МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

**Кафедра математических и компьютерных методов**

**КУРСОВАЯ РАБОТА**

**РЕШЕНИЕ ВНУТРЕННЕЙ ЗАДАЧИ ДИРИХЛЕ ДЛЯ УРАВНЕНИЯ ЛАПЛАСА В ОДНОСВЯЗНОЙ ОБЛАСТИ МЕТОДОМ БАЗИСНЫХ ПОТЕНЦИАЛОВ**

Работу выполнил Д. А. Пасько

(подпись, дата)

Факультет математики и компьютерных наук курс 3

Направление 02.03.01 математика и компьютерные науки

Научный руководитель доцент кафедры МиКМ,

канд. физ.-мат. наук А. А. Свидлов

(подпись, дата)

Нормоконтролер

преподаватель А. А. Цыбенко

(подпись, дата)

Краснодар 2017

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc496003394)

[1 Необходимые понятия и теоретические выкладки 5](#_Toc496003395)

[1.1 Определение задачи 5](#_Toc496003396)

[1.2 Суть метода базисных потенциалов 6](#_Toc496003397)

[1.3 Сведение метода базисных потенциалов к комплексу простых задач 7](#_Toc496003398)

[2 Анализ и проектирование программы. Описание главных действующих алгоритмов 11](#_Toc496003399)

[2.1 Чтение данных и работа с данными 11](#_Toc496003400)

[2.2 Поиск решения 13](#_Toc496003401)

[2.2.1 Нахождение криволинейного интеграла первого рода 13](#_Toc496003402)

[2.2.2 Нахождение решений СЛАУ с симметрической матрицей 14](#_Toc496003403)

[2.2.3 Вывод точности 15](#_Toc496003404)

[2.3 Иллюстрация решения 16](#_Toc496003405)

[2.4 Псевдокод программы 17](#_Toc496003406)

[2.5 Сложность алгоритма 17](#_Toc496003407)

[3 Реализация программы 19](#_Toc496003408)

[3.1 Тестовые области 19](#_Toc496003409)

[3.1.1 Параметризация границы круга 19](#_Toc496003410)

[3.1.2 Параметризация треугольника 20](#_Toc496003411)

[3.1.3 Параметризация границы квадрата 21](#_Toc496003412)

[3.2 Тестовые граничные функции 22](#_Toc496003413)

[3.3 Описание используемых классов и структур, глобальных переменных, некоторых функций и процедур 23](#_Toc496003414)

# Введение

Большое количество явлений современного мира люди уже умеют описывать математическими моделями, при этом оказывается, что немалая часть таких явлений сводится к краевым задачам всего нескольких категорий. Многие математические модели реальных физических процессов представляют из себя краевые задачи, в которых используются гармонические операторы; наиболее часто это верно для задач гидродинамики и квантовой физики, гармонические операторы присутствуют в уравнениях Гельмгольца, Шредингера, в уравнениях, описывающих стационарные процессы. Решение таких задач можно строить разными методами, однако наиболее продуктивными из этих методов являются проекционные, приближённо выражающие решение задачи через проекцию на какое-то пространство функций: такие решения удобнее затем использовать в прикладных исследованиях изучаемых процессов. Относительно недавно класс проекционных методов пополнился методом базисных потенциалов, который сегодня активно модифицируется. Одними из преимуществ этого метода являются: высокая точность при небольшом числе базисных функций и возможность его приемлемой реализации на ЭВМ даже с сильно ограниченными системными ресурсами.

***Постановка задачи***. Описать метод базисных (точечных) потенциалов и написать программу, которая находит решение задачи Дирихле для уравнения Лапласа этим методом, показывает точность найденного решения и иллюстрирует его.

# 1 Необходимые понятия и теоретические выкладки

## 1.1 Определение задачи

Пусть на некотором множестве пространства определена дважды непрерывно дифференцируемая функция . В таком случае оператор Лапласа устанавливается действием на эту функцию следующим образом:

*.*

Тогда задача нахождения ядра такого оператора сводится к решению уравнения Лапласа

,

которое, как известно, имеет целый класс решений, называемых гармоническими функциями; и чтобы выделить из множества решений какое-то одно, требуется наложить дополнительные условия. Как видно, уравнение Лапласа относится к уравнениям эллиптического типа и описывает стационарный процесс, а потому для него могут быть определены только граничные условия; при этом задача нахождения решения уравнения Лапласа при граничных условиях первого рода, то есть вида

,

где – граница области определения функции, называется задачей Дирихле для уравнения Лапласа.

В нашем случае требуется методом базисных потенциалов найти решение задачи Дирихле

,

где множество – плоская односвязная область с кусочно-гладкой границей.

## 1.2 Суть метода базисных потенциалов

Как было сказано, функции, удовлетворяющие уравнению Лапласа, называются гармоническими; решение задачи Дирихле заключается в нахождении гармонической функции, которая на множестве точек границы исходной области совпадает с заданной функцией . Метод базисных потенциалов заключается в следующем: 1) в дополнении исходной области фиксируется последовательность точек, 2) каждой точке ставится в соответствие функция, являющаяся фундаментальным решением уравнения Лапласа, то есть гармоническая функция, 3) граничная функция аппроксимируется линейной комбинацией соответствующих функций; таким образом мы находим приближённое решение задачи Дирихле, которое удовлетворяет уравнению Лапласа и с некоторой степенью точности отличается от граничной функции, причём точность зависит как от самих выбранных точек, так и от их количества (ведь при вычислениях нельзя работать с бесконечными последовательностями). Теперь наша задача заключается в определении необходимых понятий и в теоретическом обосновании действенности описанного метода.

Ограниченная последовательность точек будет называться ***базисной***, если она принадлежит области и удовлетворяет условию единственности гармонических функций (то есть любые две гармонические функции, совпадающие на этих точках, совпадают тождественно). В таком случае рассмотрим определённую на систему функций

которую назовём ***системой базисных потенциалов***. Эта система является линейно независимой и замкнутой в [6], а поскольку само пространство  является сепарабельным и гильбертовым, то замкнутость в нём эквивалентна полноте [8]. Поскольку система базисных потенциалов является полной и линейно независимой, то всякую функцию из пространства  можно с любой степенью точности аппроксимировать линейной комбинацией конечного числа базисных потенциалов, то есть

Зная это, мы можем свести решение задачи Дирихле для уравнения Лапласа к нахождению приближенного решения по системе базисных потенциалов: из наперёд заданных условий и заданных точек требуется найти такую комбинацию коэффициентов чтобы функция приближалась максимально возможно, то есть

Иначе говоря, мы будем искать проекцию функции на линейную оболочку конечного числа функций ; таким образом, мы максимально приблизим граничную функцию то, что найденная приближённая функция будет удовлетворять собственно уравнению Лапласа, следует из того, что она окажется линейной комбинацией фундаментальных решений этого уравнения.

Теперь же требуется свести описанную математическую задачу к системе наиболее простых, дабы начать создавать программу, её решающую.

## 1.3 Сведение метода базисных потенциалов к комплексу простых задач

Надо найти вектор из предыдущего пункта. Так как пространство – гильбертово, то норма в нём выражается через скалярное произведение:

Так как норма является неотрицательной по определению, то предыдущее выражение можно возвести в квадрат без риска потерять решения или приобрести новые:

причём возникшая матрица Грама является невырожденной ввиду линейной независимости системы базисных потенциалов. Наша задача превратилась в минимизацию полученного квадратичного функционала.

Вычислим стационарные точки этого функционала из системы:

где

Такая система сводится к виду

,

где , – матрица Грама нашей системы. Так как эта матрица является невырожденной, система имеет единственное решение; поскольку второй дифференциал функции является положительно определённой квадратичной формой с той же матрицей Грама, то в точке решения нашей системы функция действительно имеет минимум.

Таким образом, мы свели решение задачи Дирихле для уравнения Лапласа к нахождению приближённого решения , где вектор коэффициентов является решением системы

в которой

Осталось написать программу, которая будет оптимально решать исходную систему линейных алгебраических уравнений, выводить точность найденного решения задачи и создавать графическую иллюстрацию. Дополнительно придётся находить значения криволинейных интегралов первого рода и делать программу универсальной, то есть не нуждающейся в серьёзных изменениях при смене граничной функции или области .

# 2 Анализ и проектирование программы. Описание главных действующих алгоритмов

Программа нахождения решения задачи Дирихле для уравнения теплопроводности пишется на языке С++ и состоит из следующих ***этапов***:

1) ***Чтение данных и работа с этими данными***. На этом этапе выбирается вариант граничной функции , граница области  и считываются координаты точек плоскости (базисных точек), затем из считанного множества точек плоскости удаляются повторяющиеся элементы в случае их наличия; предполагается, что все точки принадлежат дополнению области .

2) ***Поиск решения задачи***. Каждой точке ставится в соответствие функция потенциала, считаются элементы матрицы Грама, решается система линейных алгебраических уравнений с целью нахождения коэффициентов проекции функции на линейную комбинацию базисных потенциалов, после этого считается и выводится точность полученного решения.

3) ***Иллюстрация решения***. При помощи графических возможностей языка С++ найденная приближённая функция изображается на рисунке вместе с заранее известным точным решением задачи Дирихле.

Далее проводится подробное описание каждого из этапов.

## 2.1 Чтение данных и работа с данными

Прежде чем описывать процесс чтения данных (в основном точек), естественно определить некоторую структуру, которая содержит всю информацию, связанную с этими точками, включая и функцию потенциала, соответствующую выбранной точке. Пусть изначально такая структура будет иметь вид:

struct basp{

double x;//координаты точки в двумерном пространстве

double y;

//функция базисного потенциала от точки z, соответствующая исходной базисной точке

double potfunc(basp z){

return log(1/sqrt((z.x-x)\*(z.x-x)+(z.y-y)\*(z.y-y)));

}

//расстояние между точками

double eudistance(basp z,basp w){

return sqrt((z.x-w.x)\*(z.x-w.x)+(z.y-w.y)\*(z.y-w.y));

}

}

***Чтение***. С текстового файла in.txt считываются строки пояснений и все числовые значения, которые в нём есть; первые два числовых значения – соответствующие номера для области и граничной функции (последние определены в самом коде), следующие числа – координаты базисных точек; пусть количество числовых значений , тогда результатом целочисленного деления выясняется мощность  множества точек плоскости, то есть количество задаваемых точек; далее эти точки снова считываются и сохраняются в массив уже определённых структур basp.

Далее производится ***проверка******на совпадение введённых точек*** с целью убрать возможную линейную зависимость потенциальных функций; если же этого не сделать, наша задача и вовсе не обязана будет иметь решений, покуда задача Дирихле имеет единственное решение. Алгоритм проверки на совпадение точек таков: точки сортируются по первой координате, потом полученный массив разбивается на участки, где имеются точки с одинаковой первой координатой, далее каждый такой участок сортируется по второй координате, а затем все точки пробегаются; если у соседних точек обе координаты равны, одна из точек удаляется, а пробег продолжается, покуда все точки не будут пройдены. В конечном итоге получается окончательный массив точек плоскости; если мощность исходного множества уменьшится, это зафиксируется. Для сортировки используется библиотечная функция sort()с наиболее оптимальной сложностью.

## 2.2 Поиск решения

Как мы установили, поиск приближённого решения задачи Дирихле для уравнения Лапласа заключается в решении системы линейных уравнений

.

При этом возникает дополнительная задача нахождения скалярного произведения функций из пространства , которое представляет из себя криволинейный интеграл первого рода по границе нашей области:

Составим алгоритмы для решения каждой из поставленных задач.

### 2.2.1 Нахождение криволинейного интеграла первого рода

Пусть граница задаётся кривой . В таком случае для определённой на этой кривой функции криволинейный интеграл первого рода может быть вычислен приближённо:

при достаточно большом.

Реализовать же такой метод можно, создав класс функций, задающих кривые, причём основным методом этого класса должно быть возвращение точки по заданному параметру. Выглядеть это будет примерно так:

class curve {//класс кривых

private:

double a;//начальное значение параметра

double b;//конечное значение параметра

public:

double h=(b-a)/M;//значение шага для этой кривой

basp transfer(double t){//возврат точки на кривой по значению параметра

basp point;

point.x=3\*cos(t);

point.y=sin(t);

return point;

}

}

### 2.2.2 Нахождение решений СЛАУ с симметрической матрицей

Так как матрица Грама является симметрической, то для решения уравнения разумнее пользоваться методами, показывающими на симметрических матрицах б**о**льшую экономичность и точность в сравнении с общими методами. К одному из них относится метод квадратного корня (метод Холецкого), который заключается в следующем. Всякую симметрическую матрицу  можно разложить на произведение двух взаимно транспонированных треугольных матриц:

,

где

В таком случае после перемножения матриц выводятся формулы для коэффициентов матрицы через коэффициенты матрицы :

Поскольку , то система сводится к виду:

.

Решая эту систему, мы придём к рекуррентным отношениям:

### 2.2.3 Вывод точности

После нахождения решения в виде , где – известные базисные потенциалы, – найденные константы, **в выходной файл для показания точности аппроксимации будет выведено расстояние между приближённым решением и функцией** , равное

## 2.3 Иллюстрация решения

С помощью библиотеки “Graph.h” в трёхмерном пространстве рисуется график исходной граничной функции и найденной приближённой функции. Чтобы рисование графика было удобным и автоматизированным, необходимо приближённую функцию рассматривать как метод некоторого класса, который является наследственным по отношению к описанным ранее.

## 2.4 Псевдокод программы

Покажем алгоритм программы (без иллюстрирования) нагляднее в следующем псевдокоде:

//основная функция

Функция main()

{

**чтение данных и работа с данными()**{}

**поиск решения()**{}

}

//внутренние функции и процедуры основной функции

**Чтение данных и работа с данными**()

{

чтение номера области;

чтение номера граничной функции;

проверка на ошибку при вводе номеров области и функции, в случае ошибки выводится сообщение о ней и программа прекращает работу;

вычисление мощности множества базисных точек;

чтение базисных точек и заполнение массива этих точек;

сортировка массива точек;

удаление из массива повторяющихся точек;

}

**Поиск решения**()

{

создание системы линейных алгебраических уравнений с размерностью, равной размерности массива базисных точек после удаления из них повторяющихся;

заполнение элементов системы соответствующими скалярными произведениями функций;

решение системы методом Холецкого;

нахождение точности и вывод;

}

## 2.5 Сложность алгоритма

Пусть – количество считанных точек. Тогда сортировка этих точек и отсеивание повторяющихся занимает операций. В результате мы получаем массив неповторяющихся базисных точек размерности . Далее происходит заполнение системы линейных уравнений, для чего требуется найти криволинейных интегралов; вычисление каждого интеграла через  шагов будет занимать  операций; следовательно, заполнение системы стоит нам  операций. Решение этой системы методом Холецкого будет занимать  операций. В таком случае общая сложность алгоритма равняется:

# 3 Реализация программы

Создание необходимых классов представило большую сложность ввиду требований знания языка на высоком уровне. Здесь будут описаны основные классы созданной программы, а также области и граничные функции, на которых проводится тестирование. Затем будет рассказано о результатах тестирования при разных областях, разных граничных функциях, разном числе и расположении базисных точек.

## 3.1 Тестовые области

В качестве тестовых областей будут использоваться простейшие: круг, внутренность треугольника, внутренность квадрата. Несмотря на их простоту, не так просто задать параметризацию границ этих областей так, чтобы ей можно было пользоваться в программе. Но придётся это сделать.

### 3.1.1 Параметризация границы круга

Как известно, параметризация окружности с центром в начале координат и радиусом  может выглядеть так:

В таком случае круг окружность 3 будет как на рисунке 1.1.

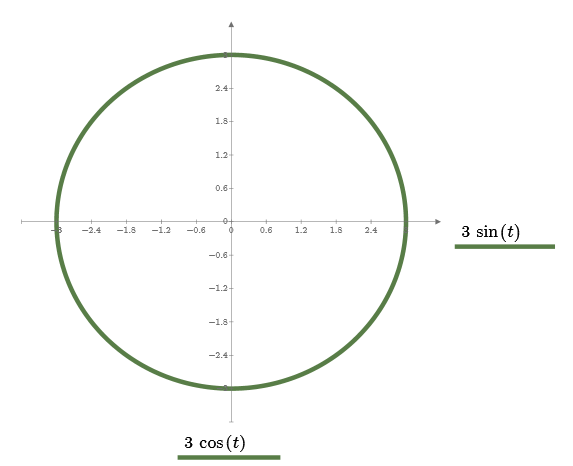


Рисунок .1 Выполненный в Mathcad Prime график окружности с центром в начале координат и радиусом 3, параметризованной известным образом.

### 3.1.2 Параметризация треугольника

Пусть нам нужно параметризовать равносторонний треугольник со стороной 4. Удобнее будет, если одна из его вершин совпадает с началом координат, а одна из сторон лежит на оси координат. Например, подойдёт такая параметризация треугольника с вершинами в точках , , , которую можно вывести с помощью известного уравнения прямой, проходящей через две точки[[1]](#footnote-1):

В результате получим треугольник, изображённый на рисунке 1.2.

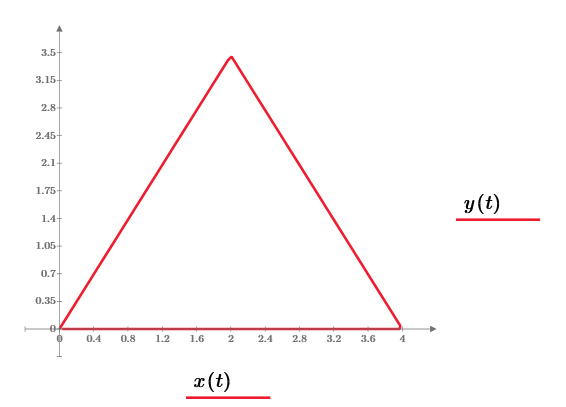


Рисунок 1.2. Выполненный в Mathcad Prime график равностороннего треугольника со стороной 4, параметризованного известным образом.

### 3.1.3 Параметризация границы квадрата

По полной аналогии с предыдущим пунктом зададим параметризацию квадрата:

Результат изображён на рисунке 1.3.

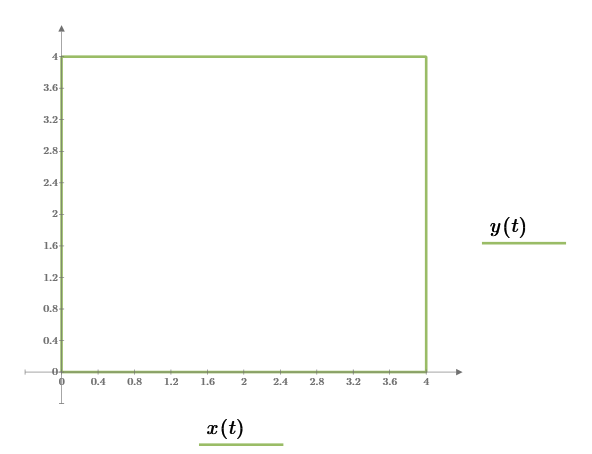


Рисунок 1.3. Выполненный в Mathcad Prime график квадрата со стороной 4, параметризованного известным образом.

## 3.2 Тестовые граничные функции

Наша внутренняя задача Дирихле

разрешима единственным образом в тех случаях, когда [7]. В качестве граничных функций возьмём следующие:

## 3.3 Описание используемых классов и структур, глобальных переменных, некоторых функций и процедур

Согласно ***Бьярне Страуструпу***, “цель программиста – выразить вычисления, причём это должно быть сделано **1)** правильно, **2)** просто, **3)** эффективно”[[2]](#footnote-2).

Именно поэтому код программы должен быть снабжён множественными ***комментариями***, переменные должны быть названы ***удобными именами***, а сама программа, желательно, должна быть разбита на большое число отдельных ***функций*** и ***процедур***, чтобы легче было проводить её отладку, и читать, и понимать; если же программа должна решать крупные задачи и не требовать крупных изменений для решения аналогичных задач, в ней не обойтись без ***классов***. Если общий алгоритм программы требовал пояснений, то отдельные функции в них не нуждаются; тем не менее, общий алгоритм представляет собой взаимодействие множества небольших алгоритмов, каждый из которых требует определённого количества времени и системных ресурсов, а потому оптимизация каждого из них – дело существенное.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// пока КОНЕЦ

В программе определена **структура** «точка» с полями координат точки на плоскости:

struct Point {

double x;

double y;

};

Перечислим **глобальные переменные**:

int m – мощность множества точек плоскости, которая вычисляется по формуле , где – количество чисел (координат) в файле input.txt;

int kt – мощность множества РАЗЛИЧНЫХ точек плоскости, то есть множества точек после сортировки и отсеивания повторяющихся; очевидно, что ;

Point z1[3], z2[3] – два массива структур Point, в которых в случае своего существования будут располагаться вершины найденных треугольников (внешнего и внутреннего соответственно);

Point \*mas – указатель на массив (всех) точек плоскости; указатель требуется потому, что неизвестна заранее размерность этого массива;

Point \*internalpoints – указатель на вспомогательный массив "внутренних точек", где для каждого найденного «внешнего» треугольника будут располагаться его внутренние точки в случае их существования;

double a0, b0, c0, d0 – пределы окна, в котором располагается иллюстрация решения;

const double eps = 0.01 – погрешность, необходимая в некоторых местах программы; погрешность используется при проверки достаточного условия, поскольку из-за ошибок округления сумма площадей трёх вписанных треугольников не всегда точно совпадает с площадью треугольника, распадающегося на три вписанных.

Перечислим некоторые **функции и процедуры**:

Функция трёх аргументов double P(double a, double b, double c) сопоставляет трём заданным числам типа double площадь треугольника, построенного на сторонах с длинами, равными этим числам соответственно. Функция выводит значение площади типа double. Площадь треугольника считается по формуле Герона; предполагается, что такой треугольник существует, и функция в программе используется уже тогда, когда известно достоверно, что треугольник существует; если же отрезки с заданными сторонами не образуют треугольник, функция выведет нулевое значение.

Процедура void exceptionmas(Point \*mas) занимается отсеиванием повторяющихся элементов из массива структур mas, указатель на первый элемент которого является её аргументом. Это делается следующим образом: если два соседних элемента в массиве (массив отсортирован) совпадают, вызывается процедура void excep(int i, Point \*a), которая удаляет из массива структур a элемент с индексом i, делая единичный сдвиг всех последующих элементов влево.

Функция bool triangleanswer(Point mas1, Point mas2, Point mas3) выясняет, составляют ли исходные три точки (аргументы типа Point) треугольник; для этого функция находит длины всех сторон между этими точками, а затем проверяет неравенства треугольника, выдавая в итоге логическое значение, совпадающее со значением логической формулы неравенств треугольника.

Функция bool triangleanswer(Point \*a, int k) выясняет, может ли какая-либо тройка точек массива a размерности k образовать треугольник, перебирая для этого все тройки и вызывая функцию bool triangleanswer(Point mas1, Point mas2, Point mas3); если какой-нибудь треугольник найден, функция не только выведет логическое значение, но и запишет координаты найденного треугольника в соответствующий массив.

## 3.2 Ошибки, с которыми я столкнулся при отладке первых версий программы:

1) Программа верно считывала координаты точек при первом прочтении файла, но почти ничего не считывала при втором прочтении.

2) Сортировка массива структур через функцию sortстандартной библиотеки не заработала. Пришлось долго думать над ней, но в итоге всё получилось и код программы стал короче.

3) При отсеивании исчезали почти все точки.

4) Программа не находила решение даже там, где оно имелось. Проблема заключалась в неверном присваивании.

5) Программа не находила решения, потому что не видела никаких треугольников (не тот знак поставил в неравенствах треугольника).

6) Программа не находила всех внутренних точек (возникла проблема в достаточном признаке).

7) Программа не считывала числа с плавающей точкой из-за того, что переменная, в которую вписывалось значение, была целочисленной.

8) Вместо отрезков программа рисовала какие-то колебания, поскольку произошло совпадение имён.

## 3.3 Предложения по оптимизации исходного алгоритма:

1) В исходном коде все функции и процедуры используют строго определённые типы переменных; программа решает поставленную задачу, однако в изначальном виде не годится для использования при решении сходной задачи. Эту проблему можно исправить при помощи шаблонов, чтобы используемые в программе функции и процедуры не ограничивались лишь этой программой.

2) Отсеивание из массива одинаковых точек можно провести быстрее, если не делать сдвиги при каждом повторяющемся элементе, но записывать нужные элементы в новый массив. Впрочем, это изменение не будет существенным, поскольку из специфики поставленной задачи следует, что задача будет решаться при небольшом количестве исходных точек.

3) Для экономии памяти можно уничтожать массивы и переменные, которые больше не пригодятся.

## 3.4 Выводы для оптимальной затраты рабочего времени на написание программы и отладку:

1) «Организовать программное обеспечение нужно так. чтобы минимизировать количество ошибок.» [1] Большинство ошибок, с которыми я столкнулся при написании программы, имели место из-за невнимательности. Но не были грубыми и не касались неверности в алгоритмах, но не давали программе работать правильно иногда уже в самом начале. Вывод: **при написании программы необходимо быть внимательным в отношении каждой процедуры, каждого объявления переменной, каждого цикла: не следует делать что-то интуитивно, но надо достоверно знать, что те или иные объекты и конструкции нужны тебе именно в таком виде, в каком ты собираешься их записать**.

2) При написании программы **нужно точно знать, чем именно ты занимаешься**: плохое понимание поставленной задачи или отдельных элементов её решения приводит к ограниченности программы и её неправильной работы с теми или иными входными данными.

3) Уже при написании программы я начал забывать, что делают и что значат те или иные функции и переменные, а ведь окончание программы ещё не намечалось; путём проб и ошибок я пришёл к следующему выводу: **не следует рассчитывать на свою память, но программу нужно снабжать хорошими комментариями; «в комментариях следует ясно и коротко указать то, что невозможно выразить в коде» [1]; сюда же относится рекомендация использовать осмысленные имена**.

4) **При написании программы её разбиение на небольшие фрагменты является обязательным требованием!** В противном случае код получится слишком громоздким, большим, его крайне тяжело будет отлаживать и исправлять.

5) **Где только можно, нужно использовать проверенные временем библиотечные функции**. Они существенно сэкономят время написания программы; они же являются наиболее оптимальными в общем случае, потому в важных случаях с ними программа работать будет быстрее, чем с их заменой.

# Заключение

Для детального решения задачи, которая даже молодому человеческому мозгу удаётся почти мгновенно, я потратил несколько дней, на создание полноценного алгоритма ушли полторы недели, реализация алгоритма (с учётом отладки) растянулась на два месяца. Потребовались немалые усилия, чтобы «научить» компьютер решению такой простой на вид задачи, а ведь немыслимо количество задач (и не простых), с которыми приходится сталкиваться, например, операционной системе, космическому кораблю, разным программам диагностики и моему смартфону каждую секунду! За всей нашей современной компьютерной цивилизацией стоит долгий труд миллионов людей, каждый из которых изучал один и тот же предмет практически с нуля, чтобы затем усовершенствовать достижения большинства программистов до него, словно он совершенствует единственную программу; и смысл этих дел в том, что людям всегда приходится учиться заново и всегда их жизнь конечна, а компьютерная система может совершенствоваться очень долго и всё это время решать за доли секунды такую массу задач, с которой десятку человек не справиться и за месяц; и может она это потому, что когда-то какие-то люди научили её этому. И многие наши вложения в прогресс этой системы дают результаты, которые будут оказывать помощь всем следующим поколениям; сегодня мы вкладываем силы в прогресс, а уже завтра компьютер сам сможет делать больше, тем самым освобождая нас от работы, которой раньше приходилось заниматься людям. Таким образом, работа над алгоритмами является частным и весьма актуальным сегодня случаем созидания, а созидание ещё с древних времён считается чертой развитого человека и также движет его развитие, что ещё важнее; и пока мы будем творить, мы будем людьми и мы будем ставиться лучше.

# Список использованных источников

1 Страуструп, Бьярне. Программирование: принципы и практика с использованием С++, 2-е изд.: Пер. с англ. - М.: ООО "И. Д. Вильяме", 2016. - 1328 с.: ил. - Парал. тит. англ.

2 Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. А95 Структуры данных и алгоритмы.: Пер. с англ.: Уч. пос. — М.: Издательский дом "Вильяме", 2000. — 384 с.: ил. — Парал. тит. англ.

3 Мейерс, Скотт. М45 Эффективный и современный С++: 42 рекомендации по использованию С++ 11и С++14.: Пер. с англ. - М.: ООО "ИЛ. Вильяме", 2016. - 304 с.: ил. – Пapал. тит. англ.

4 Васильев А. Н. Самоучитель С++ с примерами и задачами. – СПб.; Наука и Техника, 2010. – 480 с.: ил.

5 Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.

6 Метод базисных потенциалов в задачах математической физики и гидродинамики: монография / А. В. Лежнев, В. Г. Лежнев. – Краснодар: Кубанский гос. ун-т, 2009. – 111 с.

7 Задачи плоской гидродинамики: Учебное пособие. Краснодар: Кубанский государственный университет, 2000. 91 с.

8 Колмогоров А. Н., Фомин С. В. Элементы теории функции и функционального анализа. – М.: Наука. Гл. ред. физ.-мат. лит. 1989.

# Ссылки

http://phys.bspu.unibel.by/static/um/inf/vmm/pdf/vm2-02.pdf

1. [↑](#footnote-ref-1)
2. «Программирование: принципы и практика с использованием С++» [↑](#footnote-ref-2)