Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования  
«Казанский (Приволжский) федеральный университет»

Институт вычислительной математики и информационных технологий

Кафедра прикладной математики



ОТЧЕТ  
по семестровой работе

по дисциплине «Теория вероятностей»

Вариант № 14

Выполнил: студент гр. 09-022

Тепляков Н.А.

Проверил: доц. С.В. Симушкин

Казань 2021

СОДЕРЖАНИЕ

[1 Цель работы: 3](#_Toc90825735)

[2 Исходные данные и задание: 3](#_Toc90825736)

[3 Ход работы 3](#_Toc90825737)

[3.1 Описание метода Монте-Карло 3](#_Toc90825738)

[3.2 Описание алгоритма 5](#_Toc90825739)

[3.3 Пошаговая реализация алгоритма 5](#_Toc90825740)

[3.4 Анализ полученного результата 7](#_Toc90825741)

[4 Заключение 8](#_Toc90825742)

[5 Список использованных источников 8](#_Toc90825743)

[6 ЛИСТИНГ 8](#_Toc90825744)

# 1 Цель работы:

Изучить метод Монте-Карло и с его помощью вычислить интеграл.

# 2 Исходные данные и задание:

C помощью метода случайного моделирования вычислить интеграл

где область с некоторой функцией .

; ; .

# 3 Ход работы

## 3.1 Описание метода Монте-Карло

где – площадь , и функция есть функция плотности равномерного на распределения. Другими словами, интеграл совпадает (с точностью до множителя ) с математическим ожиданием относительно случайного вектора с этим равномерным распределением:

Для приближённого вычисления математического ожидания применим закон больших чисел.

Теорема. Пусть – посл-ть независимых случайных векторов. Если , то при среднее арифметическое

Таким образом, для вычисления интеграла достаточно сгенерировать большое число равномерных случайных векторов и найти среднее арифметическое всех значений функции в этих точках.

Генерировать случайные точки в произвольной области сложно. Можно поступить следующим образом:

1. Охватить область минимально узкой простой областью – прямоугольником.

2. Как известно, равномерное распределение в прямоугольнике (со сторонами, параллельными осям), можно задать с помощью равномерного распределения каждой компоненты, т.е. если – реализации случайных величин с соответствующими равномерными распределениями (на сторонах прямоугольника), то пары – суть реализации случайных величин с равномерным распределением внутри прямоугольника .

3. Оставим в наборе чисел только те, которые попадают в , т.е. . Пусть их будет штук:

4. Среднее арифметическое

по закону больших чисел будет оценкой для .

5. Отношение также по закону больших чисел есть оценка для вероятности попадания в область . По определению равномерного распределения в вероятность попадания в любую область равна отношению площади области на площадь всего прямоугольника, т.е. - оценка площади .

6. Оценка для искомого интеграла равна

## 3.2 Описание алгоритма

1. Выбрать минимальный прямоугольник, который охватит данную область;
2. Сгенерировать в этом прямоугольнике случайные величины;
3. Отберём только те , которые попадают в , т.е. .
4. Найдём сумму в отобранных точках (обозначим: ).
5. Нахождение интеграла по формуле: .

## 3.3 Пошаговая реализация алгоритма

Для решения задачи реализуем программу на C++, с подключением библиотеки GLFW спецификации OpenGL (для вывода результатов на экран).

1. Выбрать минимальный прямоугольник, который охватит данную область. Для этого найдём максимум функции . = ; где . Данный максимум будет высотой прямоугольника.

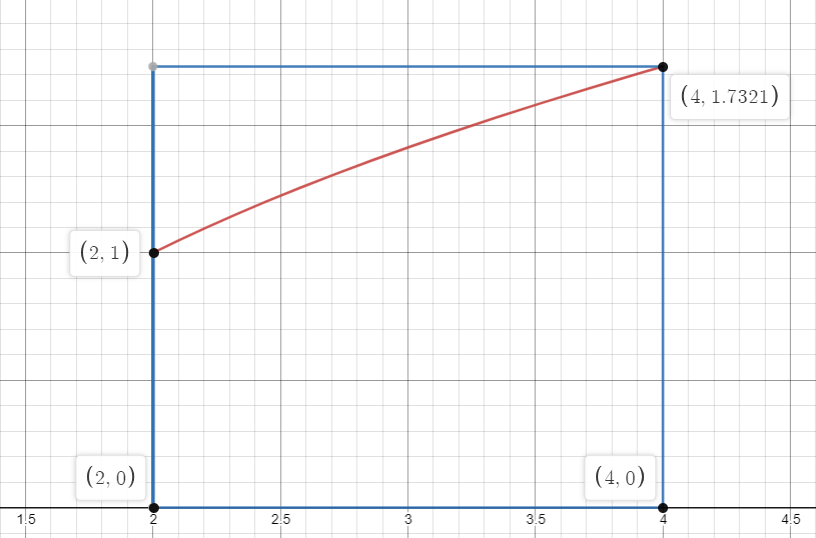


Рисунок 1 g(x) на [2;4] в среде desmos, ограниченная наименьшим прямоугольником

2.Сгенерируем случайные точки в границах прямоугольника, которые мы определили на первом шаге.

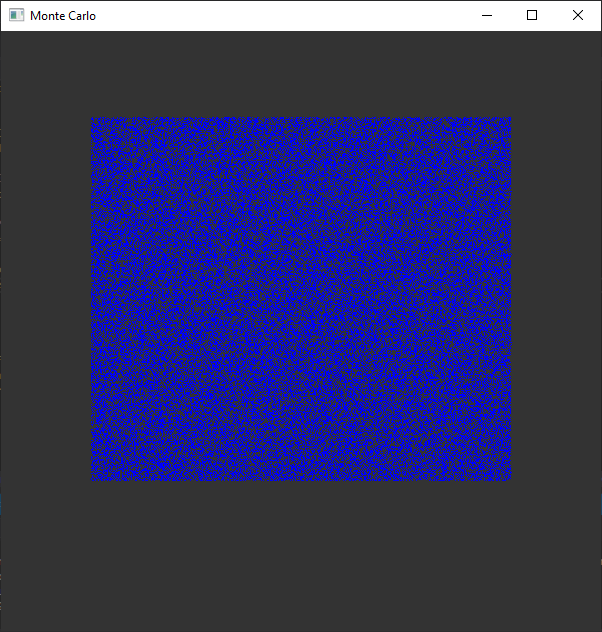


Рисунок 2 Сгенерировано 100000 точек

3. Сделаем отбор точек. Выберем только те, которые ниже нашей функции .

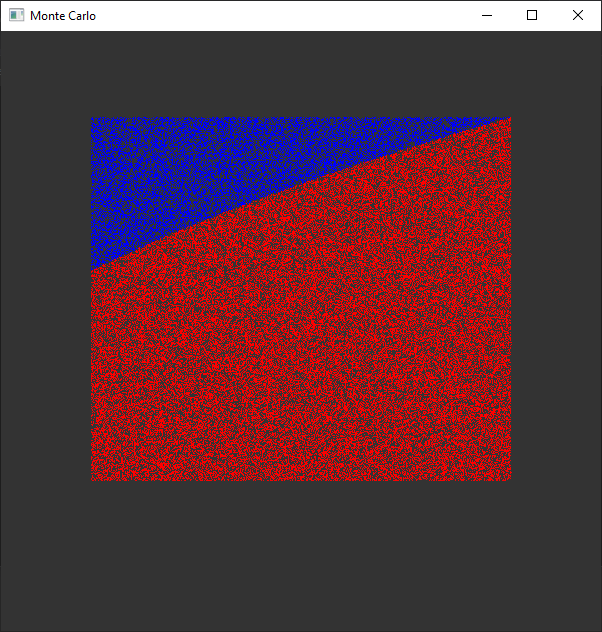


Рисунок 4 Произвели отбор (Красные точки - отобранные)

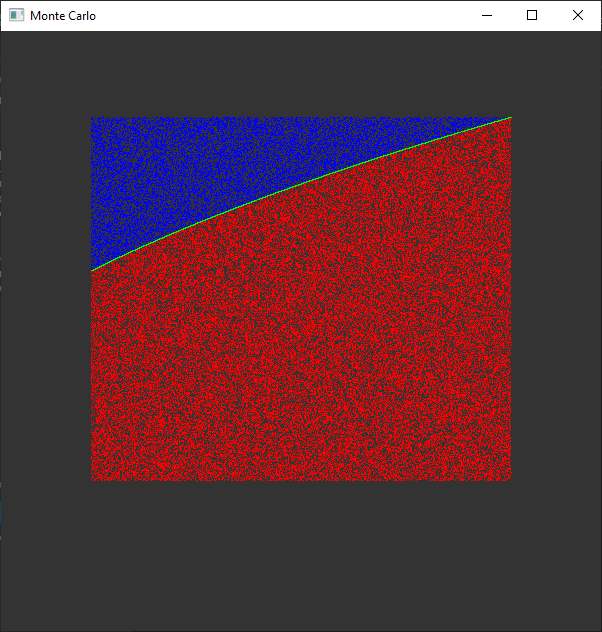


Рисунок 4 Для наглядности на рисунке отмечена функция g(x) зелёным цветом

4. Вычислим в каждой из отобранных точек и найдём их сумму.

Должен сделать замечание, что если вычислить функцию , то мы получим график поверхности. В моём случае .  
5. Вычислим интеграл по формуле: .

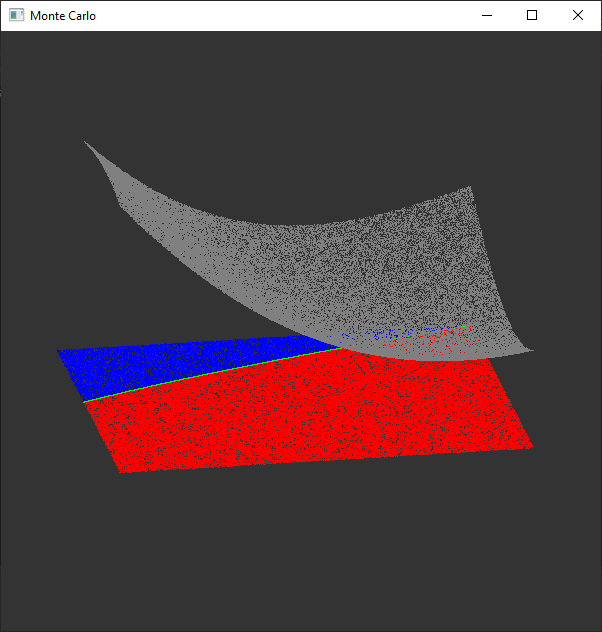


Рисунок 6 Поверхность h(x,y) ракурс №1

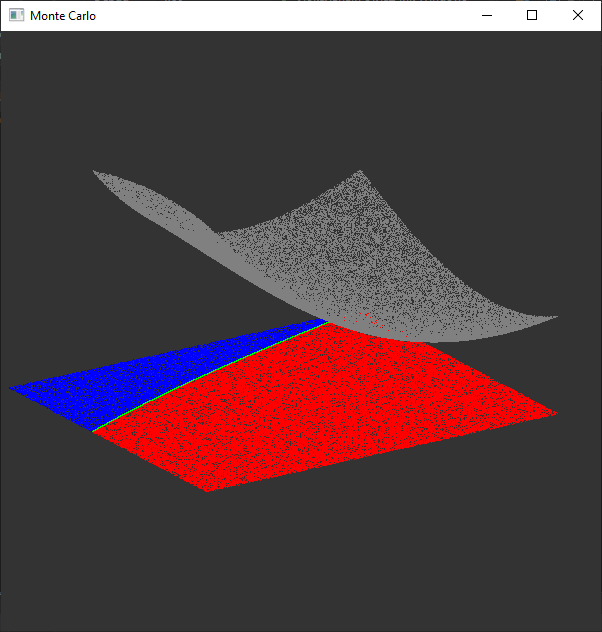


Рисунок 6 Поверхность h(x,y) ракурс №2

Мой результат -0.680754863698567 при 100000 точках.

## 3.4 Анализ полученного результата

При 100тыс.точек I = -0.680754863698567. Увеличим количество сгенерированных точек, чтобы повысить точность. Программа готова, поэтому проделывать всё с самого начала не нужно, стоит лишь поменять числовое значение в коде.

При 10млн. I = -0.678984922795595. Точность увеличилась, и она является вполне достаточной.

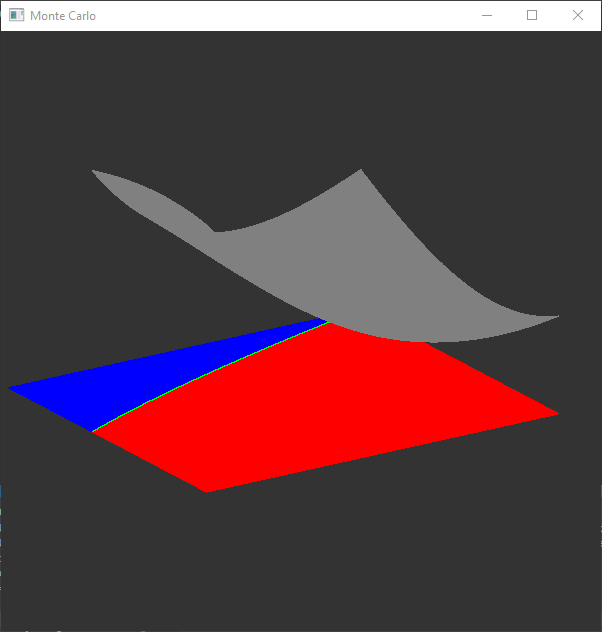


Рисунок 7 Поверхность h(x,y) при 10млн. точек

# 4 Заключение

Был изучен и применён на практике численный метод Монте-Карло. В ходе работы с помощью этого метода был вычислен заданный интеграл на отрезке. Были получены новые знания и навыки. Работа проделана без особых трудностей и сдана в срок.

# 5 Список использованных источников

1. ГОСТ Р 7.0.100-2018 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления».
2. ГОСТ 2.105-95 «ЕСКД. Общие требования к текстовым документам».
3. Соболь Илья Меерович. «Метод Монте-Карло» Год выпуска 1968
4. [Интернет ресурс №1](http://sewiki.ru/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BC_%D0%9C%D0%BE%D0%BD%D1%82%D0%B5-%D0%9A%D0%B0%D1%80%D0%BB%D0%BE)

# 6 ЛИСТИНГ

#include <vector>

#include "GLFW/glfw3.h"

#include <iostream>

#include "Functions.h"

int main()

{

int n = 1e7;

double S = 2.0 \* sqrt(3.0);

std::vector<double> index(n,0);

std::vector<double> x;

std::vector<double> y;

InitPoints(x, y, n);

SelectPoints(x, y, index, n);

double sum\_h = SumH(x, y, index, n);

double I = S/n\*sum\_h;

printf("%.15f\n", I);

#pragma region OpenGLDraws

GLFWwindow\* window;

if (!glfwInit())

return -1;

window = glfwCreateWindow(600, 600, "Monte Carlo", NULL, NULL);

if (!window)

{

glfwTerminate();

return -1;

}

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-0.1,0.1 ,-0.1,0.1,0.2,1000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glfwMakeContextCurrent(window);

while (!glfwWindowShouldClose(window))

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glRotatef(-70, 1, 0, 0);

glRotatef(20, 0, 0, 1);

glTranslatef(0,0,-0.28);

glPointSize(1.0);

glBegin(GL\_POINTS);

for (int i = 0; i < n; i++){

if (index[i] == 1) {

glColor3f(1, 0, 0);

glVertex2d(x[i] \* 0.7 - 2.1, y[i] \* 0.7 - 0.5);

}

else {

glColor3f(0, 0, 1);

glVertex2d(x[i] \* 0.7 - 2.1, y[i] \* 0.7 - 0.5);

}

}

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

{

glColor3f(0, 1, 0);

glVertex2d(x[i] \* 0.7 - 2.1, sqrt(x[i] - 1) \* 0.7 - 0.5);

}

glEnd();

glTranslatef(0, 0, 0.7);

glBegin(GL\_POINTS);

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

{

if (index[i] == 1) {

glColor3f(0.5, 0.5, 0.5);

glVertex3d(x[i] \* 0.7 - 2.1, y[i] \* 0.7 - 0.5, (sin(x[i] + y[i]) + cos(x[i] - y[i])) / (x[i] + y[i]));

}

}

glEnd();

glPopMatrix();

glfwSwapBuffers(window);

glfwPollEvents();

glClearColor(0.2, 0.2, 0.2, 1.0);

}

glfwTerminate();

#pragma endregion

}

Файл "Functions.h":

#pragma once

double SumH(std::vector<double>& x, std::vector<double>& y, std::vector<double>& index, int n);

void SelectPoints(std::vector<double>& x, std::vector<double>& y, std::vector<double>& index, int n);

void InitPoints(std::vector<double>& x, std::vector<double>& y, int n);

Файл "Functions.cpp":

#include <vector>

#include <iostream>

#include "Functions.h"

double SumH(std::vector<double>& x, std::vector<double>& y, std::vector<double>& index, int n) {

double expected\_val = 0;

for (int i = 0; i < n; i++) {

if (index[i] == 1) {

expected\_val += (sin(x[i] + y[i]) + cos(x[i] - y[i])) / (x[i] + y[i]);

}

}

return expected\_val;

}

void SelectPoints(std::vector<double>& x, std::vector<double>& y, std::vector<double>& index, int n) {

for (int i = 0; i < n; i++) {

double g\_x = sqrt(x[i] - 1);

if (y[i] <= g\_x) {

index[i] = 1;

}

}

}

void InitPoints(std::vector<double>& x, std::vector<double>& y, int n)

{

double sq = sqrt(3.0);

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

{

x.emplace\_back(((double)rand() / (double)RAND\_MAX) \* 2.0 + 2.0);

y.emplace\_back(((double)rand() / (double)RAND\_MAX) \* sq);

}

}