**Федеральное государственное автономное**

**образовательное учреждение высшего образования**

**«Казанский (Приволжский) Федеральный Университет»**

Институт вычислительной математики и информационных технологий

Кафедра теоретической кибернетики

**КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ ИНФОРМАТИКА**

**Тема:**

«Решение краевой задачи методом Монте-Карло.»

Выполнил: студент 2 курса группы 09-811   
 дневного отделения ИВМ и ИТ

Лапинский Василий Александрович

Научный руководитель:

Доцент кафедры теоретической кибернетики

Байрашева Венера Рустамовна

Казань 2019

**Оглавление**

1. Постановка задачи …………………………………………………2
2. Объяснение решения ………………………………………………4
3. Код программы……………………………………………………...8
4. Примеры работы программы…………………………………….....12
5. Вывод ……………………………………………………………......13
6. Список литературы………………………………………………….13

**Постановка задачи**

Плоскость (*x*, *y*) делится сеткой с шагом *h*, и рассматриваются только точки сетки. Задается область с границей *Г*. Точки границы обозначим через (*xi*, *y*i), *i* = 1, 2, … , *n*. В этих точках задана функция *f*(*x*, *y*). Краевая задача формулируется так: определить значение функции *u*(*x*, *y*) во внутренней точке области (*x\**, *y\**), которая удовлетворяет уравнению Лапласа Δ*u* = 0 и принимает на границе *Г* значения *u*(*xi*, *y*i) = *f*(*xi*, *y*i).

Задача решается методом Монте-Карло. Из точки (*x\**, *y\**) организуется *N* блужданий по внутренним точкам области, (точкам сетки) до выхода на граничную точку. Пусть ϕ*i* − количество блужданий с приходом в точку (*xi*, *y*i) границы *Г*. Тогда искомое значение



*Задание*. Составить программу для заданных *h*, *Г*, *f*(*x*, *y*) и «способа блуждания», которая вводит координаты точки (*x\**, *y\**) и число *N*, вычисляет *u*(*x\**, *y\**) и печатает это значение.

Граница Г имеет вид:



Из любой точки (*x*, *y*) области на шаге блуждания можно перейти в одну из восьми соседних точек (*x* + *h*, y), (*x* + *h*, *y* − *h*), (*x* + *h*, y + *h*), (*x* − *h*, y), (*x* − *h*, *y* − *h*), (*x* − *h*, *y* + *h*), (*x*, *y* − *h*), (*x*, *y* + *h*).

Так как цена деления на графике, на котором нам дана граница Г, равна 0.1 от единицы измерения, следовательно, в нашем случае h = 0.1.

Направление блуждания определяется при помощи следующей последовательности действий:

Определим массив T1, …, T12, элементами которого служат целые числа 0 или 1 (в начальный момент в него помещается любой набор чисел). На каждом шаге блуждания производится преобразование массива: образуется целое число



оно преобразуется в новое *u* = 7*t* + 11 (*mod* 4087), двоичное представление которого помещается в T. Направление движения определяется содержимым элементов T2, T7, T10 .

В условии задачи функция *f*(*x*, *y*) не дана, следовательно, предполагается, что функция может быть произвольной.

На этом условия задачи закончились, перейдем к решению.

Объяснение решения

С начала разберем условие поставленной задачи. Рассмотрим заданную нам плоскость с границей Г. Граница задана на графике в промежутке от 0 до 1 по каждой из осей, с ценой деления 0.1 . Обозначим горизонтальную ось ‘x’, а вертикальную ‘y’ . Так как операции с числами с плавающей точкой, выполняемые на компьютерах, в частности на языке C++, обладают погрешностью, а наши блуждания могут осуществлять большое количество шагов, тем самым накапливая ошибку, было бы целесообразно масштабировать рисунок на плоскость с размерами от 0 до 10 по осям x и y и ценой деления 1. Для того, чтобы избежать ошибок связанных с погрешностью вычисления.

Так как функция *f*(*x*, *y*) нам не задана в условии задачи, то мы можем выбирать ее как нам удобно. Я взял *f*(*x*, *y*) = x + y; , по нескольким причинам. Во первых, функция *u*(*x*, *y*) должна удовлетворять уравнению Лапласа Δ*u* = 0. Во вторых, так как основная цель задачи построить алгоритм вычисления функции *u*(*x*, *y*), том мы можем выбрать, как разработчик программы, такую функцию, чтобы проверка полученных результатов не занимала большое количество времени, тем самым упрощая процесс разработки программы.

Теперь рассмотрим счетчик блужданий ϕ*i* . Так как по условию задачи *u*(*xi*, *y*i) = *f*(*xi*, *y*i), где (*xi*, *y*i) принадлежат границе Г. А формула имеет вид



То при (*xi*, *y*i), принадлежащих границе Г, сумма ϕ*i* должна равняться *N* . Это будет выполняться только, если ϕ*i* будет начинаться с единицы(более подробное объяснение можно будет встретить дальше). Так же у нас присутствует проблема, когда в ходе блуждания точки “выпрыгивают” из области Г, так как перемещение происходит по всем 8 направления. В данном случае примем значение таких точек равным нулю. На это разбор условия задачи окончен.

Теперь разберем приступим к разбору программы. Моя программа состоит из 3 файлов: “Generator.h”, “Generator.cpp”, “main.cpp”.

“Generator.h” содержит подключаемые библиотеки: “vector”, “utility”, “exception” и объявления функций:

* “void generate\_vector(vector<bool>& v);”
* “int get\_direction(const vector<bool>& v);”
* “pair<double, double> get\_new\_position(pair<double, double> val, int direction, double h);”
* “bool check\_pos(pair<double, double> val);”

Библиотека “vector” подключается для получения доступа к контейнеру “vector” из стандартной библиотеки STL, который упрощает работу с массивами данных. Аналогично мы подключаем библиотеку “utility”, для получения доступа к контейнеру “pair”, с его помощью мы будем представлять и хранить координаты точек на нашей плоскости. Библиотека подключается для получения доступа к стандартным инструментам обработки ошибок, так как они могут возникать в случае неправильного пользовательского ввода.

Функция “void generate\_vector(vector<bool>& v);” используется для генерации нового массива Т из 12 элементов, заданного нам из условия. Генерация происходит согласно алгоритму, заданного нам в условии задачи.

Функция “int get\_direction(const vector<bool>& v);” используется для получения направления перемещения из массива Т. Мы берем элементы T2, T7, T10  формируем из них двоичное число и затем переводим его в десятичный вид(T2 \* 1 + T7 \* 2 + T10 \* 4), тем самим получая направление перемещения.

Функция “pair<double, double> get\_new\_position(pair<double, double> val, int direction, double h);” возвращает элемент класса “pair” с новыми координатами, полученными при перемещении начальных координат “val”, в направлении “direction”, на величину “h”. В реализации функции используется конструкция switch-case для перемещения координат.

Функция “bool check\_pos(pair<double, double> val);” проверяет принадлежит ли полученная точка координат границе Г, если принадлежит возвращает значение “true”, если нет возвращает значение “false”. Так же эта функция проверяет находиться ли эта точка внутри границы Г, в случае невыполнения данного условия, функция “бросает” объект класса “exception”, с строкой внутри “Out of range!”. Чтобы это проверить, я поделил заданную нам область на прямоугольные участки и в функции проверяю принадлежность данных нам координат к одному из участков.

“Generator.cpp” подключает файл “Generator.h” и содержит определения функций объявленных в этом файле. Так же он содержит массив со всеми точками границы Г: “vector<pair<double, double>> G”. Этот массив используется в функции “bool check\_pos(pair<double, double> val);” для проверки принадлежности границе Г.

“main.cpp” содержит подключения файла “Generator.h” и библиотеки “iostream”, благодаря которой, мы получаем доступ к консольному вводу-выводу. Так же он содержит функции:

* “pair<double, double> make\_wanding(pair<double, double> start, bool(\*check)(pair<double, double>),

int& fi,// fi возвращаемое значение

vector<bool>& T, double h)”

* “double make\_u(double (\*f)(double, double),

bool(\*check)(pair<double,double>),

pair<double, double> start, int N, vector<bool>& T, double h)”

* “int main()”

Функция “pair<double, double> make\_wanding(pair<double, double> start, bool(\*check)(pair<double, double>), int& fi, vector<bool>& T, double h)”

Осуществляет блуждание в нашей области и в результате своей работы возвращает точку границы Г, так же возвращаемым значением является переменная “int& fi”, которая передается по ссылке, она возвращает количество блужданий, если в результате работы функции “fi” стало равным нулю это означает, что в результате блуждания точка “выпрыгнула” за границы области. В качестве своих параметров функция принимает начальную точку блуждания “pair<double, double> start”, функция проверки “bool (\*check)(pair<double, double>)”, переменную “int& fi”, массив для генерации направления “vector<bool>& T” и величину перемещения “double h”.

Функция “double make\_u(double (\*f)(double, double), bool(\*check)(pair<double,double>), pair<double, double> start, int N, vector<bool>& T, double h)” формирует значение функции *u*(*x*, *y*). Она получает в качестве своих параметров функцию *f*(*x*, *y*) в виде указателя на функцию “double (\*f)(double, double)”, функцию проверки “bool (\*check)(pair<double, double>)”, начальную точку “pair<double, double> start”,

Число блужданий “int N”, массив Т “ vector<bool>& T” и величину перемещения “double h”. В результате свое работы функция возвращает число с плавающей точкой типа “double”, которое является значением функции *u*(*x*, *y*) при заданных (*x*, *y*), *f*(*x*, *y*), Г, T, N, h.

Если пользователь программы введет значение точки начала вне границы Г, ему выведется сообщение “Out of Range!”. В случае, если пользователь введет значение количества блужданий N меньше или равное нулю, ему будет выведено сообщение “ Invalid argument N”.

Функция “int main()” является главной функцией программы. Она содержит приглашение к вводу и вводу точки координат, для которой будет вычисляться значение функции *u*(*x*, *y*), и значения N. Так же она содержит вывод результата работы всей программы. На это объяснение работы программы подошло к концу.

Код программы

Файл “Generator.h”:

#pragma once

#include <vector>

#include <utility>

#include <exception>

using namespace std;

// Генерируем направление для блуждания

void generate\_vector(vector<bool>& v);

int get\_direction(const vector<bool>& v);

//Генерируем новую позицию

pair<double, double> get\_new\_position(pair<double, double> val, int direction, double h);

// Функция для проверки позиции

// Сначала проверяет входит ли в область определения Г

// Если входят осуществляет поиск

// Если не входят то бросает exception("Out of range!")

bool check\_pos(pair<double, double> val);

Файл “Generator.cpp”:

#include "Generator.h"

/////////////////////////////////////////////////////////////////////

//Формируем новый массив(который в задаче указан как массив T)

void generate\_vector(vector<bool>& v)

{

int u = 0;

int k = 1;

for (int i = 0; i < v.size(); ++i)//Происходит суммирование, формируем u

{

if (v[i]) u += k;

k \*= 2;

}

u = (u \* 7 + 11) % 4087;

for (int i = v.size() - 1; i > -1; --i)//Записываем двоичное представление в вектор

{

v[i] = u % 2;

u /= 2;

}

}

int get\_direction(const vector<bool>& v)

{

/\*

формируем число из битов под номерами 2,7,10

всего может получиться 8 чисел

т.е. то что нам и нужно

\*/

return v[1] + v[6] \* 2 + v[9] \* 4;

}

/////////////////////////////////////////////////////////////////////

pair<double, double> get\_new\_position(pair<double, double> val, int direction, double h)

{

switch (direction)

{

case 0:// x+h, y

{

val.first += h;

break;

}

case 1:// x+h, y-h

{

val.first += h;

val.second -= h;

break;

}

case 2:// x+h, y+h

{

val.first += h;

val.second += h;

break;

}

case 3:// x-h, y

{

val.first -= h;

break;

}

case 4:// x-h, y-h

{

val.first -= h;

val.second -= h;

break;

}

case 5:// x-h, y-h

{

val.first -= h;

val.second += h;

break;

}

case 6:// x, y-h

{

val.second -= h;

break;

}

case 7:// x, y+h

{

val.second += h;

break;

}

default:

throw exception("Invalid direction!");

}

return val;

}

/////////////////////////////////////////////////////////////////////

//Создаем рисунок в программе

vector<pair<double, double>> G = {

{4,0}, {3,1}, {3,2}, {3,3},

{2,3}, {1,3}, {0,4}, {0,5},

{0,6}, {1,7}, {2,7}, {2,8},

{1,9}, {2, 10}, {3,10}, {4,9},

{5,9}, {6,9}, {7,10}, {8,10},

{9,9}, {8,8}, {8,7}, {9,7},

{10,6}, {1,5}, {10,4}, {9,3},

{8,3}, {7,3}, {7,2}, {7,1},

{6,0}, {5,0}

};

bool check\_pos(pair<double, double> val)

{

double x = val.first, y = val.second;

// Проверяем на вхождение в область Г

if (

((2 <= x && x <= 8) && (3 <= y && y <= 9)) ||

((3 <= x && x <= 7) && (1 <= y && y <= 3)) ||

(((1 <= x && x <= 2) || (8 <= x && x <= 9)) && (3 <= y && y <= 4)) ||

(((2 <= x && x <= 3) || (7 <= x && x <= 8)) && (9 <= y && y <= 1)) ||

((4 <= x && x <= 6) && (0 <= y && y <= 1)) ||

(((0 <= x && x <= 1) || (9 <= x && x <= 10)) && (4 <= y && y <= 6)) ||

(((x == 1) || (x == 9)) && (y == 9))

)

{

auto res = find(G.begin(), G.end(), val); // Ищем координаты среди граничных точек

/\*

если мы точку не находим то find вернет G.end()

т.е. res == G.end()

если res != G.end() значит нашли

\*/

return res != G.end();

}

else // координаты находятся за областью

{

throw exception("Out of range!");

}

}

Файл “main.cpp”:

#include <iostream>

#include "Generator.h"

using namespace std;

pair<double, double> make\_wanding(pair<double, double> start,

bool (\*check)(pair<double, double>),

int& fi,// fi возвращаемое значение

vector<bool>& T, double h)

{

fi = 1;

int direction = 0;

try

{

while (!check(start))

{

fi++;

generate\_vector(T);

direction = get\_direction(T);

start = get\_new\_position(start, direction, h);

}

}

catch (exception& e)

{

fi = 0;

}

return start;

}

double make\_u(double (\*f)(double, double), bool (\*check)(pair<double,double>),

pair<double, double> start, int N, vector<bool>& T, double h)

{

if (N < 1)

throw exception("Invalid argument N");

if (check(start)) return f(start.first, start.second);

double result=0;

int fi = 0;

pair<double, double> temp;

for (int i = 0; i < N; ++i)

{

temp = make\_wanding(start,check, fi, T, h);

result += (fi \* f(temp.first, temp.second)) / N;

}

return result;

}

int main()

{

double x = 0, y = 0;

cout << "Enter x and y: ";

cin >> x>>y;

pair<double, double> temp(x, y);

int N = 0;

cout << "Enter N: ";

cin >> N;

vector<bool> T{ 1,1,1,1,1,1,1,1,1,1,1,1 };

try

{

double u = make\_u([](double x, double y) {return x + y; }, check\_pos, temp, N, T, 1);

cout <<"u( "<<x<<" , "<<y<<" ) = "<< u << endl;

}

catch (exception& e)

{

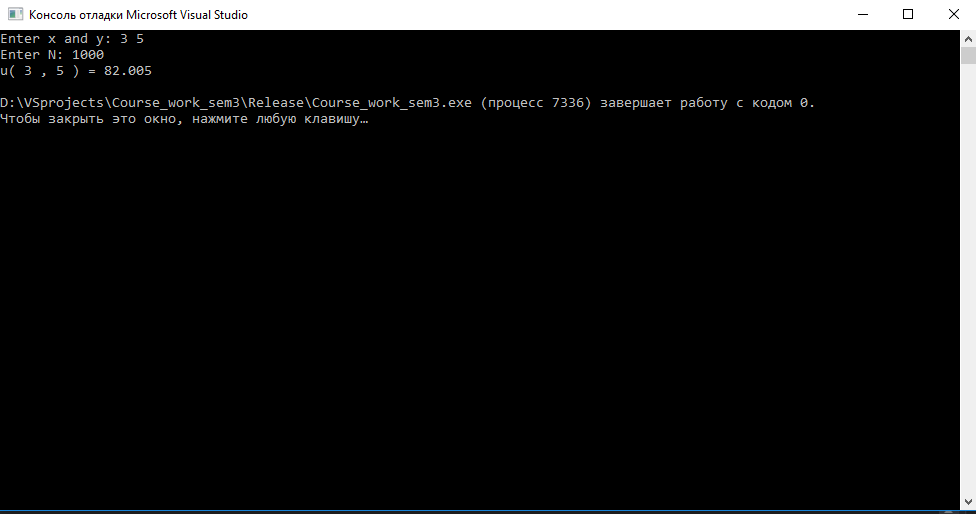
cout << e.what() << endl;

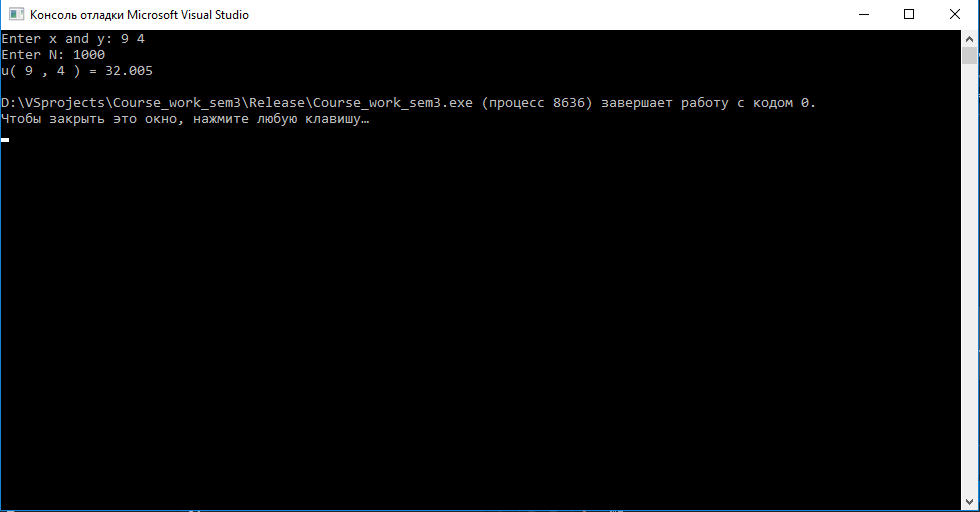
}

}

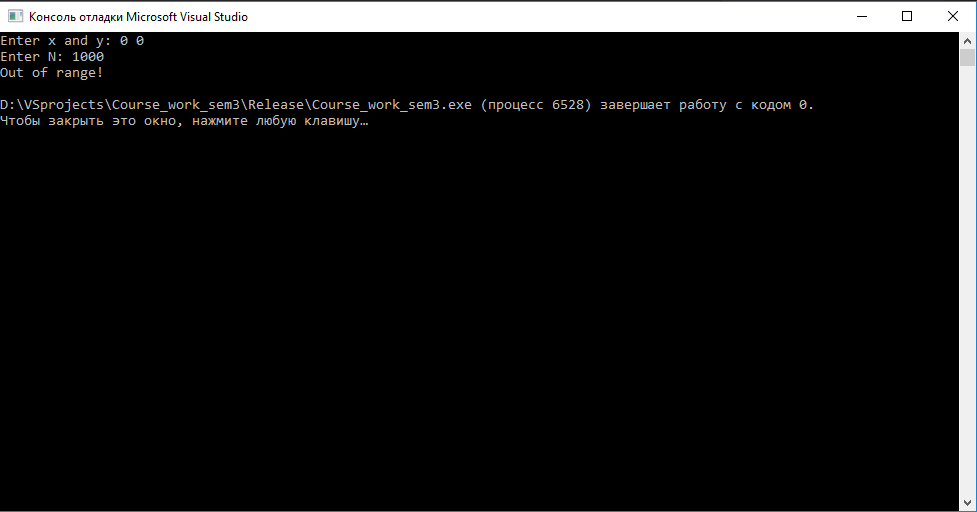
Примеры работы программы

1.



2.

3.



Вывод: в результате выполнения курсовой работы я ознакомился с методом Монте-Карло и применил его в практической задаче. Так же в процессе решения задачи, я применил полученные мною знание в течение курса.

Список литературы

1. Павловская Т.А. Программирование на языке высокого уровня. -- СПб.:Питер, 2007. - 461с.

2. С. Прата. Язык программирования C++. Лекции и упражнения. Учебник. Перевод с английского. 2012.

3. Бьерн Страуструп. Язык программирования С++

4. Бьерн Страуструп. Программирование: принципы и практика с использованием С++

5. Мартынов Н.Н. Программирование для Windows на C/C++. Том 1. - М.:ООО ``Бином-Пресс'', 2004г. -- 528 с.

6. Карпов Б., Баранова Т. С++. Специальный справочник (2-е издание). - СПб.:Питер,2005. - 381с.