. Некоторые виды задач могу быть удобным образом представлены в виде множества связанных друг с другом нитей выполнения, что помогает структурировать код более модульным образом. Например, можно моделировать компоненты пользовательского интерфейса как отдельные нити выполнения.

4 **Одновременность в реальности**. В распределённых системах и системах реального времени приходится моделировать и реагиро-

[1]Устоявшегося перевода термина *concurrent* на русский язык нет. Поэтому для ясности, так как разница в терминах *parallel* и *concurrent* крайне значительна, везде далее в тексте будет использовано выражение «одновременная программа» как перевод фразы «concurrent program»

вать на события в реальном мире, например, обрабатывать многочисленные запросы к серверу, параллельно.

Современные микропроцессоры используют так называемый thread-level parallelism, «параллелизм на уровне нитей выполнения» [25, стр. 195]. Это означает, что участки кода, которые должны выполняться параллельно (если это возможно), указываются явно или полу-явно разработчиком программного обеспечения. Будут ли эти участки *действительно* выполняться параллельно, зависит от компилятора программы и от операционной системы, которая будет её выполнять. В любом случае, для описания параллелизма выполнения программы используют понятие «нить выполнения» («thread»). Нить выполнения инкапсулирует вычисления, которые должны выполняться одновременно с другими нитями.

В настоящее время существует несколько высокоуровневых техник построения параллельных программ, например, MapReduce [19], Software Transactional Memory [26] или модель акторов [21].

В программе, которая будет реализована в рамках данного проекта, будет использована как истинная параллельность чисто математических вычислений, так и техника одновременности операций ввода/вывода. Вычисления и диалог с пользователем будут производиться одновременно, что позволит, например, в перспективе — запустить параллельно несколько вычислений из одного экземпляра программы. Для увеличения производительности же будет использован параллелизм выполнения некоторых вычислительных операций в программе.

1.3 Выбор языка программирования

Для программирования вычислений, с учётом требований, предъявленных к задаче, был выбран язык Haskell [24].

Haskell является чисто функциональным строго типизированным языком программирования с «ленивой» моделью вычислений. Он одновременно как интерпретируемый, так и компилируемый (доступны как интерпретатор Haskell, так и компилятор). Возможно заниматься отладкой программы в интерпретаторе, с последующей компиляцией исходного кода в нативное приложение, не требующее интерпретатора для выполнения.

Интерпретируемость программ на Haskell облегчает их отладку, написание и переносимость. Функциональная парадигма программирования, лежащая в основе этого языка, облегчает переформулирование математических выкладок, которые будут являться решением поставленной задачи, в программный код.

Более того, синтаксис и выразительность Haskell позволяют в некоторых случаях переписывать математические выражения в программный код «один-к-одному», без издержек на переформулирование в другие абстракции, что является обязательным при использовании императивных языков, таких, как C [27].

Интерпретатор и компилятор Haskell не поддерживают кросс-компиляцию, но сами скомпилированы для многих платформ, и являются приложениями с открытым исходным кодом, что позволяет переносить программы, написанные на Haskell, относительно простыми методами.

Для Haskell имеется обширная библиотека пакетов, включающая в себя пакет для черчения произвольных графиков функций Graphics.Rendering.Chart [20], а также пакет для поддержки истинно параллельных вычислений Control.Parallel [23] и пакет для поддержки одновременности выполнения Control.Concurrent [18]. Следует заметить, что зависимости, наличествующие для пакета Graphics.Rendering.Chart, значительно усложнят сборку приложения для ОС Windows, так как в этой операционной системе отсутствует стандартный удобный для использования из командной строки компилятор языка C.

Чистый параллелизм в Haskell сохраняет детерминизм операций, поэтому его включение в программный код не меняет результата.

С 2004 года традиционный компилятор Haskell, «GHC» поддерживает компиляцию программ, способных выполняться параллельно на многоядерных машинах. Для этого требуется компилировать с флагом `-threaded`, и запускать приложение с аргументами командной строки `+RTS -Nn`, где `n` — это количество ядер, на которые можно рассчитывать в текущей среде выполнения. Запущенное в таком режиме приложение будет пытаться распараллеливать выполнение участков кода, явно помеченных разработчиком как должные выполняться одновременно.

Также можно добиться повышения производительности приложения на языке Haskell, скомпилированного в параллельном режиме, явно указав размер кучи и стека, доступных для приложения. Увеличение кучи ведёт к уменьшению количества запусков сборщика мусора в случае требовательных к ресурсам вычислений, а каждый запуск сборщика мусора уменьшает производительность приложения.

Размер кучи указывается аргументом командной строки для приложения, например `-K100M` указывает использовать 100 MB для стека, а `-H800MB` указывает использовать 800 MB для кучи.

Глава 2. Формулирование картины взрыва согласно ТЖМ

Следуя М. А. Лаврентьеву, рассмотрим задачу теории взрыва в импульсной постановке. В этом случае течение, возникающее под действием импульсного давления, является потенциальным. Если пренебречь сжимаемостью среды, её прочностными и пластическими свойствами, а также силами трения и гравитации, то мы придём к изучению движения идеальной несжимаемой невесомой жидкости. Грунт, на который действует заряд, моделируется, таким образом, средой, движущейся как идеальная несжимаемая жидкость при скоростях больше некоторой критической скорости v_0 и является абсолютно твёрдым телом при скоростях, меньших v_0.

2.1 Постановка краевой задачи, эквивалентной исходной

Представим согласно твёрдо-жидкостной модели двумерный взрыв заглублённого заряда на двумерном евклидовом пространстве в декартовых координатах как плоское потенциальное установившееся течение идеальной невесомой жидкости на плоскости комплексного переменного $z = x + iy$. Ось ординат направлена по горизонтальной поверхности, ось абсцисс — линия симметрии и направлена вертикально вниз. В силу симметрии будем рассматривать только правую половину течения. Таким образом, рассматриваемая область течения ограничена криволинейным участком AD, прямолинейными участками AB и CD и линией тока CB (см. рис. 2.1).

Обозначим через $\pi\gamma/2$ угол, образованный касательной к заряду и горизонтальной поверхностью в точке D (рис. 2.1). Довольно очевидно, что для того, чтобы картина взрыва соответствовала рис. 2.1, должно выполняться условие $\gamma < 1$. Введём область G_u вспомогательного переменного $u = \xi + i\eta$ в виде прямоугольника $0 < \xi < \pi/4$, $0 < \eta < \pi|\tau|/4$ с соответствием точек, указанным на рис. 2.2.

Следует найти отображение $z(u)$ такое, что оно конформно отображает прямоугольник $ABCD$ из пространства u на область $ABCD$ из пространства z.

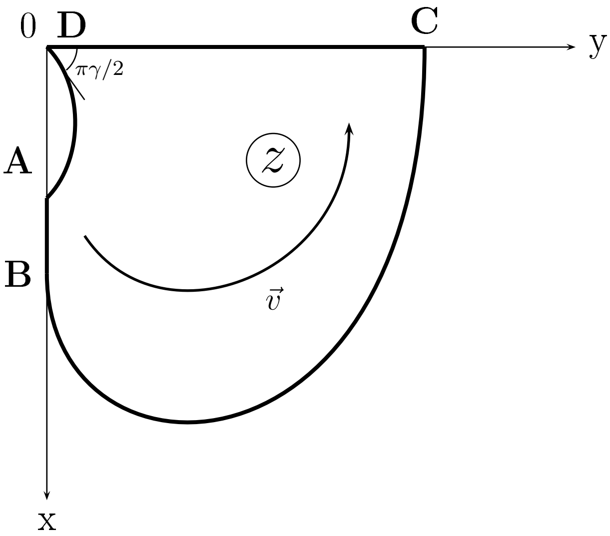

Рис. 2.1. Исходная схема модели

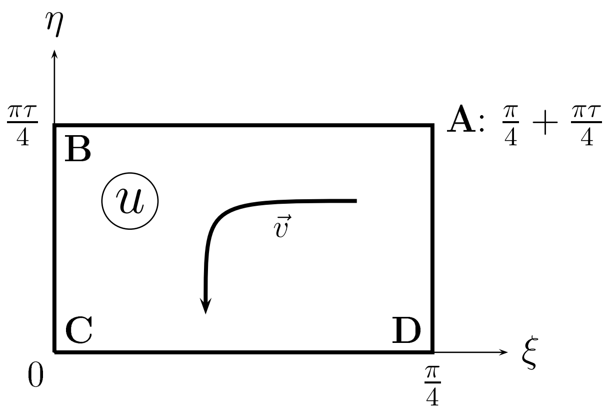

Рис. 2.2. Область дополнительного переменного u

Параметр τ задан так, чтобы $\operatorname{Im}\tau > 0$ и $\operatorname{Re}\tau = 0$.

2.2 Вывод решения для краевой задачи

Нам достаточно получить dz/du, чтобы найти $\forall z \colon z \in CB$. Нужная граница будет найдена интегрированием по u на соответствующей стороне прямоугольника.

Представим dz/du в виде производной сложной функции:

$$\frac{dz}{du} = \frac{dz}{dw} \cdot \frac{dw}{du} \tag{2.1}$$

Выведем последовательно оба множителя этого произведения.

2.2.1 Комплексный потенциал на плоскости вспомогательного переменного

Комплексный потенциал $w(u)$ над вспомогательным переменным u вводится следующим образом:

$$w(u) = \varphi(u) + i\psi(u) \tag{2.2}$$

Рассмотрим $w(u)$ как точку в поле $w = \varphi + i\psi$. Нам известны из постановки задачи краевые условия для $w(u)$:

$$CD : \varphi = 0 \tag{2.3a}$$

$$AD : \varphi = \varphi_0 \tag{2.3b}$$

$$ABC : \psi = -\psi_0 \tag{2.3c}$$

Производная комплексного потенциала dw/du имеет в области G_u нуль первого порядка в точке B (нарушается конформность отображения) и полюс первого порядка в точке D (вихрь интенсивности $4\varphi_0$). Как видно из рис. 2.1, dw/du чисто мнима на BCD и вещественна на BAD. Следовательно, её можно продолжить по принципу симметрии на всю плоскость. На основании теории эллиптических функций [16, стр. 350], найдём

$$\frac{dw}{du} = iN \frac{\vartheta_1\left(u - \frac{\pi\tau}{4}\right)\vartheta_1\left(u + \frac{\pi\tau}{4}\right)\vartheta_2\left(u - \frac{\pi\tau}{4}\right)\vartheta_2\left(u + \frac{\pi\tau}{4}\right)}{\vartheta_1\left(u - \frac{\pi}{4}\right)\vartheta_1\left(u + \frac{\pi}{4}\right)\vartheta_4\left(u - \frac{\pi}{4}\right)\vartheta_4\left(u + \frac{\pi}{4}\right)} \tag{2.4}$$

где N — вещественная положительная постоянная, $\vartheta_k(u), k = 1, 2, 3, 4$ — тэта-функции для периодов π и $\pi\tau$ [16, стр. 334].

Определяя из (2.4) вычет функции $w(u)$ в точке D, выразим постоянную N через величину φ_0:

$$N = \frac{\varphi_0 M}{\pi}, \qquad M = 2\left(\frac{\vartheta_2(0)\vartheta_3(0)\vartheta_4(0)}{\left|\vartheta_1\left(\frac{\pi}{4} + \frac{\pi\tau}{4}\right)\right|^2}\right)^2 \tag{2.5}$$

2.2.2 Выражение комплексного потенциала через функцию Жуковского

Для того, чтобы выразить dz/dw из (2.1), введём функцию Жуковского [3, с. 30]:

$$\chi(u) = \ln\frac{dw}{v_0 dz} = \ln\frac{v}{v_0} - i\Theta \equiv r - i\Theta \tag{2.6}$$

где Θ — угол наклона вектора скорости к оси Ox, v — модуль скорости в точке u, v_0 — скорость на свободной поверхности CD.

Из рис. 2.1 получим граничные условия для $\chi(u)$ на прямолинейных участках области течения:

$$AB : \operatorname{Im} \chi(u)\big|_{u=\xi+\frac{\pi\tau}{4}} = 0 \tag{2.7a}$$

$$CD : \operatorname{Im} \chi(u)\big|_{u=\xi} = -\pi \tag{2.7b}$$

$$CB : \operatorname{Re} \chi(u)\big|_{u=i\eta} = 0 \tag{2.7c}$$

Для того, чтобы получить краевое условие на AD, сделаем следующее.

Пусть на криволинейной дуге AD задан $\angle\beta$, образованный касательной с осью абсцисс, при этом $\beta = \beta(s)$, где s — длина дуги, отсчитываемая от точки A и отнесённая к полной длине l дуги AD. Тогда безразмерная кривизна дуги $\varkappa(\beta) = \frac{d\beta}{ds}$, и, так как $\beta = \Theta - \frac{\pi}{2}$, то

$$\varkappa(\Theta) = \frac{d\Theta}{ds} = \frac{d\Theta}{du} \cdot \frac{du}{ds} = \frac{d\Theta}{du} \cdot \left|\frac{du}{dz}\right| \tag{2.8}$$

Теперь, учитывая (2.4), (2.5) и (2.8) получим условие для $\chi(u)$ на AD:

$$\begin{aligned} \frac{d\Theta}{d\eta} &= \delta M \varkappa(\Theta) \left|\frac{dz}{dw} \cdot \frac{dw}{du}\right| = \\ &= \delta M \varkappa(\Theta) \left|F\left(\frac{\pi}{4} + i\eta\right)\right| e^{-r(\eta)} \end{aligned} \tag{2.9}$$

где $\delta = \frac{\varphi_0}{v_0 l \pi}$ — безразмерный параметр.

2.2.3 Разложение функции Жуковского на прямолинейную часть и корректирующую функцию

Представим $\chi(u)$ в следующем виде:

$$\chi(u) = \chi_0(u) - f(u) \tag{2.10}$$

$\chi_0(u)$ представляет собой функцию Жуковского для упрощённой задачи, изображённой на рис. 2.3, а функция $f(u)$ является специальной функцией, которая модифицирует упрощённую картину, внося в неё изменения, необходимые для возврата к исходной картине явления.

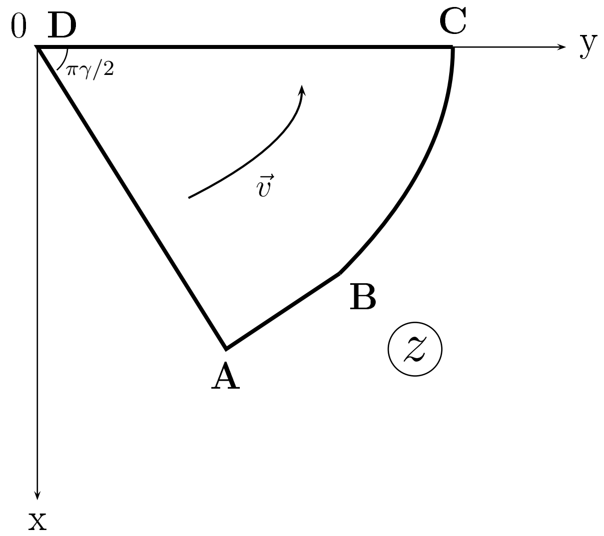

Рис. 2.3. Упрощённая схема задачи

Для χ_0 должны выполняться следующие граничные условия:

$$CD : \operatorname{Im}\chi_0(u)\big|_{u=\xi} = -\pi \tag{2.11a}$$

$$AD : \operatorname{Im}\chi_0(u)\big|_{u=\frac{\pi}{4}+i\eta} = -\pi\left(1-\frac{\gamma}{2}\right) \tag{2.11b}$$

$$AB : \operatorname{Im}\chi_0(u)\big|_{u=\xi+\frac{\pi\tau}{4}} = -\pi\left(1-\frac{\gamma}{2}\right) \tag{2.11c}$$

$$BC : \operatorname{Re}\chi_0(u)\big|_{u=i\eta} = 0 \tag{2.11d}$$

Следовательно, $\chi_0(u)$ имеет в D логарифмическую особенность с вычетом $-\gamma$.

Краевые условия для корректирующей функции $f(u)$ найдём позднее.

Функция Жуковского для упрощённой задачи

Так как

$$\frac{d\chi_0}{du} = \frac{\partial\operatorname{Re}\chi_0}{\partial\xi} + i\frac{\partial\operatorname{Im}\chi_0}{\partial\xi} = \frac{\partial\operatorname{Im}\chi_0}{\partial\eta} - i\frac{\partial\operatorname{Re}\chi_0}{\partial\eta} \tag{2.12}$$

то $\frac{d\chi_0}{du}$ чисто вещественна на CD и BA и чисто мнима на BC и AD. Значит, её можно продолжить на прямоугольник $\xi \in \left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$, $\eta \in \left[-3\frac{\pi\tau}{4}, \frac{\pi\tau}{4}\right]$ соответственно рис. 2.4.

Функцию χ_0 будем искать в виде комбинации тета-функций (см. [16, с. 334]). Так как полученный прямоугольник по размерам совпадает с периодичностью тета-функций, то такое определение χ_0 автоматически определит её на всей плоскости.

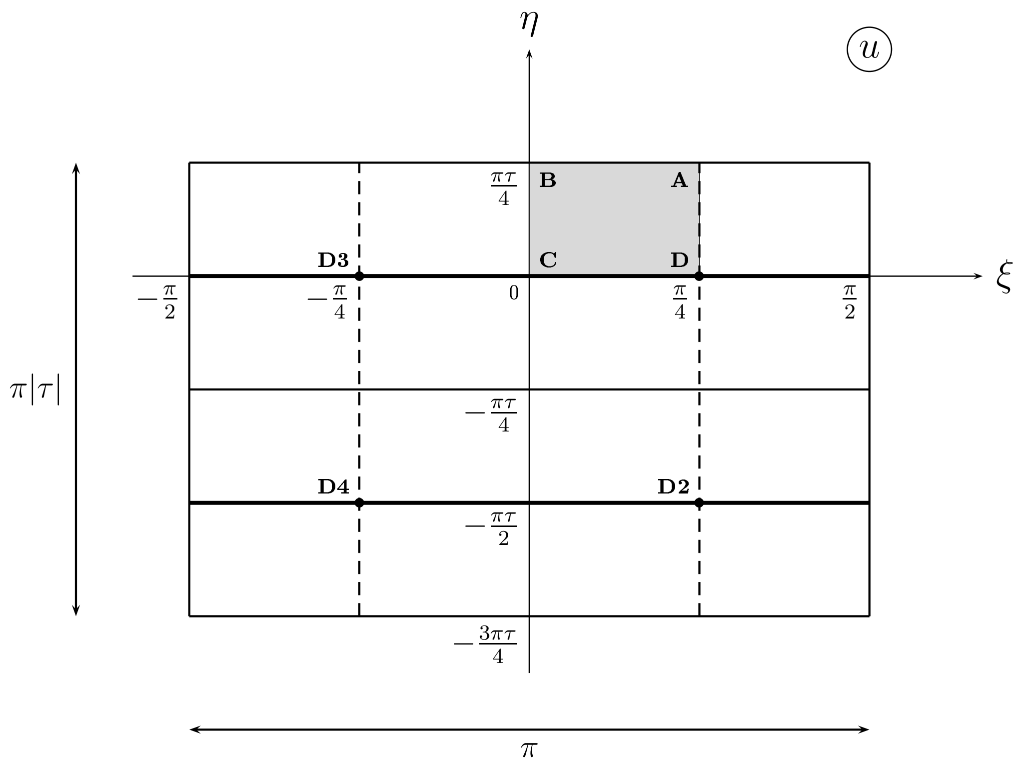

Рис. 2.4. Схема отображения χ_0 по принципу симметрии

Представим χ_0 в виде

$$\frac{d\chi_0}{du} = \sum_{k=1}^{4} A_k \frac{d}{du} \ln \vartheta_1(u - b_k) \tag{2.13}$$

где A_k — это вычеты функции χ_0 в её полюсах b_k, $k = 1, 2, 3, 4$.

Для точек D и D_2 вычет равняется $-\gamma$. Для точек D_3 и D_4 вычет равняется γ. Их сумма равна нулю, что позволяет нам использовать (2.13):

$$\begin{gathered}\frac{d\chi_0}{du} = \gamma \frac{d}{du}\Bigg(\ln \vartheta_1 \left(u + \frac{\pi}{4}\right) + \ln \vartheta_1 \left(u + \frac{\pi}{4} - \frac{\pi\tau}{2}\right) - \\ - \ln \vartheta_1 \left(u - \frac{\pi}{4}\right) + \ln \vartheta_1 \left(u - \frac{\pi}{4} - \frac{\pi\tau}{2}\right) \Bigg)\end{gathered} \tag{2.14}$$

Так как

$$\begin{aligned}&\vartheta_1 \left(u - \frac{\pi\tau}{2}\right) = N \vartheta_4(u) \\ &N = -ie^{iu + \frac{1}{4} i\pi\tau}\end{aligned}$$

(см. рис. 2.5) то

$$\frac{d\chi_0}{du} = \gamma \frac{d}{du} \ln \frac{\vartheta_1\left(u + \frac{\pi}{4}\right) \vartheta_4 \left(u + \frac{\pi}{4}\right)}{\vartheta_1\left(u - \frac{\pi}{4}\right) \vartheta_4 \left(u - \frac{\pi}{4}\right)} \tag{2.15}$$

Откуда

$$\chi_0(u) = C + \gamma \ln \frac{\vartheta_1\left(u + \frac{\pi}{4}\right) \vartheta_4 \left(u + \frac{\pi}{4}\right)}{\vartheta_1\left(u - \frac{\pi}{4}\right) \vartheta_4 \left(u - \frac{\pi}{4}\right)} \tag{2.16}$$

$$\begin{tikzcd}
\Theta_1(u) \arrow[r, "u+\frac{\pi}{2}"] & \Theta_2(u) \\
\Theta_4(u) \arrow[u, "u+\frac{\pi\tau}{2}"] \arrow[r, "u+\frac{\pi}{2}"] & \Theta_3(u)
\end{tikzcd}$$

Рис. 2.5. Соотношение между тета-функциями

Для определения константы интегрирования C воспользуемся краевыми условиями (2.11) для χ_0. Откуда получим конечное выражение для χ_0:

$$\chi_0(u) = -i\pi(\gamma - 1) + \gamma \ln \frac{\vartheta_1\left(u + \frac{\pi}{4}\right)\vartheta_4\left(u + \frac{\pi}{4}\right)}{\vartheta_1\left(u - \frac{\pi}{4}\right)\vartheta_4\left(u - \frac{\pi}{4}\right)} \tag{2.17}$$

Корректирующая функция $f(u)$

$f(u)$ — аналитическая функция, заключающая в себе отличия упрощённой схемы взрыва от данной изначально.

Граничные условия $f(u)$ найдём, сравнивая граничные условия (2.7) для $\chi(u)$ и (2.11) для $\chi_0(u)$:

$$AB : \operatorname{Im} f(u)\big|_{u=\xi+\frac{\pi\tau}{4}} = 0 \tag{2.18a}$$

$$CD : \operatorname{Im} f(u)\big|_{u=\xi} = 0 \tag{2.18b}$$

$$BC : \operatorname{Re} f(u)\big|_{u=i\eta} = 0 \tag{2.18c}$$

$$AD : \frac{d}{d\eta}\operatorname{Im} f(u)\big|_{u=\frac{\pi}{4}+i\eta} = \delta M\varkappa(\Theta)\left|F\left(\frac{\pi}{4} + i\eta\right)\right| e^{-r(\eta)} \tag{2.18d}$$

Для построения $f(u)$ воспользуемся тем, что некоторые функции можно разложить в ряд Лорана.

Отобразим область u на полукольцо $\rho \leqslant |\zeta| \leqslant 1$ с помощью следующей функции (см., напр., [17, с. 107]):

$$\zeta = e^{\frac{4u-\pi}{|\tau|}}, \qquad \rho = e^{-\frac{\pi}{|\tau|}} \tag{2.19}$$

Рассмотрим функцию

$$p(\zeta) = f(u(\zeta)) + \ln\zeta\left(1 - \frac{\gamma}{2}\right) \tag{2.20}$$

Эта функция удовлетворяет граничному условию

$$\operatorname{Im}\zeta \Rightarrow \operatorname{Im} p(\zeta) = 0 \tag{2.21}$$

Отсюда следует, что её можно продолжить на всё кольцо и на кольце можно представить в виде ряда Лорана

$$p(\zeta) = \sum_{n=-\infty}^{+\infty} c_n \zeta^n \tag{2.22}$$

А тогда

$$f(u) = -\frac{(4u-\pi)\left(1-\frac{\gamma}{2}\right)}{|\tau|} \sum_{n=-\infty}^{+\infty} c_n e^{\frac{4u-\pi}{|\tau|}} \tag{2.23}$$

Воспользуемся условием (2.18c) для того, чтобы найти некоторые из коэффициентов c_n:

$$\operatorname{Re} f(i\eta)\Big|_{\eta\in\left[0;\frac{\pi\tau}{4}\right]} = c_0 + \sum_{n=1}^{+\infty}\left(c_n e^{-\frac{\pi n}{|\tau|}} + c_{-n} e^{\frac{\pi n}{|\tau|}}\right)\cos\frac{4\eta n}{|\tau|} + \frac{\pi\left(1-\frac{\gamma}{2}\right)}{|\tau|} = 0 \tag{2.24}$$

Из (2.24) получим:

$$c_0 = -\frac{\pi}{|\tau|}\left(1-\frac{\gamma}{2}\right), \qquad c_{-n} = -c_n \rho^{2n}, \quad \rho = e^{-\frac{\pi}{|\tau|}} \tag{2.25}$$

Теперь воспользуемся условием (2.18d):

$$4\sum_{n=1}^{+\infty} C_n \frac{n}{|\tau|}\left(1-\rho^{2n}\right)\cos\frac{4n}{|\tau|}\eta - 4\frac{1-\frac{\gamma}{2}}{|\tau|} = \delta M \varkappa(\Theta) Q(\eta) e^{-\frac{\pi}{|\tau|}\left(1-\frac{\gamma}{2}\right)} \tag{2.26}$$

$$Q(\eta) = \frac{\left|\vartheta_2\left(\frac{\pi}{4}+i\eta-\frac{\pi\tau}{4}\right)\vartheta_2\left(\frac{\pi}{4}+i\eta+\frac{\pi\tau}{4}\right)\right|^2}{\left|\vartheta_2(i\eta)\vartheta_3(i\eta)\right|^{1+\gamma}\left|\vartheta_1(i\eta)\vartheta_4(i\eta)\right|^{1-\gamma}} e^{\sum\limits_{n=1}^{+\infty} c_n (1-\rho^{2n})\cos\frac{4n}{|\tau|}\eta} \tag{2.27}$$

Интегрируя (2.26) по η в пределах от 0 до $\frac{\pi\tau}{4}$ найдём условие, которому должны удовлетворять коэффициенты c_n:

$$-\pi\left(1-\frac{\gamma}{2}\right) = \delta M e^{-\pi\frac{\pi}{|\tau|}\left(1-\frac{\gamma}{2}\right)} I_0, \tag{2.28}$$

$$I_0 = \int_0^{\frac{\pi|\tau|}{2}} \varkappa(\Theta) Q(\eta)\, d\eta. \tag{2.29}$$

Домножив (2.28) на $\cos\frac{4n}{|\tau|}\eta$ и интегрируя в тех же пределах, выразим c_n:

$$c_n = -\frac{2\left(1-\frac{\gamma}{2}\right)}{n(1+\rho^{2n})}\cdot\frac{I_n}{I_0}, \tag{2.30}$$

$$I_n = \int_0^{\frac{\pi|\tau|}{2}} \varkappa(\Theta) Q(\eta)\cos\frac{4n}{|\tau|}\eta\, d\eta. \tag{2.31}$$

Так как уравнение (2.30) является выражением $c_n = A(c_1, c_2 \dots c_n)$, где A — действительнозначная функция, то константы c_n будем находить в процессе решения задачи методом простой итерации [7, с. 150]. За начальное приближение будем брать вектор $c_0, c_1 \dots c_n, n \in \mathbb{N}, n < \infty$. Следует отметить, что значения коэффициентов c_n могут быть найдены любым другим численным методом, позволяющим решать системы нелинейных уравнений, например, методом коллокации [5, стр. 353].

2.2.4 Кривизна края заряда эллиптической формы

В данной работе рассматривается случай, при котором заряд, граница которого проходит по линии AD, имеет эллиптическую форму. Тогда кривизна $\varkappa(\Theta)$, участвующая в граничном условии для функции Жуковского (2.9), а оттуда — в выражении коэффициентов c_n (2.30), имеет вид:

$$\varkappa(\Theta) = \frac{\left(1 - \varepsilon^2 \sin^2 \Theta\right)^{3/2}}{p}, \qquad p = \frac{a^2}{b^2}, \qquad \varepsilon = \frac{\sqrt{b^2 - a^2}}{b} \tag{2.32}$$

где a, b — полуоси эллипса.

2.3 Вычисление координат точек на границе воронки взрыва

Итак, зная (2.1), (2.4), (2.6), (2.17) и (2.23) сформулируем полное выражение для искомой функции:

$$\frac{dz}{du} = \frac{N}{v_0} \frac{\vartheta_1\left(u - \frac{\pi\tau}{4}\right)\vartheta_1\left(u + \frac{\pi\tau}{4}\right)\vartheta_2\left(u - \frac{\pi\tau}{4}\right)\vartheta_2\left(u + \frac{\pi\tau}{4}\right)}{\left[\vartheta_1\left(u - \frac{\pi}{4}\right)\vartheta_4\left(u - \frac{\pi}{4}\right)\right]^{1-\gamma}\left[\vartheta_1\left(u + \frac{\pi}{4}\right)\vartheta_4\left(u + \frac{\pi}{4}\right)\right]^{1+\gamma}} e^{i\pi(1-\gamma)} e^{f(u)} \tag{2.33}$$

где N определена выражением (2.5).

Зная величины N/v_0, τ, γ и функцию $f(u)$ (фактически, коэффициенты c_n, входящие в её выражение), можно найти все геометрические и гидродинамические элементы течения. Для решения поставленной задачи достаточно найти только некоторые геометрические характеристики.

Координаты точек на границе CB на плоскости переменного z будем определять последовательным интегрированием (2.33) численно. В качестве основной формулы для построения точек границы CB используем выражение

$$z(u_0) = \int\limits_0^{u_0} \frac{dz}{du} du \tag{2.34}$$

Таким образом, для любой точки границы CB, координаты можно получить из формул (2.34) и (2.33), зная, что:

$$z(E)\Big|_{E\in CB} = \int\limits_{0}^{\varepsilon} \frac{dz}{du}(i\eta)d\eta, \qquad E = i\varepsilon, \qquad \varepsilon \in \Big[0; \frac{\pi\tau}{4}\Big] \tag{2.35}$$

Точки, координаты которых получены выражением (2.35), являются точками, лежащими на границе воронки взрыва, что и требовалось найти в задаче. Следует заметить, что формула (2.34) позволяет найти, вообще говоря, любую точку из области z, а не только лежащую на границе CB.

Глава 3. Численное решение задачи

Получив точное аналитическое решение поставленной задачи, мы выполнили первый этап работы. Теперь перенесём выведенные формулы в программный код Haskell для того, чтобы иметь возможность удостовериться в том, что модель работоспособна.

В этой главе мы последовательно перенесём все отдельные формулы, выведенные в главе 2, затем введём модуль для черчения графиков функций и, наконец, сведём всё воедино в цельном приложении. Следует учесть, что расчёт на параллельность выполнения должен быть интегрирован в прогаммный код.

3.1 Представление параметров модели в программе

В модели используются следующие постоянные параметры, которые должны быть заданы извне:

1 ϕ_0 — Значение потенциала течения на поверхности заряда;

2 v_0 — Критическое значение скорости течения на свободной поверхности;

3 τ — Параметр, требуемый для полного задания тета-функций;

4 α — Угол в точках A и D области 2.1 течения;

5 a — Малый радиус кривизны эллипса криволинейного заряда;

6 b — Большой радиус кривизны эллипса криволинейного заряда;

7 $c_n, \quad n \in [0; \infty)$ — Постоянные множители из разложения корректирующей функции (2.23) $f(u)$ в ряд Лорана.

Кроме того, вычислительный характер решения поставленной задачи подразумевает использование ряда дополнительных параметров:

1 n_{theta} — Число слагаемых в рядах (3.1) и (3.4), позволяющих вычислить значение тета-функций численно;

2 $n_{integral}$ — Частота разбиения отрезка интегрирования на сетку;

3 n_{cn} — Количество постоянных множителей c_n в разложении $f(u)$, а, следовательно, и количество слагаемых в нём;

4 ε — Желаемая точность при определении коэффициентов c_n численными методами.

Для хранения этих параметров в программе был определён тип данных, предназначенный для хранения всего массива параметров. Предполагается, что в процессе выполнения программы будет создан один экземпляр этого типа, который будет передаваться во все вспомогательные функции, участвующие в вычислениях.

```
data ModelParams = ModelParams {
    tau        :: Double,    -- \tau, passed to Functions.Theta(qpar)
    phi_0      :: Double,    -- \phi_0
    v_0        :: Double,    -- v_0
    alpha      :: Double,    -- \alpha
    a          :: Double,    -- long ellipse radius
    b          :: Double,    -- short ellipse radius
    n_theta    :: Integer,   -- number of addends in \Theta_4 and \Theta_3
    n_integral :: Integer,   -- number of elements in numeric integration
    n_cn       :: Integer,   -- number of addends in f(u), essentially number of c_n
    precision  :: Double,    -- precision of calculating fucking c_n's
    c_n        :: [Double]   -- list of c_n, it should be computed separately
    } deriving (Show)
```

Тип `ModelParams` наследует класс **`Show`** для того, чтобы был простой способ визуализировать весь массив параметров. Использование типа `ModelParams` позволяет передавать в функции, участвующие в вычислениях, только один аргумент, который хранит параметры модели, вместо одиннадцати.

3.2 Представление аналитического решения задачи в программе

3.2.1 Программирование численного интегрирования

Для численного интегрирования используется метод трапеций [7, с. 86]. Вместо того, чтобы определять общую функцию интегрирования, которую вследствие особенностей интегралов от комплексного аргумента было бы крайне затруднительно сформулировать, были определены две различные функции, отличающиеся типами входных данных.

Функция `integrateX` используется для вычисления определённого интеграла в выражении (2.30). Интегрирование производится по действительной части переменного u.

```haskell
integrateX :: (RealFloat a, Enum a) => (Complex a -> Complex a) ->
               a -> a -> Integer -> Complex a
integrateX f a b n =
    ((sum $ map f xvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      xvalues = map (:+ 0) values
      t = (f (a :+ 0) + f (b :+ 0))/2
      h = (b - a) / fromInteger(n)
```

Функция integrateY используется для интегрирования на конечном этапе вычислений 3.3, в процессе получения координат точек на области z. Интегрирование производится по мнимой части переменного u.

```haskell
integrateY :: (RealFloat a, Enum a) => (Complex a -> Complex a) ->
               a -> a -> Integer -> Complex a
integrateY f a b n =
    ((sum $ map f yvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      yvalues = map ((:+) 0) values
      t = (f (0 :+ a) + f (0 :+ b))/2
      h = (b - a) / fromInteger(n)
```

3.2.2 Вычисление значений тета-функций

Согласно [16, с. 336], Θ_4 представляется в виде:

$$\Theta_4(u) = \Theta_4(u, q) = 1 + 2\sum_{n=1}^{+\infty}(-1)^n q^{n^2}\cos(2nu) \tag{3.1}$$

Здесь q вычисляется следующим образом:

$$q = e^{\pi i \tau} : \quad |q| < 1 \tag{3.2}$$

$$\tau - \text{const} : \quad \mathrm{Im}(\tau) > 0 \tag{3.3}$$

В этой функции, таким образом, используется два параметра: n_{theta} и τ. Дополнительный параметр q (3.2) вычисляется следующей вспомогательной функцией:

```haskell
qpar tau = exp $ pi * tau * (0 :+ 1)
```

В качестве параметра tau функции qpar должен передаваться параметр модели τ.

На Haskell определение (3.1) переводится так:

```haskell
theta4 :: (RealFloat a) => Integer -> Complex a -> Complex a -> Complex a
theta4 n q u = 1 + 2 * sum thetaarg
  where thetaarg = [(signfun nn) * (qfun q nn) * (cosfun u nn) | nn <- [1..n]]
        signfun :: (RealFloat a) => Integer -> Complex a
        signfun nn
          | odd nn = -1
          | otherwise = 1
        qfun :: (RealFloat a) => Complex a -> Integer -> Complex a
```

```
        qfun q nn = q ** fromInteger(nn) ** 2
        cosfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        cosfun u nn = cos $ fromInteger(2 * nn) * u
```

Аналогично, функция Θ_3 задаётся следующим математическим выражением:

$$\Theta_3(u) = \Theta_3(u, q) = 1 + 2\sum_{n=1}^{+\infty} q^{n^2}\cos(2nu) \tag{3.4}$$

Его перевод на Haskell выглядит так:

```
theta3 :: (RealFloat a) => Integer -> Complex a -> Complex a -> Complex a
theta3 n q u = 2 * sum thetaarg
  where thetaarg = [(qfun q nn) * (cosfun u nn) | nn <- [1..n]]
        qfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        qfun q nn = q ** (fromInteger nn) ** 2
        cosfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        cosfun u nn = cos $ fromInteger(2 * nn) * u
```

В обоих функциях первый аргумент задаёт количество вычисляемых слагаемых из бесконечной суммы, которой представляются тета-функции. Для краткости введём две более коротких функции, которыми будем пользоваться в дальнейшем [1]:

```
theta3' param = theta3 (n_theta param) (qpar (0 :+ tau param))
theta4' param = theta4 (n_theta param) (qpar (0 :+ tau param))
```

3.2.3 Вычисление значений функции dw/du

Производная комплексного потенциала от вспомогательного переменного dw/du является константной функцией, и переводится на Haskell следующим образом:

```
dwdu :: ModelParams -> Double
dwdu param = ((negate 2) * phi_0') / (pi * tau')
       where phi_0' = phi_0 param
             tau' = tau param
```

Её единственный входной аргумент — массив параметров `param`.

3.2.4 Вычисление значений функции Жуковского $\chi(u)$

Функция $\chi(u)$ (2.6) переводится в следующий код на Haskell:

```
chi :: ModelParams -> Complex Double -> Complex Double
chi param u = (chi_0 param u) + (f_corr param u)
```

Функция $\chi_0(u)$ (2.17) переводится в

[1] Здесь и далее аргумент «param» является переменной, хранящей значения всех параметров модели.

```
chi_0 :: ModelParams -> Complex Double -> Complex Double
chi_0 param u = c_chi_0 + const_coeff * log ( divident / divisor)
  where const_coeff = ((pi / 2) * (1 + alpha')) :+ 0
        divident  = (theta3' param 0) * (theta4' param u)
        divisor   = (theta3' param u) * (theta4' param 0)
        alpha'  = alpha param
```

Параметр $\Theta(u) = \operatorname{Im}\chi(u)$ будем получать естественным образом, вычисляя функцию $\chi(u)$ и определяя мнимую часть полученного комплексного числа:

```
imagPart(chi param u)
```

3.2.5 Вычисление значений корректирующей функции $f(u)$

Следующим образом выглядит функция Haskell, реализующая (2.23):

```
f_corr :: ModelParams -> Complex Double -> Complex Double
-- Clear functional implementation
f_corr param u = foldl (+) (c0 :+ 0) (f_arg u clist )
  where f_arg u clist = map (\ (n, cn) -> (cn :+ 0) * (exp' u (i' n) - exp' u (negate (i' n)))) clist
        i' n      = (fromInteger n) :+ 0
        clist     = zip [1..( pred $ n_cn param)] cn'
        exp' u n = exp $ 2 * (0 :+ 1) * n
        cn'       = tail $ c_n param
        c0        = head $ c_n param
```

Для корректной работы эта функция должна получить из массива данных `param` список коэффициентов c_n.

3.2.6 Вычисление коэффициентов c_n в разложении $f(u)$ в ряд

Как было сказано в 2.2.3, коэффициенты c_n, необходимые для определения $f(u)$, требуется вычислить отдельно, до того, как заниматься вычислением координат точек на границе воронки взрыва.

Для вычисления c_n определим следующую функцию, которая последовательно будет уточнять c_n, до тех пор, пока не будет достигнута требуемая точность.

```
calc_cn param
  | has_error param new_cn = param {c_n = new_cn}
  | otherwise  = calc_cn (param {c_n = new_cn})
    where
      has_error param new_cn = foldl (&&) True $ map (calc_error) (zip cn' new_cn)
      calc_error (x1, x2)      = (((x2 - x1) ** 2 ) / abs (2 * x1 - x2) ) < precision'
      new_cn                   = map (transform') [0..n']
      transform' n             = s_fun param n * (integrate (transform n) 0 (pi/2) ni')
      transform n x            = curvature param (x :+ eta)
                                   * exp ( - realPart(chi param (x :+ eta)))
                                   * cos (2 * x * fromInteger n)
      eta          = (pi * tau') / 2
```

```
      tau'        = tau param
      precision ' = precision param
      ni '        = n_integral param
      cn'         = c_n param
      n'          = n_cn param
```

`calc_cn` — рекурсивная функция со следующим условием останова:

$$\frac{(x_t - x_{t-1})^2}{|2x_{t-1} - x_t - x_{t-2}|} < \varepsilon \tag{3.5}$$

где ε есть заранее заданная константа, определяющая желаемую точность вычислений.

На каждой итерации уточнения вектора коэффициентов c_n функция `calc_cn` обновляет содержимое поля `c_n` в блоке данных `param`. При достижении условия остановки вектор `c_n` можно использовать в численном решении поставленной в данной работе задачи.

3.2.7 Программирование отображения значений дополнительного переменного на область исходного переменного

Собственно, конечная цель раздела 2, функция dz/du, определённая выражением (2.33), переводится на Haskell следующим образом:

```
dzdu :: ModelParams -> Complex Double -> Complex Double
dzdu param u = ( ((dwdu param) / v') :+ 0 ) * exp ( chi param u )
  where v' = v_0 param
```

Все остальные функции уже были определены ранее.

3.3 Программирование вычисления координат точек на границе воронки взрыва

Как следует из (2.35), точки на границе CD в области исходного переменного z можно найти интегрированием функции dz/du (2.33) по линии CD области вспомогательного переменного u. Это выполняется при помощи следующей функции Haskell:

```
zlist :: ModelParams -> [Complex Double]
zlist param = map (z param) [a', a' + h .. b']
  where
    z param e = integrateY (dzdu param) 0 e n'
    h = (b' - a') / fromInteger n'
    n' = n_integral param
    a' = 0
    b' = ( pi * (tau param) ) / 2
```

zlist отображает массив точек $\xi + i\frac{\pi\tau}{4}, \quad \xi \in \left[0; \frac{\pi}{4}\right]$ в точки $x + iy$ последовательным интегрированием. Получив координаты этих точек, мы решим задачу.

3.4 Программный пакет для вывода графиков функций средствами Haskell

Для вывода графиков будет использоваться пакет расширения для Haskell под общим названием Chart (см. [20], [22]). Chart поддерживает вывод непосредственно в файл с растровым изображением а также черчение на канве окна GTK+.

Процедура outputData, выполняющая вывод графика по заданным точкам, определена следующим образом:

```
outputData datalist = do
    renderableToWindow (toRenderable (chart datalist)) 640 480
    renderableToPNGFile (toRenderable (chart datalist)) 640 480 "test.png"

chart :: [(Double, Double)] -> Layout1 Double Double
chart datalist = layout
  where
    myPlot = plot_lines_values ^= [datalist]
               $ plot_lines_style .> line_color ^= opaque blue
               $ plot_lines_title ^= "test"
               $ defaultPlotLines
    layout = layout1_title ^= "Graphics.Rendering.Chart"
           $ layout1_plots ^= [Left (toPlot myPlot)]
           $ defaultLayout1
```

Функция chart нужна для удобства настройки оформления графика. Через аргумент datalist функция получает набор точек в виде списка пар (x, y).

3.5 Сборка цельного приложения с учётом многопоточности

Все функции, определённые в разделе 3.2, завернём в модуль под названием BlastModel (см. приложение A), за исключением функций вычисления значений тета-функций. Тета-функции вынесем в модуль под названием Theta (см. приложение B).

В главном модуле находится программный код для черчения графиков, определённый в подразделе 3.4, и точка входа программы.

Точка входа программы, функция main, состоит в целом из четырёх шагов:

1 получение параметров модели от пользователя;

2 уточнение коэффициентов c_n (это необходимый шаг, который, увы, приходится совершать именно на этом этапе выполнения программы);

3 вычисление координат точек на границе воронки взрыва;

4 черчение кривой по полученным координатам.

Все эти шаги должны быть выполнены последовательно, поэтому никакой параллельности на этом уровне абстракции не предполагается. Вот как выглядит точка входа для всей программы в целом:

```
main = do
  putStrLn "Greetings!~ This is blast model, based on solid-liquid model of Lavrentyev and Kotlyar."
  putStrLn "First, we will define model parameters."
  par <- getModelParameters
  putStrLn "Second, we compute the parameters c_n needed for computations"
-- let par_corrected = calc_cn par
  putStrLn "We will now compute points on the edge of blast."
  let pointlist = computePoints par
  putStrLn "And at last, we print the data to the file for a Gnuplot"
  outputData pointlist
  putStrLn "All done, good bye."
```

Для получения от пользователя параметров была выделена вспомогательная функция getModelParameters следующего вида:

```
getModelParameter parname = do
        putStrLn $ "Value of " ++ parname
        getLine

getModelParameters = do
  phi_0'        <- getModelParameter "phi_0"
  v_0'          <- getModelParameter "v_0"
  tau'          <- getModelParameter "|tau|"
  alpha'        <- getModelParameter "alpha"
  a'            <- getModelParameter "a"
  b'            <- getModelParameter "b"
  n_theta'      <- getModelParameter "n_theta"
  n_integral'   <- getModelParameter "n_integral"
  n_cn'         <- getModelParameter "n_cn"
  precision'    <- getModelParameter "precision"
  return ModelParams{
       phi_0      = read phi_0',
       v_0        = read v_0',
       tau        = read tau',
       alpha      = read alpha',
       a          = read a',
       b          = read b',
       n_theta    = read n_theta',
       n_integral = read n_integral',
       n_cn       = read n_cn',
       precision  = read precision',
       c_n        = take n_cn' $ repeat 2
       }
```

Для уточнения значений c_n используется определённая в BlastModel функция calc_cn, которая подробно описана в разделе 3.2.6.

Для вычисления координат точек на границе воронки определена дополнительная функция computePoints следующего вида:

```
computePoints par = zip xlist  ylist
  where n' = fromInteger(n_integral par)
        ylist  = map (\x -> (x - 1) ** 2 - (n' / 2)) xlist
        xlist  = [0..n']
```

Для вывода данных, которые были сгенерированы функцией computePoints, используется функция outputData, описанная в разделе 3.4.

Запуск приложения следует осуществлять из командной строки. В качестве аргументов командной строки следует передавать параметры +RTS -N2 (где 2 — это количество ядер). Все параметры модели программа запросит самостоятельно, после чего выполнит предусмотренные вычисления и выдаст график в отдельном окне и в файл PNG в рабочем каталоге.

Полный текст программы находится в приложении. Содержимое исходного кода главного модуля программы расположен в приложении C.

Глава 4. Анализ полученных результатов

4.1 Результаты вычислений и их интерпретация

Для разработанной программы был сформирован ряд тестов, для массового вывода графиков с вариацией того или иного параметра модели. Было получено более 150 различных графиков, после чего дальнейшие эксперименты были признаны несущественными для оценки работы.

В таблице 4.1 приведены сведения о параметрах, использованных в некоторых проверочных запусках. Параметры точности оставались неизменными: интегрирование производилось по сетке частотой в 50 отрезков, тета-функции представлялись рядами из 25 элементов, и количество вычисляемых c_n было равно 30.

График на рис.	φ_0	v_0	τ	α	a	b
4.1	100	0.2	0.7	0.7	1	5
4.2	100	0.2	0.6	0.7	1	5
4.3	100	0.2	0.7	0.49	1	5

Таблица 4.1. Параметры трёх различных тестов черчения границы воронки взрыва. Точность вычислений не менялась

Было замечено, что параметры φ_0 и v_0 влияют только на масштаб чертежа. При увеличении φ_0 чертёж увеличивается (граница отдаляется от начала координат), а при увеличении v_0 — уменьшается (граница приближается к началу координат). Эти результаты естественным образом следуют из построения функции (2.33), где эти параметры выступают в качестве коэффициентов.

Параметры a и b, представляющие собой длины радиусов эллипса, который является формой заряда, оказывают также масштабирующее влияние на чертёж. При этом при задании какого-либо b так, что $b < a$ программа начинает выдавать векторы из ошибочных значений «NaN» при вычислении коэффициентов c_n для выражения $f(u)$.

Параметры $|\tau|$ и γ оказывают намного более существенное влияние на чертёж. Оба этих параметра искажают и поворачивают границу, изображённую на графике, причём весьма сложным образом. Для определения

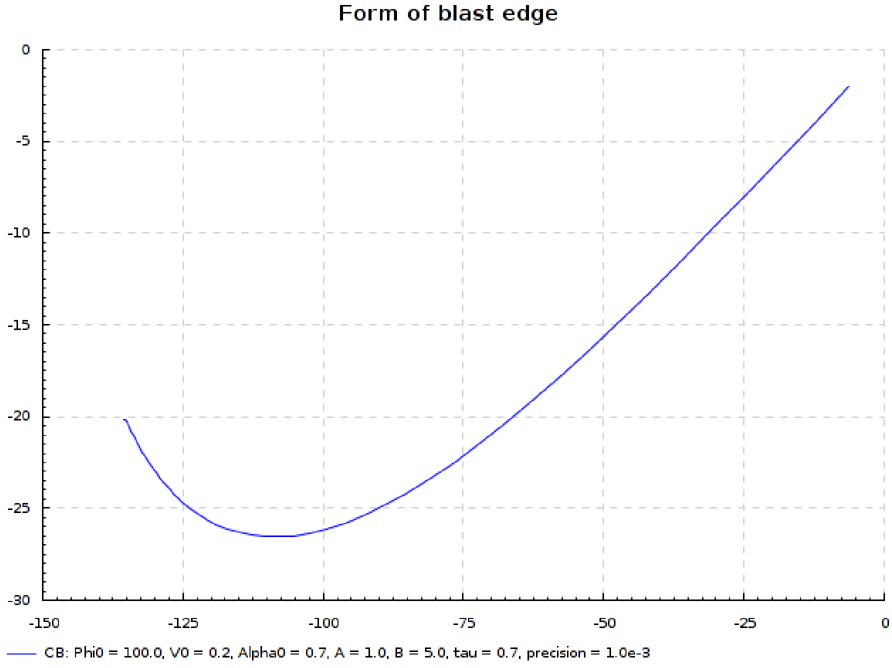

Рис. 4.1. Чертёж границы воронки взрыва при первом тестовом наборе параметров

особенностей влияния γ и $|\tau|$ на расположение и форму изображения границы воронки взрыва следует провести дополнительное исследование.

В целом интерпретация полученных результатов затруднена. На фоне аналитически верных математических формул, лежащих в основе производимых вычислений, не поддающиеся расшифровке производимые программой графики представляют собой проблему для дальнейшего применения результатов данной работы.

Следует отметить, что из-за особенностей численной реализации вычисления значений тета-функций, в случае задания более чем 25 слагаемых в ряду, представляющем тета-функции (иными словами, в случае, когда $n_{theta} > 25$), тета-функции начинают расходиться и их вычисление крайне замедляется. Это происходит из-за накопления ошибки. Уже при $n_{theta} > 10$ параметр q, входящий в разложение тета-функций, начинает возводиться в степень n^2, где $n > 10$, а при условии, что $|q| < 1$, получается, что все слагаемые в разложении тета-функции, после десятого становятся исчезающе малы.

4.2 Оценка производительности

Вычисления производились на AMD Athlon 64 X2 Dual Core 5000+ 1.96 GB оперативной памяти. Сразу же следует заметить, что в случае компиляции программы производительность возрастает на 200%. В

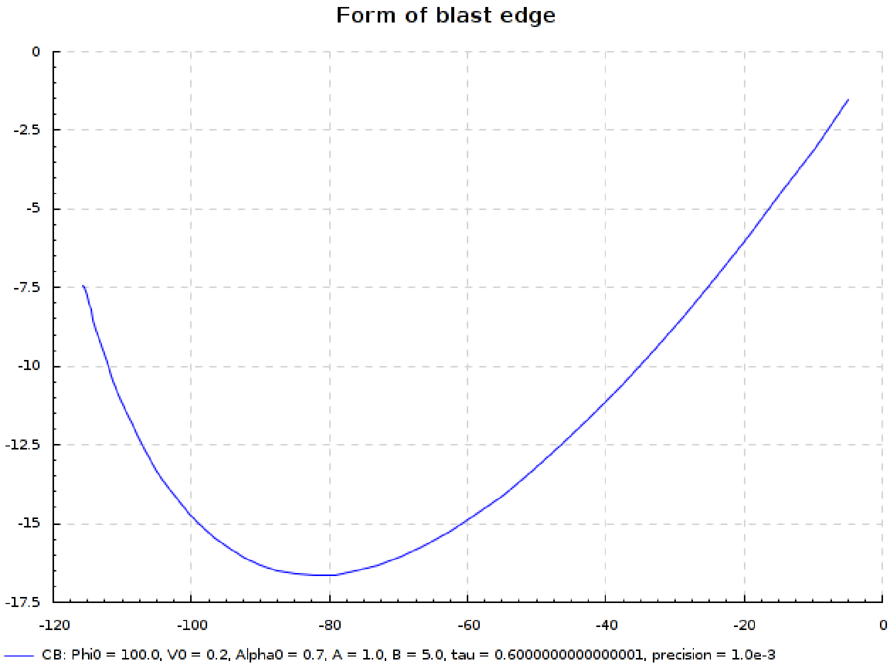

Рис. 4.2. Чертёж границы воронки взрыва при втором тестовом наборе параметров

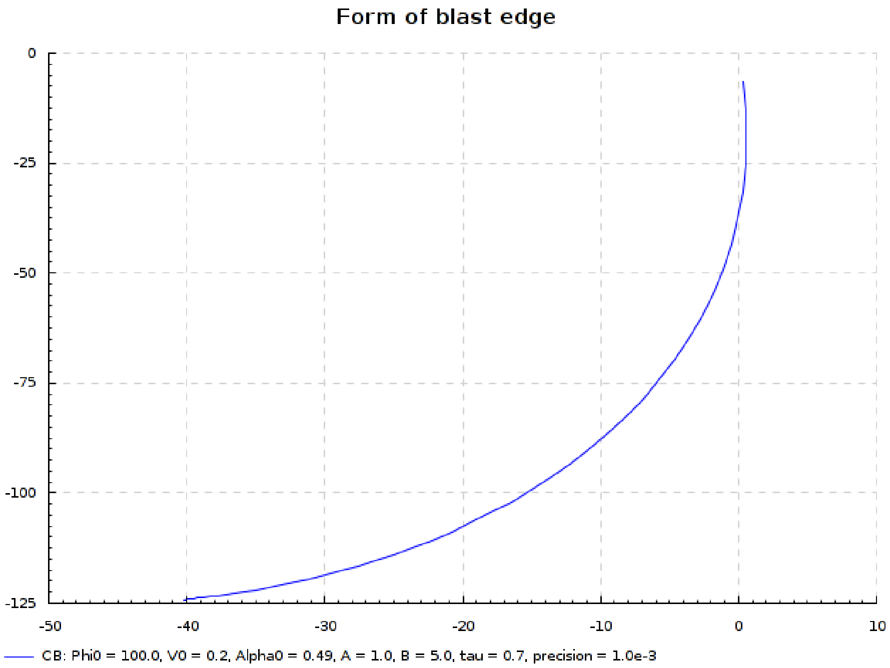

Рис. 4.3. Чертёж границы воронки взрыва при третьем тестовом наборе параметров

среднем время выполнения программы составляет 40 секунд плюс-минус пять при любом наборе входных параметров.

Ошибка, связанная с количеством элементов, представляющих тета-функции, проявляется и на производительности. В случае превышения количества элементов дальше 25 производительность программы падает.

На одну итерацию вычисления c_n на тестовом компьютере требуется от 5.9 до 6.9 секунд. На вычисление координат 50 точек, например, для постройки рисунка одной из границ области $z(u)$, требуется 7 секунд. Эти сведения по времени выполнения актуальны для скомпилированного приложения, выполняемого в среде GNU \Linux.

Внедрение техник полуавтоматического распараллеливания, предоставляемых компилятором GHC, дало противоречивые результаты. При внедрении какого-либо кода, предназначенного для указания на параллельные вычисления, в программу, производительность конечного приложения падает на 20%. Данное наблюдение распространяется на любые вычисления, проводимые программой.

При наличии в программе кода для осуществления параллельных вычислений и использовании аргументов командной строки `+RTS -N2` во время запуска программы время выполнения увеличивается. При семантически однопоточном коде и использовании одного только указания `+RTS -N2` производительность тоже падает, хотя и менее значительно.

Такое поведение приложения связано, очевидно, с тем, что в операционной системе, в которой выполняется приложение, работает свой собственный механизм распараллеливания задач. Это предположение подтверждается наблюдением за нагрузкой процессора сторонними приложениями.

Применение методики увеличения объёма кучи и стека, выделяемых на программу, с целью уменьшения количества вызовов сборщика мусора, также понижает производительность. Следует, однако, отметить при этом, что количество запусков сборщика мусора действительно уменьшается. Был произведён эксперимент с запуском приложения, выделив ему 100 MB для стека и 500 MB для кучи. Время, проведённое за сборкой мусора, уменьшилось с 2 сек, получаемых при запуске с параметрами по умолчанию, до 0.2 сек. При этом суммарное время выполнения программы увеличилось.

Определённо, можно сделать вывод о том, что, кроме, собственно, применения техник распараллеливания, а тем более, таких специфичных, как предоставляемых средой выполнения Haskell, требуется ещё дополнительно проводить исследования для выявления участков кода, распараллеливание которых действительно приведёт к повышению производительности. В остальных случаях компиляция с оптимизацией, предоставляемая GHC, позволяет добиться наилучшей производительности без применения каких-либо дополнительных техник.

Заключение

Задача была решена полностью.

1 В качестве первого этапа работы была построена и решена краевая задача, позволяющая описать течение, эквивалентное процессу взрыва согласно ТЖМ.

2 Сформулированная на первом этапе математическая модель взрыва была переформулирована на язык Haskell с расчётом на технику многопоточности.

Следует отметить крайнюю лёгкость написания программ на языке Haskell, по сравнению с C — традиционным языком для написания программ, занимающихся сложными вычислениями. В некоторых случаях определение подпрограммы на Haskell с точностью до обозначений напоминает определение соответствующей математической функции.

Разработано консольное приложение для ОС GNU\Linux, позволяющее произвольно задавать параметры модели и получать чертежи свободной поверхности, соответствующей заданным параметрам. В качестве дополнения для облегчения анализа реализована возможность массового вывода чертежей, с варьированием заранее заданного параметра. При запуске возможно задать количество доступных на аппаратной базе ядер для распараллеливания вычислений. Приложение переносимо на другие операционные системы, при условии, что оно должно быть предварительно скомпилировано перед использованием.

Сформулированная модель может быть с учётом её недостатков, отмеченных в разделе 4.1, использована для расчётов реальных воронок от взрывов заглублённых зарядов. Полученные формулы достаточно общие, чтобы использовать их для моделирования взрывов зарядов любой формы, не ограничиваясь эллипсоидами. В перспективе видится возможность вывода решения ещё более общей задачи, в различных постановках.

Реализованная программа может быть с учётом особенностей работы, выявленных в разделе 4.2, дополнена и развита до полноценного приложения, пригодного для использования на местах конкретными специалистами по взрывотехнике.

Полученная программа может быть использована несколькими способами:

1 для реальных расчётов воронок от взрывов заглублённых зарядов эллиптической формы;

2 для дальнейших исследований в области моделирования взрывов при помощи ТЖМ;

3 в качестве примера для студентов младших курсов по дисциплине «параллельные вычисления».

О данной работе был сделан доклад [14] на межрегиональной научно-практической конференции студентов, аспирантов и молодых учёных «III Камские чтения», занявший 2 место в секции «Естественные науки».

Представляется целесообразным продолжить работу по данной теме. Полученные в рамках решения данной задачи результаты подтверждают необходимость дальнейших исследований. Также заметим, что характерные особенности решения задачи безоговорочно требуют применения вычислительной техники для получения необходимого результата, поэтому данная работа в целом находится на стыке двух наук: математической физики и инженерии программного обеспечения, представляя собой синтетическую задачу, с образовательной, прикладной и научной точек зрения весьма полезную.

Список литературы

1. *Броуд, Г.* Расчёты взрывов на ЭВМ. Подземные взрывы / Г. Броуд. — М.: Мир, 1975. — 163 с.

2. *Власов, О. Е.* Основы теории действия взрыва / О. Е. Власов. — М.: Изд-во ВИА, 1957.

3. *Гуревич, М. И.* Теория струй идеальной жидкости / М. И. Гуревич. — Наука, 1979.

4. Действие ядерного взрыва. Сб. переводов / Под ред. С. С. Григорян, Г. С. Шапиро. — М.: Мир, 1971. — 312 с.

5. *Демидович, Б. П.* Численные методы анализа. Приближение функций, дифференциальные и интегральные уравнения / Б. П. Демидович, И. А. Марон. — 2 изд. — М.: Физматгиз, 1963.

6. *Ильинский, Н. Б.* Краевые задачи теории взрыва / Н. Б. Ильинский, А. В. Поташёв. — Казань: Изд-во Казанского университета, 1986.

7. *Калиткин, Н. Н.* Численные методы / Н. Н. Калиткин. — М.: Главная редакция физико-математической литературы изд-ва «Наука», 1978.

8. *Котляр, Л. М.* О взрыве на поверхности грунта линейно-распределённого заряда криволинейной формы / Л. М. Котляр // *Журнал прикладной механики и технической физики.* — 1975. — № 1. — С. 187–191.

9. *Кузнецов, В. М.* О форме воронки выброса при взрыве на поверхности грунта / В. М. Кузнецов // *ПМТФ.* — 1962. — № 3. — С. 152–156.

10. *Кузнецов, В. М.* Математические модели взрывного дела / В. М. Кузнецов. — Новосибирск: Наука, 1977. — 264 с.

11. *Лаврентьев, М. А.* Вариационные методы краевых задач для систем уравнений эллиптического типа / М. А. Лаврентьев. — М.: Издательство АН СССР, 1962.

12. *Мартынюк, П. А.* О форме воронки выброса при взрыве в грунте шнурового заряда / П. А. Мартынюк // Народно-хозяйственное использование взрыва. — Новосибирск: СО АН СССР, 1965. — С. 3–9.

13. *Муйземнек, А. Ю.* Математическое моделирование процессов удара и взрыва в программе Ls-dyna: учебное пособие / А. Ю. Муйземнек, А. А. Богач. — Пенза: Информационно-издательский центр ПГУ, 2005. — 106 с.

14. *Сафронов, М. А.* Задача о взрыве криволинейного заглублённого заряда / М. А. Сафронов // III Камские чтения: межрегиональная научно-практическая конференция. В 3-х частях. — Т. 3. — Набережные Челны: 2011.

15. Технические правила ведения взрывных работ на дневной поверхности. — М.: Недра, 1972. — 240 с.

16. *Уиттекер, Э. Т.* Курс современного анализа: В 2 т. / Э. Т. Уиттекер, Д. Н. Ватсон. — М.: Физматгиз, 1963. — Т. 2: Трансцендентные функции.

17. *Шабат, Б. В.* Функции комплексного переменного и некоторые их приложения / Б. В. Шабат, Б. А. Фукс. — Наука, 1964.

18. Control.concurrent. — HackageDB The Haskell Packages Database. — 2010 (проверялась 25 апреля 2011). — [В Сети]. http://www.haskell.org/ghc/docs/latest/html/libraries/base-4.3.1.0/Control-Concurrent.html.

19. *Dean, J.* Mapreduce: Simplified data processing on large clusters / J. Dean, S. Ghemawat // OSDI'04: Sixth Symposium on Operating System Design and Implementation. — San Francisco: 2004.

20. *Docker, T.* Haskell charts. — Haskell Charts wiki. — 2010 (проверялась 25 апреля 2011). — [В Сети]. http://http://dockerz.net/twd/HaskellCharts.

21. Erlang programming language. — Erlang web node. — 2011 (проверялась 25 апреля 2011). — [В Сети]. http://www.erlang.org/.

22. Hackagedb: chart-0.14. — HackageDB The Haskell Packages Database. — 2010 (проверялась 25 апреля 2011). — [В Сети]. http://hackage.haskell.org/package/Chart.

23. Hackagedb: parallel-3.1.0.1. — HackageDB The Haskell Packages Database. — 2010 (проверялась 25 апреля 2011). — [В Сети]. http://hackage.haskell.org/package/parallel.

24. The haskell programming language. — HaskellWiki. — 2011 (проверялась 27 апреля 2011). — [В Сети]. http://haskell.org/haskellwiki/Haskell.

25. *Hennessy, J. L.* Computer architecture: a quantitative approach / J. L. Hennessy, D. A. Patterson. — 4th edition. — USA: Elsevier, Inc, 2007. — 704 pp.

26. *Jones, S. P.* Beautiful Concurrency / S. P. Jones // Beautiful Code / Ed. by G. Wilson; Microsoft Research, Cambridge. — O'Reilly, 2007.

27. *Kernighan, B.* The C programming language / B. Kernighan. — Englewood Cliffs, N.J: Prentice Hall, 1988.

Приложения

A BlastModel.hs

```
module BlastModel where
------------------------------------------
-- Модель взрыва заглублённого в грунт заряда криволинейной формы.
-- Позволяет вычислить границу воронки взрыва на основании ряда параметров.
-- Взрыв представляется как потенциальное течение струи идеальной жидкости, а воронка взрыва --- как линия тока на течении, вдоль которой скорость течения равна некоторому "критическому" значению.
------------------------------------------

-- Мы используем параллельность!
import Control.Parallel.Strategies
import Control.Parallel

-- Наша модель построена на комплекснозначных функциях комплексного аргумента
import Data.Complex

-- Импортируем самописные тета-функции
import Theta

-- Параметры модели будем передавать объектом следующего типа:
data ModelParams = ModelParams {
    tau        :: Double,     -- параметр, доопределяет тета-функции
    phi_0      :: Double,     -- начальное значение потенциала течения
    v_0        :: Double,     -- критическое значение скорости, скорость течения равна v_0 на границе воронки взрыва
    alpha      :: Double,     -- угол, с которым граница заряда наклонена к оси абсцисс
    a          :: Double,     -- заряд предполагается эллиптическим, поэтому это больший радиус заряда
    b          :: Double,     -- заряд предполагается эллиптическим, поэтому это меньший радиус заряда
    n_theta    :: Integer,    -- количество слагаемых в ряду, представляющем тета-функцию (т. е., это, по сути, точность вычислений тета-функций)
    n_integral :: Integer,    -- частота разбиения отрезка интегрирования
    n_cn       :: Integer,    -- количество коэффициентов cN в разбиении f(u) в ряд, т. е., заодно и точность вычисления f(u)
    precision  :: Double,     -- точность вычисления cN методом простых итераций. Да и вообще "точность" там, где она может быть нужна
    c_n        :: [Double]    -- список коэффициентов cN в разбиении f(u) в ряд. При задании параметров равны начальному приближению, потом уточняются.
-- Не существует c_n !! 0, параметр c0 задаётся в формуле, и здесь не хранится и не обновляется.
    } deriving (Show)

-----------------------------------------
-- 0. Различные хелперы для упрощения работы BEGIN

-- Переобозначения тета-функций в более удобоваримый вид
theta1' param = theta1 (n_theta param) (qpar (0 :+ tau param))
theta2' param = theta2 (n_theta param) (qpar (0 :+ tau param))
theta3' param = theta3 (n_theta param) (qpar (0 :+ tau param))
theta4' param = theta4 (n_theta param) (qpar (0 :+ tau param))
```

```
-- Параметры "по умолчанию", для упрощения отладки
null_parameters = ModelParams {
    tau        = 0.7,
    phi_0      = 100,
    v_0        = 0.2,
    alpha      = 0.25,
    a          = 2,
    b          = 5,
    n_theta    = 25, -- you'll never need more, 'cause there's an q ** n_theta ** 2 in definition of
        both theta-functions with q < 1
    n_integral = 50,
    n_cn       = 30,
    precision  = 0.001,
    c_n        = take 30 $ repeat 0
    }

-- Функция для удобства печати коэффициентов cN
printCnList :: ModelParams -> IO()
printCnList param = do
  let cnlist = c_n param
      nlist  = [1..(n_cn param)]
      cnnstr = (\(n, cn) -> "n" ++ (show n) ++ ": " ++ (show cn))
  mapM print $ map cnnstr (zip nlist cnlist )
  return ()

-- Более-менее удобное изменение параметров
renew_params param (phi_0', v_0', tau', alpha', a', b') =
  param {phi_0 = phi_0', v_0 = v_0', tau = tau', a = a', b = b'}

-- 0. Различные хелперы для упрощения работы END
-----------------------------------------------

-----------------------------------------------
-- 1. Вычисление координат точек на границе воронки взрыва BEGIN

-- Функция для вычисления сразу всех координат
--  получает параметры модели
--  выдаёт список координат точек, пригодных для передачи их на график
zlist :: ModelParams -> [(Double, Double)]
zlist param = map constructPoint [a', a' + h .. b']
  where
    constructPoint e = asPoint $ z param e
    asPoint u = (realPart u, imagPart u)
    h = (b' - a') / fromInteger n'
    a' = precision param
    b' = ( pi * (tau param) ) / 2
    n' = n_integral param

-- Функция для вычисления координат одной точки на границе воронки взрыва
--  получает параметры модели и одну координату точки на границе воронки в поле 'u'
--  выдаёт координаты одной точки в поле 'z'
-- В данной модели интегрирование производится по линии (pi/4) + i*e'
z :: ModelParams -> Double -> Complex Double
z param e = finalIntegrate (dzdu param) lowl e n'
  where n' = n_integral param
        lowl = precision param


-- Функции для получения точек на всех границах области по отдельности
```

```
-- Точки на границе CD
zlistCD  param = map constructPoint [a', a' + h .. b']
  where
    constructPoint  e = asPoint $ zCD param e
    asPoint  u = (realPart u, imagPart u)
    h = (b' - a') / fromInteger n'
    a' = precision  param
    b' = pi / 4
    n' = n_integral param

zCD param e = integrateCD (dzdu param) lowl e n'
  where n' = n_integral param
        lowl  = precision  param

-- Точки на границе BA
zlistBA  param = map constructPoint [a', a' + h .. b']
  where
    constructPoint  e = asPoint $ zBA param e
    asPoint  u = (realPart u, imagPart u)
    h = (b' - a') / fromInteger n'
    a' = precision  param
    b' = pi / 4
    n' = n_integral param

zBA param e = integrateBA tau' (dzdu param) lowl e n'
  where n'   = n_integral param
        lowl  = precision  param
        tau'  = tau param

-- Точки на границе CB
zlistCB  param = map constructPoint [a', a' + h .. b']
  where
    constructPoint  e = asPoint $ zCB param e
    asPoint  u = (realPart u, imagPart u)
    h = (b' - a') / fromInteger n'
    a' = precision  param
    b' = ( pi * (tau  param) ) / 4
    n' = n_integral param

zCB param e = integrateCB (dzdu param) lowl e n'
  where n' = n_integral param
        lowl  = precision  param

-- Точки на границе DA
zlistDA  param = map constructPoint [a', a' + h .. b']
  where
    constructPoint  e = asPoint $ zDA param e
    asPoint  u = (realPart u, imagPart u)
    h = (b' - a') / fromInteger n'
    a' = precision  param
    b' = ( pi * (tau  param) ) / 4
    n' = n_integral param

zDA param e = integrateDA (dzdu param) lowl e n'
  where n' = n_integral param
        lowl  = precision  param

-- 1. Вычисление координат точек на границе воронки взрыва END
----------------------------------------------------------------------
```

```haskell
------------------------------------
-- 2. Координаты точки на границе воронки взрыва получаем интегрированием dz/du BEGIN

dzdu :: ModelParams -> Complex Double -> Complex Double
dzdu param u = ((dwdu param u) / v_0') * exp ( negate $ chi param u )
  where v_0' = (v_0 param :+ 0)

-- 2. Координаты точки на границе воронки взрыва получаем интегрированием dz/du END
------------------------------------

------------------------------------
-- 3. Производная комплексного потенциала BEGIN

dwdu :: ModelParams -> Complex Double -> Complex Double
dwdu param u = (npar * mpar * divident) / divisor
  where
    npar = 0 :+ ((phi_0' * 2) / pi)
    mpar = (mdivident / mdivisor) ^ 2
    mdivident = theta2' param 0 * theta3' param 0 * theta4' param 0
    mdivisor = abs $ (theta1' param pA) ^ 2
    pA = (pi/4) :+ (pi * tau' / 4)
    divident = theta1' param pB' * theta1' param pB * theta2' param pB' * theta2' param pB
    divisor  = theta1' param pD' * theta1' param pD * theta4' param pD' * theta4' param pD
    pB  = u + (0 :+ (pi * tau' / 4))
    pB' = u - (0 :+ (pi * tau' / 4))
    pD  = u + ((pi / 4) :+ 0)
    pD' = u - ((pi / 4) :+ 0)
    tau' = tau param
    phi_0' = phi_0 param

-- 3. Производная комплексного потенциала END
------------------------------------

------------------------------------
-- 4. Функция Жуковского BEGIN

chi :: ModelParams -> Complex Double -> Complex Double
chi param u = (chi_0 param u) - (f_corr param u)

-- 4. Функция Жуковского END
------------------------------------

------------------------------------
-- 5. Функция Жуковского для упрощённой задачи BEGIN
chi_0 :: ModelParams -> Complex Double -> Complex Double
chi_0 param u = cpar + (gamma :+ 0) * log (divident / divisor)
  where
    cpar = 0 :+ (pi * (gamma - 1))
    gamma = alpha param
    divident = (theta1' param pD ) * (theta4' param pD )
    divisor  = (theta1' param pD') * (theta4' param pD')
    pD  = u + ((pi / 4) :+ 0)
    pD' = u - ((pi / 4) :+ 0)

-- 5. Функция Жуковского для упрощённой задачи END
------------------------------------

------------------------------------
-- 6. Корректирующая функция f(u) BEGIN
f_corr :: ModelParams -> Complex Double -> Complex Double
```

```haskell
f_corr param u = const_part + (foldl (+) c0 (f_arg u clist ))
  where
    const_part    = ((4 * (1 - (gamma/2)) * negate(1/tau') ) :+ 0) * u
    gamma         = alpha param
    c0            = 0 :+ 0 -- :)
    f_arg u clist = map (\ (n, cn) -> ((cn * (1 - rho n)) :+ 0) * exp' n) clist
     clist        = zip [1..( n_cn param)] cn'
    exp' n        = exp $ (4 * u - pi * (fromInteger n)) / (tau' :+ 0)
    rho n         = exp $ (negate (2 * pi * fromInteger n)) / tau'
    cn'           = c_n param
    tau'          = tau param

-- 6. Корректирующая функция f(u) END
------------------------------------------

-- Здесь мы закончили с каркасом для вычисления собственно координат точек. Но для того, чтобы корректирующая функция работала, необходимо, чтобы были вычислены коэффициенты cN.

------------------------------------------
-- 7. Вычисление коэффициентов cN BEGIN

-- Функция для обновления коэффициентов, уже записанных в ModelParams.
-- Она рекурсивная с условием остановки. Основана на методе простых итераций.
renew_cn_all :: ModelParams -> IO (ModelParams)
renew_cn_all param = do
  print "renew_cn_all started"
  new_param <- calc_new_cnlist param
  err <- high_error param new_param
  if err == True
    then renew_cn_all new_param
    else return $ new_param

-- Дополнительная функция для вычисления новых коэффициентов.
-- Получает текущие ModelParams
-- Возвращает обновлённые ModelParams
calc_new_cnlist :: ModelParams -> IO (ModelParams)
calc_new_cnlist param = do
  print "calc_new_cnlist started"
  print "current cnlist:"
  printCnList param
  -- Вычисляем новые параметры
  let n' = n_cn param
      new_cn = map (calc_cn param) [1..n']
      new_param = param {c_n = new_cn}
  print "new cnlist:"
  printCnList new_param
  return new_param

-- Дополнительная функция для определения того, велика ли ошибка (и надо ли продолжать вычисления)
-- Получает старые ModelParams и новые ModelParams.
-- Возвращает True если ошибка велика и False в противном случае
high_error :: ModelParams -> ModelParams -> IO(Bool)
high_error old_param new_param = do
  let old_cn = c_n old_param
      new_cn = c_n new_param
       errlist = map calc_error (zip old_cn new_cn)
      err = (> precision') $ foldl (+) 0 errlist
      -- Вычисляем ошибку
```

```
      calc_error (x1, x2) = ((x2 - x1) ** 2 ) / abs (2 * x1 - x2)
      precision' = precision old_param
  print $ "Error␣list :␣" ++ ((show . map (<precision')) errlist )
  print $ "Is␣error␣big␣totally :␣" ++ (show err)
  return err

-- Функция для вычисления нового коэффициента на основе вектора существующих.
--  принимает объект ModelParams (оттуда берёт вектор cN)
--  принимает номер коэффициента M
--  возвращает число --- уточнённое значение c(M)
calc_cn :: ModelParams -> Integer -> Double
calc_cn param n = (negate divident / divisor) * (ifun param n / ifun param 0)
  where
    divident = 2 * (1 - (gamma / 2))
    divisor  = fromInteger n * (1 + rho)
    rho      = exp $ negate 2 * (pi/tau') * fromInteger n
    tau'     = tau param
    gamma    = alpha param

ifun :: ModelParams -> Integer -> Double
ifun param n = integrate f lowl (pi * tau' / 4) n'
  where
    lowl     = precision param -- Это хак. В нуле функция qfun' обращается в бесконечность
        (деление на ноль).
    f e      = curvature param (t e) * qfun' param e * cosfun e
    cosfun e = cos $ 4 * fromInteger n * e / tau'
    n'       = n_integral param
    tau'     = tau param
    t e      = imagPart (chi param (0 :+ e))

qfun' :: ModelParams -> Double -> Double
qfun' param e = ((divident ^ 2) * (efun param e)) / ( divisor1 * divisor2 )
  where
    divident = (realPart . abs) $ theta2' param eA * theta2' param eA'
    divisor1 = ((** gamma1) . realPart . abs) $ (theta2' param ie) * (theta3' param ie)
    divisor2 = ((** gamma2) . realPart . abs) $ (theta1' param ie) * (theta4' param ie)
    eA  = ((pi / 4) :+ (e + (pi * tau' / 4)))
    eA' = ((pi / 4) :+ (e - (pi * tau' / 4)))
    ie  = (0 :+ e)
    tau' = tau param
    gamma1 = (1 + alpha param)
    gamma2 = (1 - alpha param)

efun :: ModelParams -> Double -> Double
efun param e = (exp . sum) $ map (transform e) cnlist
  where
    transform e (n, cn) = cn * (1 - exparg n) * cosarg n * e
    exparg n = exp $ negate 2 * pi * fromInteger n / tau'
    cosarg n = cos $ 4 * fromInteger n / tau'
    cnlist = zip [1..n'] cn'
    cn' = tail $ c_n param
    n'  = n_cn param
    tau' = tau param

-- 7. Вычисление коэффициентов cN END
-----------------------------------------



-----------------------------------------
-- 8. Функция кривизны BEGIN
```

```
-- Определена как кривизна в точках, расположенных вдоль кривой.
-- Т. о., параметризована длиной кривой, и имеет один аргумент.
-- получает параметр указывающий на точку на криволинейной дуге,
-- выдаёт значение кривизны в этой точке
curvature :: ModelParams -> Double -> Double
curvature param x = ((1 - epssin) ** (3/2)) / p
  where epssin = (epsilon * (sin x)) ^ 2
        epsilon = ( sqrt ( b'*b' - a'*a' ) ) / b'
        p = (a'*a') / (b'*b')
        a' = a param
        b' = b param

-- 8. Функция кривизны END
----------------------------------------


----------------------------------------
-- 9. Операторы интегрирования BEGIN

-- Оператор интегрирования по ординате. То есть, интегрируем функцию комплексного переменного f(x + iy) по линии (0 + ia) .. (0 + ib).
--  Используется метод трапеций. Вроде как.
--  на вход получаем функцию комплексного аргумента, начальную ординату, конечную ординату и количество отрезков разбиения сетки.
integrateY :: (RealFloat a, Enum a) => (Complex a -> Complex a) -> a -> a -> Integer -> Complex a
integrateY f a b n =
    ((sum $ map f yvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      yvalues = map ((:+) x) values -- DO NOT REMOVE BRACKETS AROUND ':+'
      t = (f (x :+ a) + f (x :+ b))/2
      h = (b - a) / fromInteger(n)
      x = 0

-- Тот же самый integrateY, только интегрирование производится по ((pi/4) + ia) .. ((pi/4) + ib)
 finalIntegrate :: (RealFloat a, Enum a) => (Complex a -> Complex a) -> a -> a -> Integer -> Complex a
 finalIntegrate f a b n =
    ((sum $ map f yvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      yvalues = map ((:+) x) values -- DO NOT REMOVE BRACKETS AROUND ':+'
      t = (f (x :+ a) + f (x :+ b))/2
      h = (b - a) / fromInteger(n)
      x = pi / 4

integrateCD,integrateDA,integrateCB :: (RealFloat a, Enum a) => (Complex a -> Complex a) -> a -> a -> Integer -> Complex a
integrateCB f a b n =
    ((sum $ map f yvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      yvalues = map ((:+) x) values -- DO NOT REMOVE BRACKETS AROUND ':+'
      t = (f (x :+ a) + f (x :+ b))/2
      h = (b - a) / fromInteger(n)
      x = 0

```

```haskell
integrateBA :: (RealFloat a, Enum a) => a -> (Complex a -> Complex a) -> a -> a -> Integer -> Complex a
integrateBA tau' f a b n =
    ((sum $ map f xvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      xvalues = map (:+ y) values
      t = (f (a :+ y) + f (b :+ y))/2
      h = (b - a) / fromInteger(n)
      y = pi * tau' / 4

integrateCD f a b n =
    ((sum $ map f xvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      xvalues = map (:+ y) values
      t = (f (a :+ y) + f (b :+ y))/2
      h = (b - a) / fromInteger(n)
      y = 0

integrateDA f a b n =
    ((sum $ map f yvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      yvalues = map ((:+) x) values -- DO NOT REMOVE BRACKETS AROUND ':+'
      t = (f (x :+ a) + f (x :+ b))/2
      h = (b - a) / fromInteger(n)
      x = pi / 4

-- Оператор интегрирования по абсциссе. То есть, интегрируем функцию комплексного переменного f(x + iy) по линии (a + i0) .. (b + i0).
--  Используется метод трапеций. Вроде как.
--  на вход получаем функцию комплексного аргумента, начальную абсциссу, конечную абсциссу и количество отрезков разбиения сетки.
integrateX :: (RealFloat a, Enum a) => (Complex a -> Complex a) -> a -> a -> Integer -> Complex a
integrateX f a b n =
    ((sum $ map f xvalues) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      xvalues = map (:+ 0) values
      t = (f (a :+ 0) + f (b :+ 0))/2
      h = (b - a) / fromInteger(n)

 integrateYreal :: (Double -> Complex Double) -> Double -> Double -> Integer -> Complex Double
 integrateYreal f a b n =
    ((sum $ map f values) + t) * (h :+ 0)
    where
      values = [a + h * fromInteger(nn) | nn <- [0..n]]
      t = (f a + f b) / (2 :+ 0)
      h = (b - a) / fromInteger(n)
      x = 0

-- Оператор интегрирования действительнозначных функций методом трапеций.
-- Получает на вход функцию, нижний предел интегрирования, верхний предел интегрирования и количество отрезков разбиения сетки.
integrate f a b n =
    ((sum $ map f values) + t) * h
    where
```

```
        values = [a + h * fromInteger(nn) | nn <- [0..(n-1)]]
        t = (f a + f b)/2
        h = (b - a) / fromInteger(n)

-- 9. Операторы интегрирования END
----------------------------------------
```

B Theta.hs

```
module Theta where
-- Реализация тета-функций на основе тригонометрических рядов.
-- Тета-функции зависят от параметра tau, мнимая часть которого должна быть положительна.
-- Тау используется только в параметре 'q' в самих функциях, так что можно считать, что математически тета-функции зависят от кью, но технически работа этого модуля зависит от наличия тау.
-- Любую функцию надо вызывать через thetaN <n> (qrap <tau>) <u>

import Control.Parallel
import Control.Parallel.Strategies

-- Мы работаем с комплексными числами
import Data.Complex

-- Параметр q из математического представления тета-функций.
qpar tau = exp $ pi * tau * (0 :+ 1)

-- Функция, изображающая из себя (-1)^n
signfun :: (RealFloat a) => Integer -> Complex a
signfun nn
  | odd nn = -1
  | otherwise = 1

-- Функция \Theta_1
theta1 :: (RealFloat a) => Integer -> Complex a -> Complex a -> Complex a
theta1 n q u = 2 * sum thetaarg
   where thetaarg = [(signfun nn) * (qfun q nn) * (sinfun u nn) | nn <- [1..n]]
         qfun :: (RealFloat a) => Complex a -> Integer -> Complex a
         qfun q nn = q ** (0.5 + fromInteger nn) ** 2
         sinfun :: (RealFloat a) => Complex a -> Integer -> Complex a
         sinfun u nn = sin $ fromInteger(2 * nn + 1) * u

-- Функция \Theta_2
theta2 :: (RealFloat a) => Integer -> Complex a -> Complex a -> Complex a
theta2 n q u = 2 * sum thetaarg
  where thetaarg = [(qfun q nn) * (cosfun u nn) | nn <- [1..n]]
        qfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        qfun q nn = q ** (0.5 + fromInteger nn) ** 2
        cosfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        cosfun u nn = cos $ fromInteger(2 * nn + 1) * u

-- Функция \Theta_3
theta3 :: (RealFloat a) => Integer -> Complex a -> Complex a -> Complex a
theta3 n q u = 1 + 2 * sum thetaarg
  where thetaarg = [(qfun q nn) * (cosfun u nn) | nn <- [1..n]]
        qfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        qfun q nn = q ** (fromInteger nn) ** 2
        cosfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        cosfun u nn = cos $ fromInteger (2 * nn) * u

```

```
-- Функция \Theta_4
theta4 :: (RealFloat a) => Integer -> Complex a -> Complex a -> Complex a
theta4 n q u = 1 + 2 * sum thetaarg
  where thetaarg = [(signfun nn) * (qfun q nn) * (cosfun u nn) | nn <- [1..n]]
        qfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        qfun q nn = q ** fromInteger(nn) ** 2
        cosfun :: (RealFloat a) => Complex a -> Integer -> Complex a
        cosfun u nn = cos $ fromInteger(2 * nn) * u
```

C Main.hs

```
module Main where

-- Скорее всего, здесь не понадобится этот модуль, потому что работа с комплексными числами будет инкапсулирована в модуле BlastModel
--import Data.Complex

-- Импортируем собственно саму модель
import BlastModel

-- Модули, необходимые для поддержки черчения графиков функций
import Graphics.Rendering.Chart
import Graphics.Rendering.Chart.Gtk
import Data.Colour
import Data.Colour.Names
import Data.Accessor

-- Модуль для оценки времени выполнения той или иной функции. Самописный, на основе более низкоуровневых модулей
import Time

-- Точка входа программы. Программа делает последовательно ряд шагов и завершается.
main = do
-- 1. Печатаем приглашение и описание того, кто мы такие вообще.
    printGreeting
-- 2. Загружаем параметры тем или иным способом
  param <- getParameters
-- 3. Печатаем параметры, с которыми в итоге работаем.
  printParameters param
-- 4. Обновляем в параметрах список коэффициентов cN ``c_n param``
-- 4.5. Выводим данные оценки времени выполнения
  new_param <- renewCoeffs param
-- 5. Вычисляем список точек на границе воронки взрыва
-- 5.5. Выводим данные оценки времени выполнения
   pointlist <- calcPoints new_param
  let linetitle = extract_param_names new_param
-- 6. Передаём список точек функции черчения графика, которая чертит график
  plotFullGraph linetitle pointlist
-- 7. Пишем, что всё прошло успешно, так что завершаемся
  printGoodbye
-- Точка входа программы END

----------------------------------------
-- 1. Печатаем приглашение и описание того, кто мы такие вообще. BEGIN
printGreeting :: IO ()
printGreeting = do
  putStrLn "Greetings!~ This is blast model, based on solid-liquid model of Lavrentyev and Kotlyar."
-- 1. Печатаем приглашение и описание того, кто мы такие вообще. END
----------------------------------------
```

```haskell

----------------------------------------------------------------------
-- 2. Загружаем параметры тем или иным способом. BEGIN

getParameters :: IO (ModelParams)
getParameters = do
  print "[1]␣---␣load␣parameters␣manually"
  print "[anykey]␣---␣load␣defaults"
  print "Parameters␣n_integral,␣n_cn,␣n_theta␣stay␣constant"
  what <- getLine
  if what == "1"
    then do print "Ok,␣will␣load␣parameters␣from␣you"
            getModelParameters
    else do print "Ok,␣will␣load␣default␣parameters"
            return null_parameters

getModelParameter :: String -> IO(Double)
getModelParameter parname = do
        putStr $ "Value␣of␣" ++ parname
        param <- getLine
        return $ read param

getModelParameters :: IO (ModelParams)
getModelParameters = do
  print      "Value␣of␣phi_0:"
  phi_0'       <- getLine
  print      "Value␣of␣v_0:"
  v_0'         <- getLine
  print      "Value␣of␣|tau|:"
  tau'         <- getLine
  print      "Value␣of␣alpha:"
  alpha'       <- getLine
  print      "Value␣of␣a:"
  a'           <- getLine
  print      "Value␣of␣b:"
  b'           <- getLine
  print      "Precision:"
  precision'   <- getLine
  -- Параметры точности константные, потому что.
  -- n_theta'    <- getModelParameter "n_theta"
  -- n_integral' <- getModelParameter "n_integral"
  -- n_cn'       <- getModelParameter "n_cn"
  print       "Filler␣for␣c_n:"
  fill_cn      <- getLine
  return ModelParams{
       phi_0      = read phi_0',
       v_0        = read v_0',
       tau        = read tau',
       alpha      = read alpha',
       a          = read a',
       b          = read b',
       precision  = read precision',
       -- Берём параметры точности из параметров по умолчанию
       n_theta    = n_theta    null_parameters,
       n_integral = n_integral null_parameters,
       n_cn       = n_cn       null_parameters,
       -- Начальные значения коэффициентов приравниваем к значению-заполнителю
       c_n        = replicate ((fromInteger . n_cn) null_parameters) (read fill_cn)
       }

```

```
-- 2. Загружаем параметры тем или иным способом. END
-----------------------------------------

-----------------------------------------
-- 3. Печатаем параметры, с которыми в итоге работаем. BEGIN
printParameters :: ModelParams -> IO ()
printParameters param = do
-- TODO: Напиши меня!
  print "Current model parameters: "
  print param
-- 3. Печатаем параметры, с которыми в итоге работаем. END
-----------------------------------------

-----------------------------------------
-- 4. Обновляем в параметрах список коэффициентов cN ``c_n param``. BEGIN
renewCoeffs :: ModelParams -> IO (ModelParams)
renewCoeffs param = do
  putStrLn "Second, we compute the parameters c_n needed for computations"
-- 4.5. Выводим данные оценки времени выполнения
  new_param <- time $ renew_cn_all param
  return new_param

-- 4. Обновляем в параметрах список коэффициентов cN ``c_n param``. END
-----------------------------------------

-----------------------------------------
-- 5. Вычисляем список точек на границе воронки взрыва. BEGIN
type PointList = [(Double, Double)]
type BAPointList = PointList
type CDPointList = PointList
type CBPointList = PointList
type DAPointList = PointList
type ZPlanePoints = (CBPointList, CDPointList, DAPointList, BAPointList)
calcPoints :: ModelParams -> IO (ZPlanePoints)
calcPoints param = do
  print "We will now compute points on the edge of blast."
  let da_pointlist = (init . tail) $ zlistDA param
      ba_pointlist = (init . tail) $ zlistBA param
      cb_pointlist = (init . tail) $ zlistCB param
      cd_pointlist = (init . tail) $ zlistCD param
  --5.5. TODO: Выводим данные оценки времени выполнения
  print "Points at DA: "
  time $ outputData da_pointlist
  print "Points at BA: "
  time $ outputData ba_pointlist
  print "Points at CD: "
  time $ outputData cd_pointlist
  print "Points at CB: "
  time $ outputData cb_pointlist
  return (cb_pointlist, cd_pointlist, da_pointlist, ba_pointlist)

outputData datalist = do
  mapM (\(x, y) -> putStrLn $ (show x) ++ "; " ++ (show y)) datalist
  return ()

-- 5. Вычисляем список точек на границе воронки взрыва. END
-----------------------------------------

-----------------------------------------
-- 6. Передаём список точек функции черчения графика, которая чертит график. BEGIN
```

```
extract_param_names param =
  let phi0    = show $ phi_0 param
      v0      = show $ v_0 param
      alpha0  = show $ alpha param
      a0      = show $ a param
      b0      = show $ b param
      abstau  = show $ tau param
      precision0  = show $ precision param
  in "Phi0␣=␣" ++ phi0 ++ ",␣V0␣=␣" ++ v0 ++ ",␣Alpha0␣=␣" ++ alpha0 ++ ",␣A␣=␣" ++ a0 ++ ",␣B␣=␣" ++ b0 ++ ",␣tau␣=␣" ++ abstau ++ ",␣precision␣=␣" ++ precision0

plotGraph :: String -> PointList -> IO()
plotGraph linetitle datalist = do
    renderableToWindow (toRenderable (chart linetitle datalist)) 640 480
    renderableToPNGFile (toRenderable (chart linetitle datalist)) 640 480 (linetitle ++ ".png")
    return ()

plotFullGraph :: String -> ZPlanePoints -> IO()
plotFullGraph linetitle datalists = do
    renderableToWindow (toRenderable (manychart linetitle datalists)) 640 480
    renderableToPNGFile (toRenderable (manychart linetitle datalists)) 640 480 (linetitle ++ ".png")
    return ()

chart :: String -> PointList -> Layout1 Double Double
chart linetitle datalist = layout
  where
    myPlot = plot_lines_values ^= [datalist]
              $ plot_lines_style .> line_color ^= opaque blue
              $ plot_lines_title ^= linetitle
              $ defaultPlotLines
    layout = layout1_title ^= "Form␣of␣blast␣edge"
           $ layout1_plots ^= [Left (toPlot myPlot)]
           $ defaultLayout1

manychart :: String -> ZPlanePoints -> Layout1 Double Double
manychart linetitle (pointsCB, pointsCD, pointsBA, pointsDA) = layout
  where
    plotBA = plot_lines_values ^= [pointsBA]
              $ plot_lines_style .> line_color ^= opaque green
              $ plot_lines_title ^= "BA"
              $ defaultPlotLines
    plotDA = plot_lines_values ^= [pointsDA]
              $ plot_lines_style .> line_color ^= opaque red
              $ plot_lines_title ^= "DA"
              $ defaultPlotLines
    plotCB = plot_lines_values ^= [pointsCB]
              $ plot_lines_style .> line_color ^= opaque blue
              $ plot_lines_title ^= "CB:␣" ++ linetitle
              $ defaultPlotLines
    plotCD = plot_lines_values ^= [pointsCD]
              $ plot_lines_style .> line_color ^= opaque cyan
              $ plot_lines_title ^= "CD"
              $ defaultPlotLines
    layout = layout1_title ^= "Form␣of␣blast␣edge"
           $ layout1_plots ^= [
             Left (toPlot plotBA),
             Left (toPlot plotDA),
             -- Left (toPlot plotCD),
```

```haskell
               Left (toPlot plotCB)
               ]
             $ defaultLayout1

-- Пример оформления чертежа
-- -- chart = layout
-- -- where
-- --   am :: Double -> Double
-- --   am x = (sin (x*3.14159/45) + 1) / 2 * (sin (x*3.14159/5))
-- --   sinusoid1 = plot_lines_values ^= [[ (x,(am x)) | x <- [0,(0.5) ..400]]]
-- --               $ plot_lines_style .> line_color ^= opaque blue
-- --               $ plot_lines_title ^= "am"
-- --               $ defaultPlotLines
-- --   sinusoid2 = plot_points_style ^= filledCircles 2 (opaque red)
-- --               $ plot_points_values ^= [ (x,(am x)) | x <- [0,7..400]]
-- --               $ plot_points_title ^= "am points"
-- --               $ defaultPlotPoints
-- --   layout = layout1_title ^= "Amplitude Modulation"
-- --            $ layout1_plots ^= [Left (toPlot sinusoid1),
-- --                                Left (toPlot sinusoid2)]
-- --            $ defaultLayout1

-- 6. Передаём список точек функции черчения графика, которая чертит график. END
------------------------------------------

------------------------------------------
-- 7. Пишем, что всё прошло успешно, так что завершаемся. BEGIN
printGoodbye :: IO ()

printGoodbye = do
  putStrLn "All done, good bye."
-- 7. Пишем, что всё прошло успешно, так что завершаемся. END
------------------------------------------


-- Автоматизированные тесты для некоторого покрытия поля параметров модели
silentlyPlotGraph :: ModelParams -> IO()
silentlyPlotGraph param = do
  new_param <- renewCoeffs param
   pointlist <- calcPoints new_param
  let linetitle = extract_param_names new_param
  renderableToPNGFile (toRenderable (manychart linetitle pointlist )) 640 480 ( linetitle ++ ".png")
  return ()

testPhi0 :: ModelParams -> IO()
testPhi0 param = do
  let plotWithPhi0 p = silentlyPlotGraph param{phi_0 = (p * 0.1)}
  mapM plotWithPhi0 [0..20]
  return ()

testV0 :: ModelParams -> IO()
testV0 param = do
  let plotWithV0 p = silentlyPlotGraph param{v_0 = (p * 15)}
  mapM plotWithV0 [0..10]
  return ()

testTau :: ModelParams -> IO()
testTau param = do
  let plotWithTau p = silentlyPlotGraph param{tau = (p * 0.1)}
  mapM plotWithTau [1..10]
```

```
  return ()

testAlpha :: ModelParams -> IO()
testAlpha param = do
  let plotWithAlpha p = silentlyPlotGraph param{alpha = (p * 0.1)}
  mapM plotWithAlpha [0..10]
  return ()
```